

tfarima: an R package to build customized TFARIMA models

José Luis Gallego
Universidad de Cantabria

Abstract

The R package **tfarima** provides classes and methods to build and use customized Transfer Function and ARIMA models with multiple operators and/or parameter restrictions. Model estimation can be accomplished by exact or conditional maximum likelihood (EML/CML). Procedures for automatic outlier detection, calendar effect estimation, prediction and seasonal adjustment based on this class general of models are also provided. Some classic time series are analyzed to illustrate the usage of the package.

Keywords: ARIMA models, transfer function models, prediction, outliers, R.

1. Introduction

The `arima()` function contained in the R **stats** package (R Core Team 2018) allows to estimate traditional multiplicative ARIMA(p,d,q)(P,D,Q)_s models with regression variables, sometimes called RegARIMA models. This function is extended by the wrapper `arimax()` function of the **TSA** package (Chan and Ripley 2018) to allow for transfer functions (TFs) and innovational outliers. Although these two functions are useful to analyze and forecast a great variety of time series, sometimes it is needed to specify ARIMA models with more than two ARIMA structures and/or with certain parameter constraints. The `um()` function of the **tfarima** package here presented extends the `arima()` and `arimax()` functions in these two directions to allow the estimation of univariate models (UMs) with multiple AR, I and MA operators, as well as with restrictions on the coefficients of these operators.

The general class of ARIMA(p,d,q) models that can be handled with the **tfarima** package is defined by the equation

$$\varphi_p(B)[\delta_d(B)z_t^{(\lambda)} - \mu] = \vartheta_q(B)a_t, \quad t = 1, 2, \dots, n; \quad (1)$$

where z_t is a time series of length n ; $z_t^{(\lambda)}$ is the power Box-Cox transformation $z_t^{(\lambda)} = (z_t^\lambda - 1)/\lambda$; $\varphi_p(B)$, $\delta_d(B)$ and $\vartheta_q(B)$ are the AR, I and MA operators, respectively, which are polynomials in the backshift operator B of degree p , d and q ; μ is the mean of the transformed time series $\delta_d(B)z_t^{(\lambda)}$, and a_t is a Gaussian white noise process with variance σ_a^2 . Each one of the three operators AR, I and MA can in turn be expressed as the product of several lag polynomials of the form

$$a_d(B^s)^p = (1 - a_1B^s - \dots - a_dB^{sd})^p, \quad (2)$$

which is a polynomial in B^s of degree d , raised to the integer power p , that is, a polynomial of order (d, s, p) . Note that, following the Box-Jenkins notation (Box, Jenkins, Reinsel, and Ljung 2015), the polynomial (2) is normalized to $a_d(1) = 1$ and the coefficients (a_1, \dots, a_d) are preceded by a minus sign. Such coefficients can be expressed as functions of a set of parameters (b_1, \dots, b_k) :

$$a_j = f_j(b_1, \dots, b_k), \quad j = 1, \dots, d \text{ and } k \leq d. \quad (3)$$

In this way, the **tarima** package allows us to specify customized ARIMA models with multiple operators and parameter restrictions.

By the same token, the impulse response function (IRF) $\nu(B) = \nu_0 + \nu_1 B + \nu_2 B^2 + \dots$ for an input X_t can be approximated by the rational transfer function

$$\nu(B) = w_0 \frac{w_s(B)}{\delta_r(B)} B^b, \quad (4)$$

where b is the delay or dead time, $w_0 = \nu_b$ is the first non-null coefficient of the IRF $\nu(B)$, and $\delta_r(B)$ and $w_s(B)$ are polynomials that, like the AR and MA operators of (1), can be expressed as the product of multiple normalized polynomials of the class (2)-(3).

Combining (1) and (4) we obtain a very flexible class of TF models for an output Y_t given by

$$Y_t = \sum_{i=1}^k w_{i0} \frac{\omega_{is_i}(B)}{\delta_{ir_i}(B)} B^{b_i} X_{it} + N_t \quad (5)$$

$$\delta_d(B) N_t = \mu + \frac{\vartheta_q(B)}{\varphi_p(B)} a_t,$$

where each input X_{it} has associated a $\text{TF}(r_i, s_i, b_i)$ of the class (4) and the noise N_t follows a general $\text{ARIMA}(p, d, q)$ model of the class (1). This specification contains as special cases the intervention models used to deal with four common types of outliers: additive outlier (AO), innovative outlier (IO), level shift (LS) and transitory change (TC), see, e.g., Box and Tiao (1975) and Chen and Liu (1993).

Unlike the **arma()** function of the **stats** package, which uses the state space representation of the ARIMA model suggested by Gardner, Harvey, and Phillips (1980), the (exact or conditional) maximum likelihood estimation of models (1) and (5) is based on the closed form of the inverse covariance matrix following the algorithm of Ljung and Box (1979). This procedure is also used to compute the exact residuals, which can differ from the innovations provided by the Kalman filter, see, e.g., Mauricio (2008). The covariance matrix for the estimated parameters is computed with the **numDeriv** package (Gilbert and Varadhan 2016) from the Jacobian matrix of partial derivatives of the scaled exact residuals, see, e.g., Ansley (1979).

Other capabilities of the **tfarima** package are the estimation of unobserved components based on the decomposition of the eventual forecast function proposed by Box, Pierce, and Newbold (1987), the automatic detection of outliers proposed by Chen and Liu (1993) and the tests for MA unit roots along the lines of Tam and Reinsel (1997).

To improve the computational efficiency, some functions of the **tfarima** package have been implemented in C++ with RStudio (RStudio Team 2020) using the **Rcpp** and **RcppArmadillo** packages (Eddelbuettel and Sanderson 2014).

This paper describes the main capabilities of the **tarima** package for R and is organized as follows. Section 2 outlines a class of factored seasonal ARIMA models, which are an example of customized ARIMA models. Sections 3, 4 and 5 describe and illustrate the usage of the S3 classes **lagpol**, **um** and **tfm** to specify and handle ARIMA and TF models with multiple operators and parameter restrictions. Section 5 contains a summary and some extensions in progress.

2. Factored seasonal ARIMA models

As an illustration of some customized ARIMA models that can be estimated with the **tfarima** package, we consider the generalized seasonal ARIMA models suggested by Gallego and Treadway (1995) and Aston, Findley, Wills, and Martin (2004). They extended the widely used airline ARIMA(0, 1, 1)(0, 1, 1)_s model,

$$(1 - B)(1 - B^s)z_t = (1 - \theta B)(1 - \Theta B^s)a_t, \quad (6)$$

noting that, for $s > 1$, the seasonal difference $1 - B^s$ and the so-called seasonal MA polynomial $1 - \Theta B^s$ admit two interesting factorizations, which here are referred to as the basic factorization

$$1 - \Theta B^s = (1 - \Theta^{1/s}B)(1 + \Theta^{1/s}B + \Theta^{2/s}B^2 + \dots + \Theta^{(s-1)/s}B^{s-1})$$

and the full factorization

$$1 - \Theta B^s = (1 - \Theta^{1/s}B)(1 + \Theta^{1/s}B) \prod_{k=1}^{[(s-1)/2]} (1 - 2\cos(2\pi k/s)\Theta^{1/s}B + \Theta^{2/s}B^2),$$

where $[x]$ is the integer part of x and the $(1 + \Theta^{1/s}B)$ factor only appears when s is even. Note that for $\Theta = 1$ we obtain the factorizations of $1 - B^s$. Hence, two generalizations of the model (6) arise: the basic generalization with three parameters

$$(1 - B)^2(1 + B + \dots + B^{s-1})z_t = (1 - \theta B)(1 - \Theta_0^{1/s}B)(1 + \Theta_1^{1/s}B + \dots + \Theta_1^{(s-1)/s}B^{s-1})a_t \quad (7)$$

and the full generalization with $[s/2] + 1$ parameters

$$(1 - B)^2(1 + B) \prod_{k=1}^{[(s-1)/2]} (1 - 2\cos(2\pi k/s)B + B^2)z_t = (1 - \theta B)(1 - \Theta_0^{1/s}B)[(1 + \Theta_{s/2}^{1/s}B) \prod_{k=1}^{[(s-1)/2]} (1 - 2\cos(2\pi k/s)\Theta_k^{1/s}B + \Theta_k^{2/s}B^2)]a_t. \quad (8)$$

It is clear that both generalizations are simply obtained by replacing the common parameter Θ in each factor of the seasonal polynomial $1 - \Theta B^s$ by a specific parameter Θ_i for each factor i , where $i = 1, 2, 3$ for the basic generalization and $i = 0, 1, \dots, [s/2]$ for the full generalization. Besides, both specifications involve nonlinear constraints on some coefficients of the MA polynomials. Thus, the MA($s - 1$) polynomial $1 - a_1B - \dots - a_{s-1}B^{s-1}$ in (7) is constrained so that its $s - 1$ coefficients $a_j = -\Theta_1^{j/s}$ ($j = 1, \dots, s - 1$) depend on a single

parameter Θ_1 . Similarly, the MA(2) polynomials $1 - a_{1j}B - a_{2j}B^2$ in (8) are constrained so that $a_{1j} = 2\cos(2\pi j/s)\Theta_j^{1/s}$ and $a_{2j} = -\Theta_j^{2/s}$. These special polynomials and other non-conventional polynomials defined by the user can be handled by the **tfarima** package. A general test for multiple MA unit roots is also being implemented, which can be used to evaluate the noninvertibility of these and other MA operators.

It is worth highlighting that these two factorizations of the airline model (6) are closely related to two popular structural time series models. The basic structural model (BSM) decomposes a time series Z_t into a sum of three unobserved components:

$$Z_t = \mu_t + \gamma_t + \epsilon_t.$$

where μ_t , γ_t and ϵ_t are the trend, seasonal and irregular components.

The trend component μ_t is described by the so-called local linear trend model

$$\begin{aligned}\mu_t &= \mu_t + \beta_{t-1} + \eta_t, & \eta_t &\sim iidN(0, \sigma_\eta^2), \\ \beta_t &= \beta_{t-1} + \zeta_t, & \zeta_t &\sim iidN(0, \sigma_\zeta^2),\end{aligned}$$

where the stochastic slope β_t follows a random walk process. The seasonal component can be modeled in two ways, as suggested by [Harvey and Todd \(1983\)](#),

$$\gamma_t = -\sum_{j=1}^{s-1} \gamma_{t-j} + w_t, \quad w_t \sim iidN(0, \sigma_w^2),$$

or by [Harvey and Durbin \(1986\)](#),

$$\gamma_t = \sum_{j=1}^{[s/2]} \gamma_{jt},$$

where

$$\begin{bmatrix} \gamma_{jt} \\ \gamma_{jt}^* \end{bmatrix} = \begin{bmatrix} \cos(2\pi k/s) & \sin(2\pi k/s) \\ -\cos(2\pi k/s) & \cos(2\pi k/s) \end{bmatrix} \begin{bmatrix} \gamma_{j,t-1} \\ \gamma_{j,t-1}^* \end{bmatrix} + \begin{bmatrix} \omega_{jt} \\ \omega_{jt}^* \end{bmatrix},$$

and ω_{jt} and ω_{jt}^* are both $iidN(0, \sigma_\omega^2)$. The reduced form of these two BSMs is an ARIMA(0,s+1,s+1) model

$$(1 - B)(1 - B^s)Z_t = \theta_{s+1}(B)a_t$$

where the coefficients $\theta_{s+1}(B)$ are subject to complicated restrictions. The `sum_um()` function of the **tfarima** allows to obtain the reduced form of structural time series by providing the ARIMA models of the unobserved components.

3. The lagpol class: lag polynomials

Lag polynomials of order (d, s, p) defined by equations (2)-(3) can be created with the `lagpol()` function of the **tfarima** package,

```
lagpol <- function(param = NULL, s = 1, p = 1, lags = NULL, coef = NULL)
```

where `param` is a numeric vector of named parameters, `coef` is a vector of expressions to compute the coefficients of the polynomial as functions of the parameters, `s` is the power of the backshift operator, `p` is the power to the which the polynomial is raised and `lags` is an optional integer vector for unequally spaced polynomials indicating the non-null coefficients. For example, to create and print the nonseasonal polynomial $(1 - \theta B)^2$ with $\theta = 0.5$, we run the following lines:

```
R> lp1 <- lagpol(param = c(theta = 0.5), p = 2L)
R> lp1
```

```
(1 - 0.5B)^2 = 1 - B + 0.25B^2
```

where `param = c(theta = 0.5)` is a single-parameter vector and `p = 2L` indicates that this lag polynomial is raised to the power of 2. Here, the default value for `coef = NULL` is equivalent to `coef = "theta"`. Note that lag polynomials are created following the Box-Jenkins notation. If the notation $(1 + \theta B)^2$ with $\theta = -0.5$ is preferred, the sign of the coefficients must be changed in the `coef` argument,

```
R> lp2 <- lagpol(param = c(theta = -0.5), p = 2L, coef = "-theta")
R> lp2
```

```
(1 - 0.5B)^2 = 1 - B + 0.25B^2
```

where `coef = "-theta"` would be the coefficient in $(1 - (-\theta)B)^2$.

The so-called seasonal polynomials are created in a similar way, but using the `s` argument to indicate the seasonal period. For example, the second-order seasonal polynomial $1 - \Theta_1 B^{12} - \Theta_2 B^{24}$ with $\Theta_1 = 1.2$ and $\Theta_2 = -0.9$ is created as follows:

```
R> lp3 <- lagpol(param = c(Theta1 = 1.2, Theta2 = -0.9), s = 12)
R> lp3
```

```
1 - 1.2B^12 + 0.9B^24
```

Customized lag polynomials with restrictions on the coefficients can be created passing to the `coef` argument a vector of strings with such restrictions. For example, to create the lag polynomial with three-coefficients and two-parameters $1 - \theta B - \Theta B^{12} + \theta \Theta B^{13}$, with $\theta = 0.8$ and $\Theta = 0.9$, we run the following sentence:

```
R> lp4 <- lagpol(param = c(theta = 0.8, Theta = 0.9),
+               coef = c("theta", "Theta", "-theta*Theta"),
+               lags = c(1, 12, 13))
R> lp4
```

```
1 - 0.8B - 0.9B^12 + 0.72B^13
```

where `lags = c(1, 12, 13)` contains the positions of the three non-null coefficients.

The `lagpol()` function also allows us to create the factors of the seasonal polynomial $1 - \Theta B^s$ described in Section 2. Some examples for $s = 12$ and $\Theta = 0.8$ are the following:

1. $1 + \Theta^{1/12}B + \Theta^{2/12}B^2 + \dots + \Theta^{11/12}B^{11}$. To create this polynomial we first create a character vector with the expressions of the coefficients and then we call the `lagpol()` function setting a single named parameter to the argument `param = c(Theta = 0.8)` and a vector of expressions to the argument `coef = coef`:

```
R> coef <- paste("-Theta^(", 1:11, "/12)", sep = "")
R> coef
```

```
[1] "-Theta^(1/12)" "-Theta^(2/12)" "-Theta^(3/12)"
[4] "-Theta^(4/12)" "-Theta^(5/12)" "-Theta^(6/12)"
[7] "-Theta^(7/12)" "-Theta^(8/12)" "-Theta^(9/12)"
[10] "-Theta^(10/12)" "-Theta^(11/12)"
```

```
R> lp5 <- lagpol(param = c(Theta = 0.8), coef = coef)
R> lp5
```

```
1 + 0.98B + 0.96B^2 + 0.95B^3 + 0.93B^4 + 0.91B^5 + 0.89B^6 + 0.88B^7
+ 0.86B^8 + 0.85B^9 + 0.83B^10 + 0.82B^11
```

2. $1 - \Theta_0^{1/12}B$ and $1 + \Theta_6^{1/12}B$:

```
R> lp6 <- lagpol(param = c(Theta0 = 0.8), coef = "Theta0^(1/12)")
R> lp7 <- lagpol(param = c(Theta6 = 0.8), coef = "-Theta6^(1/12)")
R> printLagpolList(list(lp6, lp7))
```

```
[1] 1 - 0.98B [2] 1 + 0.98B
```

3. $1 - 2 \cos(2\pi/12)\Theta_k^{1/12}B + \Theta_k^{2/12}B^2$ for $k = 1, \dots, 5$:

```
R> coef <- c("2*cos(2*pi/12)*Theta1^(1/12)", "-Theta1^(2/12)")
R> lp8 <- lagpol(param = c(Theta1 = 0.8), coef = coef)
R> lp8
```

```
1 - 1.7B + 0.96B^2
```

Here we first create a vector with the expressions of the coefficients for the factor of frequency 1 and then create the corresponding lag polynomial. We can use the `lapply()` function to create all the MA(2) factors,

```
R> lp9 <- lapply(1:5, function(k) {
+   th <- paste("Theta", k, sep = "")
+   param <- 0.8
+   names(param) <- th
+   coef1 <- paste("2*cos(2*pi*", k, "/12)*", th, "^(1/12)", sep = "")
+   coef2 <- paste("-", th, "^(2/12)", sep = "")
+   lagpol(param = param, coef = c(coef1, coef2))
+ })
R> printLagpolList(lp9)
```

```
[1] 1 - 1.7B + 0.96B^2    [2] 1 - 0.98B + 0.96B^2    [3] 1 - 1.2e-16B +
0.96B^2    [4] 1 + 0.98B + 0.96B^2    [5] 1 + 1.7B + 0.96B^2
```

Integrated operators such as nonseasonal and seasonal differences and other simplifying factors, which do not contain any parameter, can be created as follows:

1. $(1 - B)^2$

```
R> d2 <- lagpol(coef = "1", p = 2)
R> d2
```

```
(1 - B)^2 = 1 - 2B + B^2
```

2. $1 - B^{12}$

```
R> D <- lagpol(coef = "1", s = 12)
R> D
```

```
1 - B^12
```

3. $1 - 2 \cos(2\pi * k/12)B + B^2$ for $k = 1, \dots, 5$

```
R> sf <- lapply(1:5, function(k) {
+   coef1 <- paste("2*cos(2*pi*", k, "/12)", sep = "")
+   coef2 <- "-1"
+   lagpol(coef = c(coef1, coef2))
+ })
R> printLagpolList(sf)
```

```
[1] 1 - 1.7B + B^2    [2] 1 - B + B^2    [3] 1 - 1.2e-16B + B^2    [4] 1
+ B + B^2    [5] 1 + 1.7B + B^2
```

It is convenient to note that the `lagpol()` function returns an S3 object of class `lagpol`, which is a building block to create ARIMA and TF models. Some useful methods of the `lagpol` class are `roots()` and `inv()`.

```
R> class(D)
```

```
[1] "lagpol"
```

```
R> roots(sf[[1]])
```

	Real	Imaginary	Modulus	Frequency	Period	Mult.
[1,]	0.8660254	0.5	1	0.08333333	12	1
[2,]	0.8660254	-0.5	1	0.08333333	12	1

```
R> as.lagpol(inv(d2, lag.max = 9))
```

```
1 + 2B + 3B^2 + 4B^3 + 5B^4 + 6B^5 + 7B^6 + 8B^7 + 9B^8 + 10B^9
```

4. The um class: univariate models

Univariate ARIMA models belonging to the class defined by (1) can be created using the `um()` function of the **tfarima** package,

```
um <- function(z = NULL, bc = FALSE, ar = NULL, i = NULL, ma = NULL,
              mu = NULL, sig2 = 1.0, fit = TRUE)
```

where `z` is an object of class `ts`; `bc` is a logical value that indicates whether or not to take logs; `ar`, `i` and `ma` are lists of objects of class `lagpol`, `character` and/or `numeric` representing lag polynomials; `mu` is a numeric value for the mean of the stationary series $\delta(B)z_t^{(\lambda)}$; `sig2` is a numeric value for the variance of the error, and `fit` is a logical value indicating whether or not to estimate the model. The function returns an object of class `um`.

To illustrate the usage of this function we create a multiplicative ARIMA model with orders $(0, 1, 1)(0, 1, 1)_3(0, 1, 1)_{12}$, $\nabla \nabla_{12} z_t = (1 - \theta_1 B)(1 - \theta_2 B^3)(1 - \theta_3 B^{12})a_t$ where $\theta_1 = \theta_2 = \theta_3 = 0.8$ and $\sigma_a^2 = 1$. We can set the arguments `ar`, `i` and `ma` providing three types of objects:

1. Lists of objects of class `lagpol`:

```
R> d1 <- lagpol(coef = 1)
R> d3 <- lagpol(coef = 1, s = 3)
R> d12 <- lagpol(coef = 1, s = 12)
R> ma1 <- lagpol(param = c(th1 = 0.8))
R> ma3 <- lagpol(param = c(th2 = 0.8), s = 3)
R> ma12 <- lagpol(param = c(th3 = 0.8), s = 12)
R> um1 <- um(i = list(d1, d3, d12), ma = list(ma1, ma3, ma12))
```

In this code chunk, we first create six `lagpol` objects, one for each operator of the model, and then we pass them as lists to the arguments of the `um` function. This approach is recommended to create customized models since it provides a way to specify names, preestimates and restrictions for parameters.

2. Character strings with the equations of the polynomials:

```
R> um2 <- um(i = "(1 - B)(1 - B3)(1 - B12)",
+           ma = "(1 - .8B)(1 - .8B3)(1 - .8B12)")
```

Here we provide to each argument `i` and `ma` the equations of the operators as character strings. This way of creating a model is more compact than the previous one, but doesn't allow us neither to set the names of the parameters nor create customized models. It is useful to study the statistical properties of common ARMA models (Section 4.1).

3. Lists containing the orders `c(d, s = 1, p = 1)` of the operators:

```
R> um3 <- um(i = list(1, c(1, 3), c(1, 12)),
+           ma = list(1, c(1, 3), c(1, 12)))
```

In this case, we only provide the orders of each type of operator and the function sets the names and values of the parameters: `.1` for AR parameters, `.2` for MA parameters and `1` for I coefficients. Hence, `ar = list(1, c(1, 3), c(1, 12))` and `ma = list(1, c(1,`

$3)$, $c(1, 12))$ are equivalent to $ar = "(1 - .1B)(1 - .1B3)(1 - .1B12)"$ and $ma = "(1 - .2B)(1 - .2B3)(1 - .2B12)"$, whereas $i = list(1, c(1, 3), c(1, 12))$ is equivalent to $ma = "(1 - B)(1 - B3)(1 - B12)"$. This approach is convenient to fit conventional models to time series.

We can combine these three types of objects to set the arguments `ar`, `i` and `ma`:

```
R> um4 <- um(i = list(1, "1 - B3", d12), ma = list("1 - .8B", c(1, 3), ma12))
```

Each list of operators AR, I and MA in a `um` object, and the corresponding extended polynomials, are stored into the data members `ar`, `i` and `ma`, `phi`, `nabla` and `theta`.

```
R> printLagpolList(um1$i)
```

```
[1] 1 - B    [2] 1 - B^3    [3] 1 - B^12
```

```
R> printLagpolList(um1$ma)
```

```
[1] 1 - 0.8B    [2] 1 - 0.8B^3    [3] 1 - 0.8B^12
```

```
R> nabla(um1)
```

```
1 - B - B^3 + B^4 - B^12 + B^13 + B^15 - B^16
```

```
R> theta(um1)
```

```
1 - 0.8B - 0.8B^3 + 0.64B^4 - 0.8B^12 + 0.64B^13 + 0.64B^15 - 0.51B^16
```

Note that we have created the object `um1` without providing a time series, which is useful, for example, to study the statistical properties of a particular model, to simulate data or to fit the same model to several time series. These and other capabilities for objects of class `um` are explained in the next sections.

4.1. Statistical properties

We can create an object of class `um` to study the characteristics of a particular ARMA model in terms of the simple and partial autocorrelation functions (ACF and PACF) and the spectral density function. The following code creates five basic time series models and displays their main functions (see Figure 1):

```
R> ar1p <- um(ar = "(1 - 0.9B)")
R> ar1n <- um(ar = "(1 + 0.9B)")
R> ma1p <- um(ma = "(1 - 0.9B)")
R> ma1n <- um(ma = "(1 + 0.9B)")
R> ar2c <- um(ar = "(1 - 1.52B + 0.8B^2)")
R> display(list(ar1p, ar1n, ma1p, ma1n, ar2c), lag.max = 20)
```

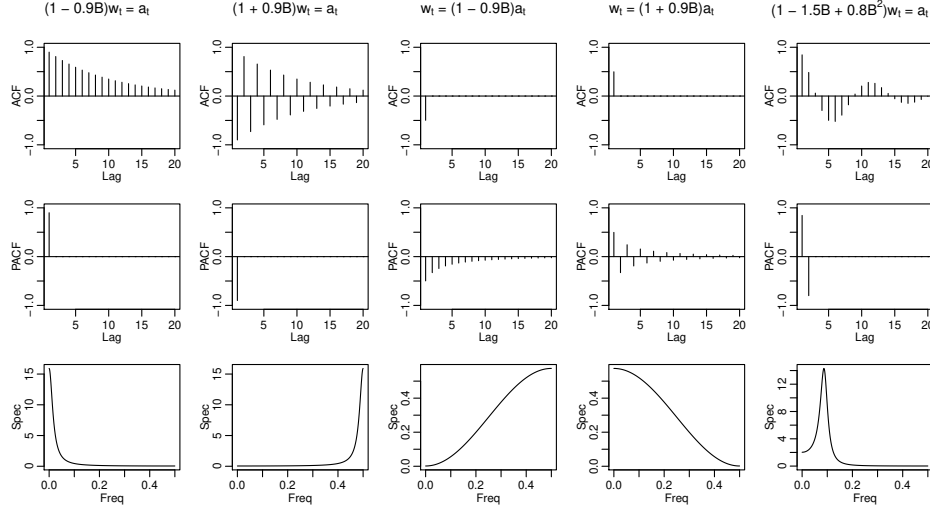


Figure 1: ACF, PACF and spectrum of five UM's.

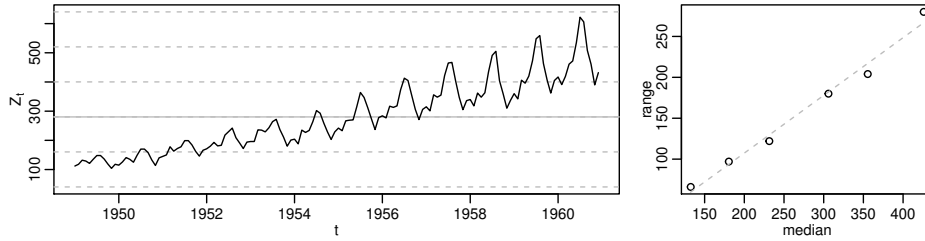


Figure 2: Plot and range-median diagram for the monthly AirPassenger series.

The `display()` function shows the ACF, PACF and/or spectrum of a list of objects of class `um`. Other methods of the `um` class to characterize stochastic processes are `pi.weights()` and `psi.weights()`, which show the $AR(\infty)$ and $MA(\infty)$ forms; `autocov()` and `autocorr()`, which compute the theoretical autocovariances and (simple/partial) autocorrelations, and `roots()`, which computes the roots of the different polynomials of the model.

4.2. Model identification

Figure 2 shows the plot and the range-median diagram for the totals of international airline passengers listed as Series G in Box *et al.* (2015). These graphs have been generated with the `ide()` function of the `tfarima` package:

```
R> Z <- AirPassengers
R> ide(Z, graphs = c("plot", "rm"))
```

where the option `rm` in the `graphs` argument means range-median.

The `ide()` function can also show several graphs for a list of transformations. Figure 3 displays the plot, ACF and PACF for the transformed series: $\log(Z_t)$, $(1 + B + \dots + B^{11}) \log(Z_t)$ and $\nabla \nabla_{12} \log(Z_t)$. The `transf` argument allows us to set a list of lists of transformations such as

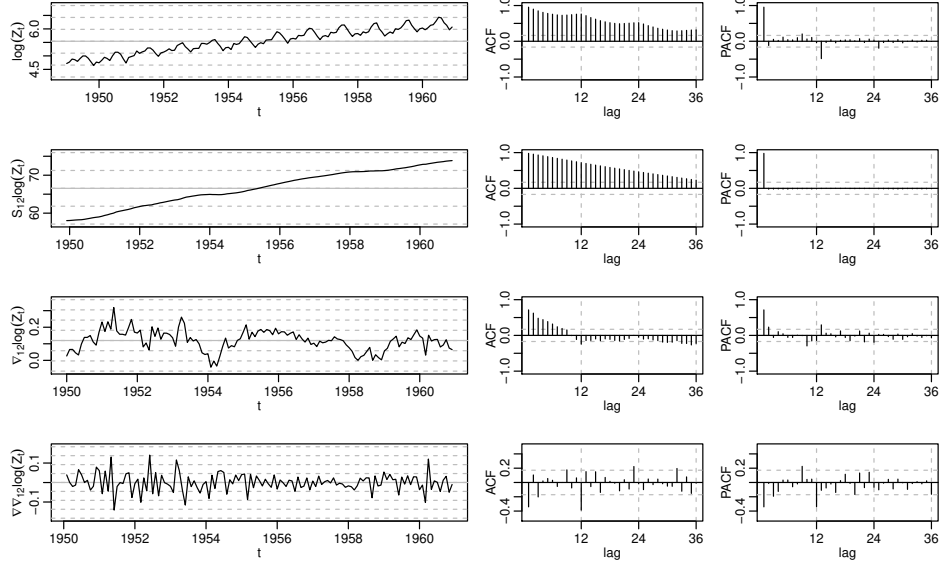


Figure 3: Some identification tools for three data transformations of AirPassengers

the Box-Cox transformation (bc), the number of nonseasonal and seasonal differences (d and D), the annual sum (S) or a list of `lagpol` objects with the integrated operators (i).

```
R> ide(Z, transf = list(list(bc = T), list(bc = T, S = 1),
+                       list(bc = T, D = 1), list(bc = T, D = 1, d = 1)))
```

4.3. Model estimation

Box *et al.* (2015) fit a multiplicative ARIMA(0,1,1)(0,1,1)₁₂ to the `log(AirPassengers)` series. This model can be estimated with the `um` function as follows:

```
R> airl <- um(Z, bc = T, i = list(1, c(1, 12)), ma = list(1, c(1, 12)))
R> airl
```

```
      theta1      theta2      sig2
0.401846366 0.557039272 0.001348078
```

The basic generalization of this model given in (7) is estimated by running the command:

```
R> bairl <- um(Z, bc = T, i = list(1, c(1, 12)), ma = list(1, "0/12", "12"))
```

where "0/12" means factor of frequency 0/12, $1 - \Theta^{1/12}B$, and "12" means $1 + \Theta^{1/12}B + \Theta^{2/12}B^2 + \dots + \Theta^{11/12}$. Such factors can also be created as described in Section 2. We can see in this factorized model that there is a unit MA root not detected in the conventional ARIMA model:

```
R> bairl
```

```

      theta1      theta2      theta3      sig2
0.378636665 0.999813862 0.539476315 0.001307118

```

```

R> #bairl$ma
R> printLagpolList(bairl$ma)

```

```

[1] 1 - 0.38B   [2] 1 - B   [3] 1 + 0.95B + 0.9B^2 + 0.86B^3 +
0.81B^4 + 0.77B^5 + 0.73B^6 + 0.7B^7 + 0.66B^8 + 0.63B^9 + 0.6B^10 +
0.57B^11

```

Similarly, the full generalization of the airline model given in (eq:fgam) is estimated by running the command:

```

R> fairl <- um(Z, bc = T, i = list(1, c(1, 12)), ma = list(1, "(0:6)/12"))

```

where "(0:6)/12" means factors with roots at the frequencies $0/12, 1/12, \dots, 6/12$.

```

R> fairl

```

```

      theta1      theta2      theta3      theta4      theta5
0.449259930 0.998903194 0.449990440 0.443816228 0.999011827
      theta6      theta7      theta8      sig2
0.537213429 0.626780620 0.985501321 0.001182939

```

```

R> printLagpolList(fairl$ma)

```

```

[1] 1 - 0.45B   [2] 1 - B   [3] 1 - 1.6B + 0.88B^2   [4] 1 - 0.93B +
0.87B^2   [5] 1 - 1.2e-16B + B^2   [6] 1 + 0.95B + 0.9B^2   [7] 1 +
1.7B + 0.93B^2   [8] 1 + B

```

4.4. Model diagnostic checking

Detailed results of the estimation together with diagnostic statistics can be printed with the `summary()` function:

```

R> summary(bairl)

```

Model:

```

bairl <- um(z = Z, i = list(1, c(1, 12)), ma = list(1, "0/12", "12"), bc = T)

```

Time series:

```

Z

```

Maximum likelihood method:

```

exact

```

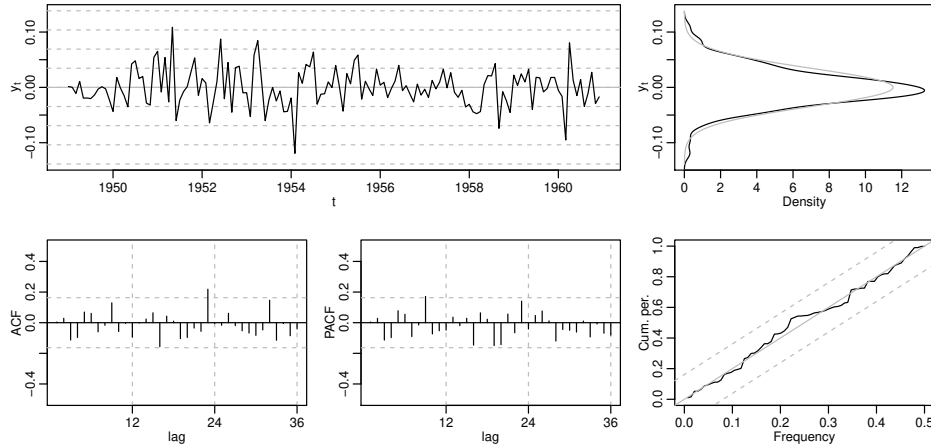


Figure 4: Some diagnostic tools for the residuals of model bairl

Coefficients:

	Estimate	Gradient	Std. Error	z Value	Pr(> z)
theta1	3.786e-01	-8.223e-06	7.893e-02	4.797	1.61e-06 ***
theta2	9.998e-01	-2.111e-06	2.567e-01	3.895	9.81e-05 ***
theta3	5.395e-01	3.416e-06	7.635e-02	7.066	1.59e-12 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Total nobs	144	Effective nobs	131
log likelihood	245.8	Error variance	0.001307
Mean of residuals	-1.348e-06	SD of the residuals	0.0346
z-test for residuals	-0.0004674	p-value	0.9996
Ljung-Box Q(1) st.	9.982	p-value	0.00158
Ljung-Box Q(32) st.	46.39	p-value	0.04801
Barlett H(3) stat.	1.326	p-value	0.5154
AIC	-3.707	BIC	-3.641

whose argument is an object of class `um`. Some diagnostic graphs for this object can be shown with the `diagchk()` function (see Figure 4):

```
R> diagchk(bairl)
```

4.5. Multiple seasonalities

ARIMA models to forecast time series with multiple seasonalities can also be estimated with the `um()` function. As an illustration we consider the half-hourly electricity demand data in England and Wales from Monday 5 June 2000 to Sunday 27 August 2000 analyzed by Taylor (2003) and provided by the **forecast** package. This series exhibits daily and weekly seasonality and can be described by a multiplicative $\text{ARIMA}(0, 1, 1)(0, 1, 1)_{48}(0, 1, 1)_{336}$ model,

$$\nabla \nabla_{48} \nabla_{336} z_t = (1 - \theta_1 B)(1 - \theta_2 B^3)(1 - \theta_3 B^{336}) a_t.$$

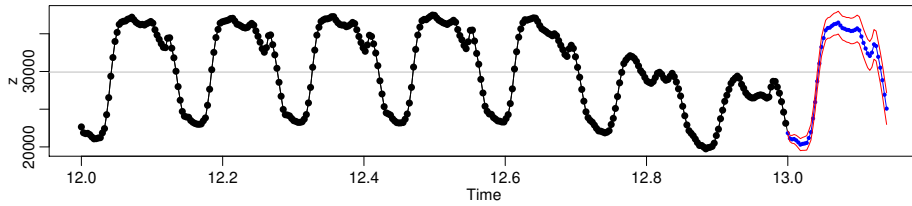


Figure 5: Point and interval forecasts for electricity demand.

We can estimate this model by running the sentences:

```
R> library(forecast)
R> E <- taylor
R> um.E <- um(E, i = list(1, c(1, 48), c(1, 336)),
+           ma = list(1, c(1, 48), c(1, 336)),
+           method = "cond")
```

For this kind of models it is recommended to use CML as first estimation method given that the exact method requires to invert a 336×336 matrix. Once we have good preestimates, we could reestimate the model by EML. Next, we report these estimates:

```
R> #um.E <- fit(um.E, method = "exact")
R> um.E$param[["theta1"]] <- -6.551086e-02
R> um.E$param[["theta2"]] <- 7.000628e-01
R> um.E$param[["theta3"]] <- 7.206957e-01
R> um.E$sig2 <- 2.180469e+04
```

4.6. Forecasting

Point and interval forecast from a `um` object can be computed with the `predict()` function, whose arguments `n.ahead` and `level` allow to set the lead time and the confidence levels. The object returned by this function can be displayed with the `plot()` function, whose `n.back` argument sets the number of previous observations to be shown. The following script computes 48 forecasts (one day) and displays a plot with the last 336 observations (seven days) followed by the 48 forecasts:

```
R> p <- predict(um.E, n.ahead = 48)
R> plot(p, n.back = 336)
```

4.7. Unobserved components

The `ucomp()` function computes unobserved components for a time series based on an object of class `um`. For example, the half-hourly electricity data can be decomposed by running

```
R> p <- ucomp(um.E)
R> plot(p)
```

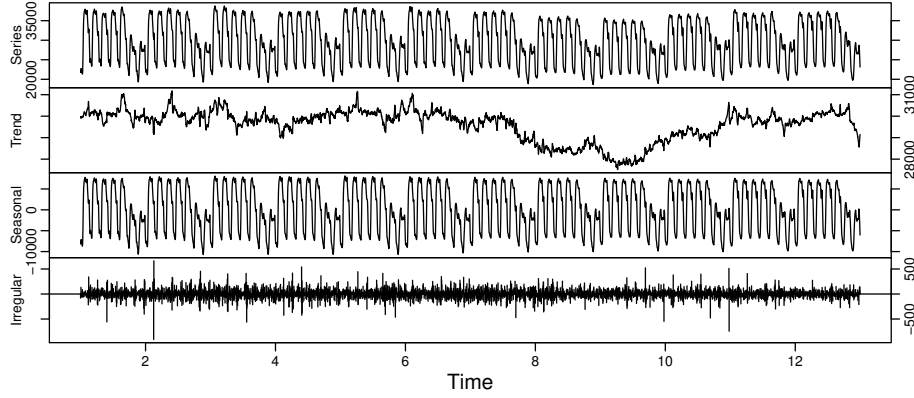


Figure 6: Unobserved components for electricity demand.

This function performs a decomposition of the eventual forecast function of a `um` object following a similar approach to that described by [Box *et al.* \(1987\)](#). The starting point is the fundamental decomposition of a time series:

$$z_t = \hat{z}_{t-1}(1) + a_t$$

where $\hat{z}_{t-1}(1)$ is the forecast of z_t from origin $t - 1$. For a general ARIMA(p,d,q) model

$$\hat{z}_t(l) = b_1^{(t)} f_1(l) + \dots + b_{p+d}^{(t)} f_{p+d}(l), \quad l \geq p + d - q,$$

where the functions $f_j(t)$ are determined by the roots of the AR and I polynomials and the adaptive coefficients $b_j^{(t)}$ can be determined from $p + q$ forecasts. Such roots can be classified into three categories: real unit roots (trend), complex unit roots (seasonality) and real/complex non-unit roots (cycle). The irregular component coincides with the residuals of the model. Since the `ucomp()` returns matrices with the values of $f_j(t)$ and $b_j^{(t)}$ it is possible to estimate the unobserved components according to other criteria.

5. The `tfm` class: transfer function models

To create TF models defined by equation (5), we have to create previously both the TF for each input and the UM for the noise. The `tf()` function of the `tfarima` package specifies the TF for a particular input,

```
tf <- function(x = NULL, delay = 0, w0 = 0.1, ar = NULL, ma = NULL,
              um = NULL, par.prefix = "")
```

where the input `x` is an object of class `"ts"`, `delay` is an integer value for the input lag, `w0` a numeric value for the parameter w_0 of the TF, `ar` and `ma` are lists of objects of class `lagpol` in the denominator (AR) and numerator (MA) of the TF, `um` is an optional object of class `um` used to back/forecast the input `x`, and `par.prefix` is an optional character to set a prefix for the parameters of the TF. Any of the three ways described to provide lag polynomials in the `um()` function can also be used with the `tf()` function.

Once we have specified a TF for each input, we can create a TF model with the `tfm()` function,

```
tfm <- function(output = NULL, xreg = NULL, inputs = NULL, noise, fit = TRUE)
```

where `output` is an object of class `ts`, `xreg` is a vector or matrix of regressors, `inputs` is a list of objects of class `tf`, `noise` is an object of class `um` with the UM for the noise, `fit` is a logical value indicating whether or not to fit the TF model. Two comments are in order: (1) if the `um` object for the output is set to the `noise` argument, then it is not necessary provide the `output` argument because it is already contained in this `um` object; (2) time series for the inputs and regressors must have at least the same length as the output, but they can be extended with backcasts and forecasts to improve the estimation or to forecast the output (more on that later).

The function returns an object of class `tfm`, whose main data members are `xreg`, `inputs` and `noise`. Some useful methods for these of objects are: `noise()`, `fit()`, `diagchk()`, `calendar()`, `outliers()`, `predict()` and `ucomp()`, which are illustrated in the following subsections.

5.1. Calendar effects

Hillmer (1982) used the telephone data of Thompson and Tiao (1971) to illustrate how to forecast time series with trading day variation. This data refers to outward station movements (disconnections) of the Wisconsin telephone company from January 1951 to October 1968. Clearly the series exhibits increasing seasonality and an upward trend requiring logs and both types of differences. Besides, it is affected by the number of trading days in a month because there are more disconnections on weekdays than on weekends. Due to these calendar effects, the sample ACF and PACF of $\nabla\nabla \log(Y_t)$ do not have a recognizable pattern. Hence, to identify a tentative ARMA structure for this series, the calendar effects are removed by fitting the regARIMA model

$$\log(Z_t) = \sum_{j=1}^7 \alpha_j X_{jt} + N_t$$

$$\nabla\nabla_{12} N_t = a_t,$$

where $X_{jt}, i = 1, \dots, 7$, are, respectively, the number of Mondays, Tuesdays, and so on in month t and N_t is the noise. Defining $\beta_0 = \sum_{j=1}^7 \alpha_j / 7$ and $L_t = X_{1t} + \dots + X_{7t}$ (length of month t), the regression component $TD_t = \sum_{j=1}^7 \alpha_j X_{jt}$ can be reparametrized as

$$TD_t = \beta_0 L_t + \sum_{j=1}^6 \beta_1 (X_{jt} - X_{7t})$$

where $\beta_j = \alpha_j - \beta_0$ for $j = 1, \dots, 6$ (see, e.g., Bell and Hillmer 1983).

The `calendar()` function enlarges an ARIMA or a TF model by adding calendar variables:

```
calendar <- function(x, form = c("dif", "td"), easter = FALSE, n.ahead = NULL)
```

where `x` is an object of class `um` or `tfm`, `form` is a character indicating the type of representation for TD_t , `easter` is a logical value to include an extra calendar variable for Easter, and `n.ahead` is an optional integer to extend the regressors with future observations to forecast the output.

To estimate this regARIMA model, we create an $\text{ARIMA}(0,1,0)(0,1,0)_{12}$ model without parameters and call the `calendar()` function:

```
R> Y <- Wtelephone$Y
R> umY <- um(Y, bc = TRUE, i = list(1, c(1,12)))
R> tfmY <- calendar(umY, n.ahead = 13)
R> tfmY
```

Lom	Mon_Sun	Tue_Sun	Wed_Sun	Thu_Sun
-0.025279555	0.023436662	0.024087935	0.011829553	0.009641849
Fri_Sun	Sat_Sun	sig2		
0.004062339	-0.047702989	0.003895825		

Alternatively, we could estimate this model generating the calendar variables and creating a `tfm` object:

```
R> X <- CalendarVar(Y, form = "lom", n.ahead = 13)
R> tfmY <- tfm(xreg = X, noise = umY)
```

where the `Y` argument in the `CalendarVar()` function is used to get the sample period.

We can recovery the noise N_t or its transformations $\exp(N_t)$ and $\nabla\nabla_{12}N_t$ with the function

```
noise <- function(tfmY, diff = TRUE, exp = FALSE)
```

where `diff` is a logical value indicating if the noise must be differenced and `exp` is a logical value indicating if the antilog must be applied to the non-differenced noise. Note that these transformations are determined by the noise model.

Now we can identify a pattern in the ACF and PACF of the corrected series compatible with an $\text{ARIMA}(0,1,1)(0,1,1)_{12}$:

```
R> y <- noise(tfmY, diff = FALSE, exp = TRUE)
R> ide(list(Y, y), transf = list(bc = T, d = 1, D = 1))
```

The two MA operators can be added to the `tfmY` object with the `modify()` function:

```
R> tfmY <- modify(tfmY, ma = list(1, c(1,12)))
R> tfmY
```

Lom	Mon	Tue	Wed	Thu
-0.042666547	0.049018049	0.048648330	0.044885508	0.022852854
Fri	Sat	theta1	theta2	sig2
0.039365675	-0.022720778	0.742669684	0.456118447	0.001933313

Finally, we could predict future values:

```
R> p <- predict(tfmY, n.ahead = 13)
R> print(p, rows = c(1, 13))
```

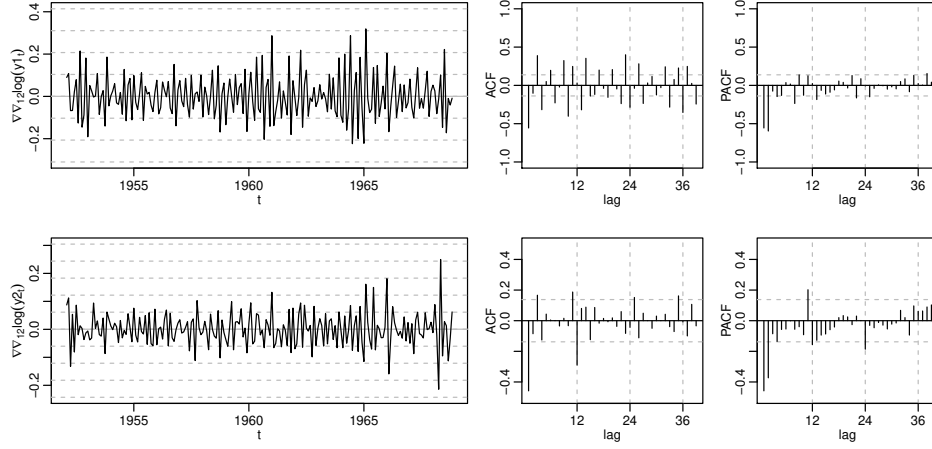


Figure 7: Identification tools for original and corrected telephone data.

	Forecast	RMSE	95% LB	95% UB
dic. 1968	18482.07	0.04396946	16956.02	20145.48
dic. 1969	20113.77	0.06769510	17614.54	22967.60

For the sake of completeness, we estimate the ARIMA model developed by [Thompson and Tiao \(1971\)](#):

$$(1 - \phi_1 B^3)(1 - \phi_2 B^{12}) \log(Y_t) = (1 - \theta_9 B^9 - \theta_{12} B^{12} - \theta_{13} B^{13}) a_t$$

```
R> ma <- lagpol(c(theta9 = .2, theta12 = .2, theta13 = .2),
+               lags = c(9, 12, 13))
R> TT <- um(Y, bc = TRUE, ar = list(c(1, 3), c(1, 12)), ma = ma)
R> TT
```

```
      phi1      phi2      theta9      theta12      theta13
0.941588020 0.995575838 -0.235380779 0.319588129 0.033347043
      sig2
0.004822835
```

5.2. Intervention analysis and outlier detection

Intervention models suggested by [Box and Tiao \(1975\)](#) are special cases of TF models and can be estimated with the `tfm()` function to deal with outliers. For example, [Box et al. \(2015\)](#) identify three innovational outliers (IO) at times 58, 59 and 60 in the Series C, the “uncontrolled” temperature readings every minute in a chemical process. To estimate the effects of these IOs they fit the outlier model:

$$(1 - B)Y_t = \frac{1}{1 - \phi B} (w_1 P_t^{(58)} + w_2 P_t^{(59)} + w_3 P_t^{(60)} + a_t),$$

where $P_t^{(\tau)}$ is a pulse function at τ :

$$P_t^{(\tau)} = \begin{cases} 0 & t \neq \tau \\ 1 & t = \tau \end{cases}$$

They estimated the model by conditional least squares (CLS) and obtained the following results (standard errors in parenthesis): $\hat{\phi} = 0.851(0.035)$, $\hat{\omega}_1 = 0.745(0.116)$, $\hat{\omega}_2 = -0.551(0.120)$, $\hat{\omega}_1 = -0.455(0.116)$, $\hat{\sigma}_a^2 = 0.0132$. This model can be reformulated as a single input transfer function model:

$$Y_t = w_0 \frac{(1 - w'_1 B - w'_2 B^2)}{(1 - \phi B)(1 - B)} P_t^{(58)} + N_t \quad (9)$$

$$N_t = \frac{1}{(1 - \phi B)(1 - B)} a_t,$$

where the input and the noise share the same ARI operators. The capabilities of the **tfarima** package to estimate models with multiple operators and parameter restrictions allows us to estimate this model by running the following code:

```
R> Y <- as.ts(seriesC)
R> um1 <- um(Y, ar = 1, i = 1, method = "cond")
R> P58 <- InterventionVar(Y, 58)
R> tf58 <- tf(P58, ma = 2, ar = c(um1$ar, um1$i))
R> tfm1 <- tfm(inputs = tf58, noise = um1)
R> tfm1
```

P58	P58.w1	P58.w2	phi1	sig2
0.74473341	0.74054615	0.61106025	0.85123172	0.01389304

```
R> printLagpol(tfm1$inputs[[1]]$theta, digits = 3)

0.745 - 0.552B - 0.455B^2
```

Firstly, we load the Series C and fit an ARIMA(1,1,0) model, which is used as noise model. Next, we create a pulse variable at $\tau = 58$ and its transfer function by providing the order of the MA polynomial and the ARI operators of the **um1** model. Finally, we create and fit the transfer function model which is made up of a single input and an ARIMA(1,1,0) noise. We can see that the results of the estimation by conditional likelihood maximum are very similar to those of [Box et al. \(2015\)](#).

The **outliers()** function implements a version of the [Chen and Liu \(1993\)](#) procedure to detect and correct the effects of four common types of anomalies: additive outliers, innovational outliers, level shifts and transitory changes. We apply this function to model **um1** and use a critical value $c = 3.5$ to determine if an observation is anomalous:

```
R> tfm2 <- outliers(um1, c = 3.5)
R> tfm2
```

LS58	I060	phi1	sig2
0.70372279	-0.45606236	0.85354122	0.01391037

The function detects two outliers at times 58 and 60 of type LS and IO, respectively. Note that model `tfm2` is compatible with model `tfm1` since (9) can be reformulated as

$$Y_t = w_0 \frac{(1 - w'_1 B)}{(1 - \phi B)} S_t^{(58)} - w_0 * w'_2 P_t^{60} + N_t,$$

where $S_t^{50} = (1 - B)^{-1} P_t^{58}$ is a step function at $T = 58$ and $w'_1 \simeq \phi$.

The `outliers()` function can also be used to identify and estimate the effects of possible outliers at known times:

```
R> tfm3 <- outliers(um1, dates = c(58, 59, 60))
R> summary(tfm3)$table
```

	Estimate	Gradient	Std. Error	z Value	Pr(> z)
LS58	0.74464605	-3.695249e-07	0.1183491	6.2919457	3.135112e-10
LS59	0.08264826	6.578909e-07	0.1558444	0.5303256	5.958862e-01
I060	-0.38500574	-3.937869e-07	0.1782898	-2.1594375	3.081624e-02
phi1	0.85116889	-1.583515e-06	0.0355065	23.9721977	5.423579e-127

We can see that the possible outlier at time $\tau = 59$ is identified as a LS but it is not significant.

5.3. Building transfer function models

To illustrate the Box-Jenkins approach to the identification, fitting and checking of TF models, we replicate the analysis of the gas furnace data by [Box et al. \(2015\)](#), Series J. They identify and fit an AR(3) model for the input X_t , which can be also fitted to the output Y_t :

```
R> Y <- seriesJ$Y - mean(seriesJ$Y)
R> X <- seriesJ$X - mean(seriesJ$X)
R> umx <- um(X, ar = 3)
R> umy <- fit(umx, Y)
```

This `umx` model is used to prewhiten the input X and the output Y . The `residuals()` function of the `tfarima` package compute the conditional or exact residuals for a time series from an object of class `um`:

```
R> a <- residuals(umx, Y, method = "cond")
R> b <- residuals(umx, X, method = "cond")
```

Now we can use the `ccf()` function of the `stats` package to display the estimated cross correlation function for the gas furnace data after filtering. Alternatively, we can use the `pccf()` function of the `tfarima` defined as

```
R> pccf(X, Y, um.x = umx, um.y = NULL, lag.max = 16)
```

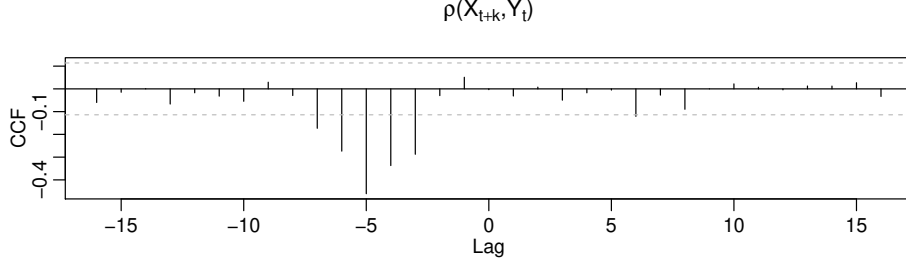


Figure 8: Estimated cross correlation function.

where x and y are the input and the output, `um.x` and `um.y` are the univariate models used to prewhiten both series, and `lag.max` is the number of correlation in each side. If both `um.x` and `um.y` are equal to `NULL`, the `ccf` is estimated without prewhitening; if only one of these arguments is equal to `NULL`, the `um` provided is used to prewhiten both series; otherwise, each series is prewhitenned by its own UM.

As in [Box et al. \(2015\)](#) we identify a TF with orders $(1, 2, 3)$ or $(2, 2, 3)$, (see Figure 8). Preestimates of the parameters of the TF can be computed with the `tfest()` function

```
R> tfx <- tfest(Y, X, delay = 3, p = 2, q = 2, um.x = umx, um.y = umy)
```

where we provide the output and the input, the delay, the orders of the AR and MA lag polynomials of the TF, and the UMs for the input and the output. The extended lag polynomials of the TF are stored into the data members `theta` and `phi`.

```
R> printLagpol(tfx$theta)
```

```
- 0.51 - 0.32B - 0.48B^2
```

```
R> printLagpol(tfx$phi)
```

```
1 - 0.65B + 0.087B^2
```

[Box et al. \(2015\)](#) fitted the following TF model:

$$Y_t = \frac{w_0 - w_1 B - w_2 B^2}{1 - \delta_1 B - \delta_w B^2} X_t + N_t,$$

$$N_t = \frac{1}{1 - \phi_1 B - \phi_2 B^2} a_t,$$

which can be estimated as follows:

```
R> tfmy <- tfm(Y, inputs = tfx, noise = um(ar = 2))
```

```
R> printLagpol(tfmy$inputs[[1]]$theta)
```

```
- 0.53 - 0.37B - 0.51B^2
```

```
R> printLagpol(tfmy$inputs[[1]]$phi)
```

```
1 - 0.57B + 0.012B^2
```

```
R> printLagpol(tfmy$noise$phi)
```

```
1 - 1.5B + 0.63B^2
```

where we can see that the estimates are very close to those reported by Box *et al.* (2015): $\hat{\omega}_0 = -0.53$, $\hat{\omega}_1 = 0.33$, $\hat{\omega}_2 = 0.51$, $\hat{\delta}_1 = 0.57$, $\hat{\delta}_2 = 0.02$, $\hat{\phi}_1 = 1.54$, $\hat{\phi}_2 = -0.64$.

It is worth mentioning that the `tfm()` function use the backcasting method to extend the input using its own UM so that the effect of transients can be minimized. The number of backcasts is controlled with the `n.back`, by default it is set to `n/4`. The UM of the input is also used to compute forecasts when forecasting the output.

6. Summary

We have presented some of the capabilities offered by the **tfarima** package to build customized TF and ARIMA models, which can include multiple conventional and user-defined lag polynomials. The package provides a full set of functions to apply the three stages of the Box-Jenkins methodology: identification, estimation and diagnosis of TF-ARIMA models, which can be used to forecast and decompose time series. Currently the package is been improved to include tests for a diversity of MA unit roots. A multivariate version of the package is also being developed.

References

- Ansley CF (1979). “An algorithm for the exact likelihood of a mixed autoregressive-moving average process.” *Biometrika*, **66**(1), 59–65. URL <https://doi.org/10.1093/biomet/66.1.59>.
- Aston J, Findley D, Wills K, Martin D (2004). “Generalizations of the Box-Jenkins’ Airline Model with Frequency-Specific Seasonal Coefficients and a Generalization of Akaike’s MAIC.” In *Proceedings of the 2004 NBER/NSF Time Series Conference*. <http://www.census.gov/ts/papers/findleyner2004.pdf>.
- Bell WR, Hillmer SC (1983). “Modeling Time Series With Calendar Variation.” *Journal of the American Statistical Association*, **78**(383), 526–534. ISSN 01621459. URL <http://www.jstor.org/stable/2288114>.
- Box GEP, Jenkins GM, Reinsel GC, Ljung GM (2015). *Time Series Analysis: Forecasting and Control*. 5th edition. John Wiley and Sons Inc., Hoboken, New Jersey. ISBN 978-1-118-67502-1.
- Box GEP, Pierce DA, Newbold P (1987). “Estimating Trend and Growth Rates in Seasonal Time Series.” *Journal of the American Statistical Association*, **82**(397), 276–282.

- Box GEP, Tiao GC (1975). “Intervention Analysis with Applications to Economic and Environmental Problems.” *Journal of the American Statistical Association*, **70**(349), 70–79. URL <https://www.tandfonline.com/doi/abs/10.1080/01621459.1975.10480264>.
- Chan KS, Ripley B (2018). *TSA: Time Series Analysis*. R package version 1.2, URL <https://CRAN.R-project.org/package=TSA>.
- Chen C, Liu LM (1993). “Joint Estimation of Model Parameters and Outlier Effects in Time Series.” *Journal of the American Statistical Association*, **88**(421), 284–297. URL <https://doi.org/10.1080/01621459.1993.10594321>.
- Eddelbuettel D, Sanderson C (2014). “RcppArmadillo: Accelerating R with high-performance C++ linear algebra.” *Computational Statistics and Data Analysis*, **71**, 1054–1063. URL <http://dx.doi.org/10.1016/j.csda.2013.02.005>.
- Gallego J, Treadway A (1995). “The general family of seasonal stochastic processes.” Departamento de Economía, Universidad de Cantabria.
- Gardner G, Harvey AC, Phillips GDA (1980). “Algorithm AS 154: An Algorithm for Exact Maximum Likelihood Estimation of Autoregressive-Moving Average Models by Means of Kalman Filtering.” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **29**(3), 311–322. ISSN 00359254, 14679876. URL <http://www.jstor.org/stable/2346910>.
- Gilbert P, Varadhan R (2016). *numDeriv: Accurate Numerical Derivatives*. R package version 2016.8-1, URL <https://CRAN.R-project.org/package=numDeriv>.
- Harvey AC, Durbin J (1986). “The effects of seat belt legislation on British road casualties: a case study in structural time series modelling.” *JRSSA*, **149**(4), 187–227.
- Harvey AC, Todd P (1983). “Forecasting Economic Time Series with Structural and Box-Jenkins Models: A Case Study.” *JBES*, **1**(4), 299–307.
- Hillmer SC (1982). “Forecasting time series with trading day variation.” *Journal of Forecasting*, **4**(1), 385–95.
- Ljung GM, Box GEP (1979). “The likelihood function of stationary autoregressive-moving average models.” *Biometrika*, **66**(2), 265–270. URL <https://doi.org/10.1093/biomet/66.2.265>.
- Mauricio J (2008). “Computing and using residuals in time series models.” *Computational Statistics & Data Analysis*, **52**, 1746–1763. doi:10.1016/j.csda.2007.05.034.
- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- RStudio Team (2020). *RStudio: Integrated Development Environment for R*. RStudio, PBC., Boston, MA. URL <http://www.rstudio.com/>.
- Tam W, Reinsel G (1997). “Tests for seasonal moving average unit root in ARIMA models.” *JASA*, **92**(438), 725–738.

Taylor JW (2003). “Short-Term Electricity Demand Forecasting Using Double Seasonal Exponential Smoothing.” *The Journal of the Operational Research Society*, **54**(8), 799–805. ISSN 01605682, 14769360. URL <http://www.jstor.org/stable/4101650>.

Thompson H, Tiao GC (1971). “Analysis of Telephone Data: A Case Study of Forecasting Seasonal Time Series.” **2**, 515–541.

Affiliation:

José Luis Gallego
Department of Economics
Faculty of Economics and Buisness
Universidad de Cantabria
Avda. de los Castros s/n
39005 Santander, Spain
E-mail: jose.gallego@unican.es