ECOTOOL Manual

Diego J. Pedregal

February 22, 2019

Contents

1 ECOTOOL overview

ECOTOOL is MATLAB toolbox that embodies several routines for identification, validation and forecasting of dynamic models based on time series analysis. The toolbox includes a wide range of exploratory, descriptive and diagnostic statistical tools with visual support, designed in easy-to-use Graphical User Interfaces (GUI). It also incorporates complex automatic procedures of identification, exact maximum likelihood estimation and outlier detection for many types of models available in the literature (like multi-seasonal ARIMA models, transfer functions, Exponential Smoothing, Unobserved Components, VARX). **ECOTOOL** is the outcome of a long period of programming effort with the aim of producing a user friendly toolkit such that, just a few lines of code written in MATLAB are able to perform a comprehensive analysis of time series. The toolbox is supplied with a comprehensive documentation system and online help and is available on the internet. This manual describes the main functionalities of the toolbox, and its power is shown working on several real examples.

ECOTOOL is user oriented in the sense that the coding effort demanded from the user is reduced to a minimum at the cost of the programmer elaborating long and comprehensive functions. The result is that it is possible to carry out an exhaustive analysis of time series with the recourse to a few functions. The main ones are listed on Table 1 in separated sections showing the GUI and demo tools, model functions, functions for the generation of calendar effects dummies and general purpose functions. The real number of functions in **ECOTOOL** is actually much greater because all the available options in the menus of the `toolTEST` GUI are actually implemented as separate functions that may be run independently, see the documentation. Some explanations about this functions are included in the next paragraphs and the way to use them is illustrated in the worked examples below.

The main features of **ECOTOOL** are:

- A number of time series methods are implemented, ranging from simple naïve univariate models to multivariate methods.
- Extensive and detailed documentation is available. All functions are provided with a thorough help accessible in the usual way. Eleven detailed demos with extensive explanations that show all the properties of the toolbox are also provided (accessed by `ECOTOOLdemos`).

Table 1: Main **ECOTOOL** functions.

	GUI and demos
`toolTEST`	Exploratory, descriptive and diagnostic checking tool
`toolFORECAST`	Forecasting tool
`ECOTOOLdemos`	ECOTOOL demos
	Modeling functions
`modelAUTO`	Automatic identification of ARIMA models
`modelETS`	Exponential Smothing models
`modelNAIVE`	Forecasting with several naive models
`modelTF`	MISO Transfer Function analysis
`modelUC`	Unobserved Components models
`modelVARX`	VAR model with eXogenous variables analysis
	Calendar effects
`days`	Dummy variable for number of days in months or quarters
`easter`	General dummy Easter variables on monthly or quarterly data
`leapy`	Dummy variable for leap year intervention
`trading`	Trading day variables on monthly or quarterly frequency
	General purpose functions
`acft`	Theoretical autocorrelation functions of ARMA processes
`varstep`	Var impulse and step function analysis
`vboxcoxinv`	Inverse of Box-Cox transformation
`vConv`	Multiplication of vector polynomials
`vdif`	Differenciation of a vector of variables
`vfilter`	Filters a vector of inputs with a vector digital filter
`vRoots`	Roots of a vector polynomial

- The design of the toolbox is such that it is possible to perform a full time series analysis with just a few MATLAB instructions. In this way, the memory effort demanded from the user is reduced to a minimum. In addition, function names are selected following mnemonic rules such that they are easy to remember and easy to look for.

- The toolbox is composed of four types of functions, see Table 1: i) GUI tools to perform several tasks that are named as tool* (where '*' stands for a name, at the moment two are available, `toolTEST` and `toolFORECAST`, see below); ii) Modeling methods that are named as model* (like `modelNAIVE`, `modelTF`, etc.); iii) functions that facilitates building dummy variables to deal with calendar effects; and iv) general purpose functions to perform a number of important tasks when dealing with time series analysis, like transforming the data (standardising, differencing, Box-Cox variance transforming, etc.) or doing other tasks (filtering and differencing vector time series, calculating convolutions of multivariate polynomials, calculating roots of multivariate polynomials, etc.).

- Specification of models is rather simple and flexible. All functions for estimation of models, i.e., model* functions, are written with a common syntax in order to make the model specification task easier to the user. Besides, in the case of `modelTF` for TF or ARIMA model estimation and forecasting, the way the models are specified is in fact very similar to its analytical expression according to equations (1) and (2). For example, the MATLAB code for specifying an ARIMA$(0, 1, 2)$ is '`(1+ma1*B+ma2*B2)/(1-B)`', where `ma1` and `ma2` stand for two arbitrary names chosen to label the moving average coefficients and `B` is the backshift operator.

  One advantage of this feature is that imposing constraints among parameters are straightforward. For example, if in the previous model for a given dataset the constraint `ma1 = ma2` wanted to be imposed, the model code would be '`(1+ma1*B+ma1*B2)/(1-B)`' instead, and only the `ma1` parameter needs to be estimated.

- Either conditional or exact Maximum Likelihood (ML) estimation of ARIMA models are available (see e.g., [1]). Conditional ML is always used as a mean to obtain initial conditions for exact estimation, but it is convenient when the model involves very long time series or is very complex, as is the case of models with multiple seasonal factors or many parameters.

- An algorithm for automatic identification of ARIMA models (function `modelAUTO`) is included, inspired in [2] and coded in the widespread **forecast** package in R. The procedure follows this reference except in the way differencing orders are identified. In particular, instead of relying on formal unit root tests, **ECOTOOL** selects difference orders orders by minimizing the variance of the resulting time series. This discrepancy is introduced due to many problems detected with formal unit root tests when applied to real time series. In addition, the automatic method is expanded to multi-seasonal models, making **ECOTOOL** the unique piece of software that implements this procedure, to the author knowledge.

- Automatic identification of four types of outliers are coded for ARIMA and TF models, following [3] and [4]. The types are additive, innovative, level shift and transitory change (coded in **ECOTOOL** as AO, IO, LS and TC). They are modeled as particular TFs applied to impulse dummy variables.

- ETS and UC models are estimated from their equivalent 'reduced' or ARIMA form, as in [5]. This allows to incorporate to these methods all the power **ECOTOOL** offers for modeling ARIMA processes. In particular, they may be estimated by exact ML, may include inputs as transfer functions and automatic detection of outliers may be carried out. All these are, once more, unique properties of **ECOTOOL**, as far as the author is concerned.

- The estimation output of any sort of models is rather exhaustive in tabular form. Such tables show parameter values with their standard errors and T tests, information criteria, correlation among parameters and, in the case of TF and ARIMA models, warnings about problems with unit roots in either numerator or denominator polynomials.

There are two functions that produce GUI interfaces that deserve special attention, namely `toolTEST` and `toolFORECAST`.

`toolTEST` is conceived as a friendly and exhaustive environment for descriptive statistics, as well as an identification tool and model validation tool

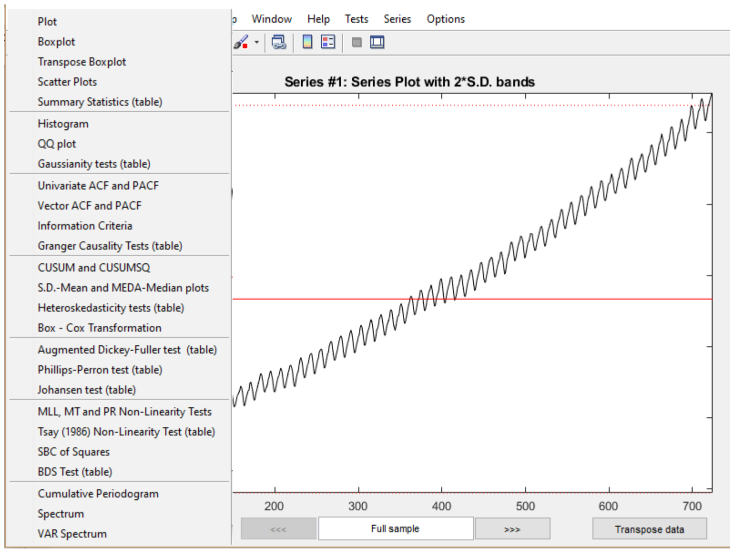

Figure 1: `toolTEST` caption showing CO_2 concentration at Mauna Loa in first worked example with the 'Tests' menu displayed.

of multivariate time series. When it is invoked, three menus unfold in a standard figure window in addition to the usual figure menus: i) a comprehensive 'Tests' menu, shown in Figure 1 and briefly explained below; ii) a 'Series' menu if the input is multivariate and allows to apply the tests to any individual and/or to all the time series at once; and iii) an 'Options' menu to deal with specific options for each item in the 'Tests' menu. Figure 1 offers an overall view of the tool with the 'Tests' menu unfolded.

The 'Tests' menu offers a thorough combination of tabular and graphical information for many tests available. The following is a non-exhaustive list of such tests, classified in different categories:

- Descriptive information: time plots, box plots, scatter plots, descriptive statistics, histograms, formal Gaussianity tests [6, 7].
- Identification tools: univariate and multivariate sample autocorrelation and partial autocorrelation functions, Ljung-Box Q and Monti tests [8, 9], information criteria (Akaike's, Schwarz, Hannan and Quin, see [10, 11, 12]), Granger causality tests based on VAR models [13].
- Constant mean and heteroskedasticity tests: CUSUM and CUSUMSQ tests [14], mean vs standard deviation scatter plots, formal ratio of

variances heteroskedasticity test, Box-Cox transformation estimation [15, 16].

- Integration and cointegration tests: Dickey-Fuller and Perron unit root tests, Johansen cointegration tests [17, 18, 19].
- Non-linearity tests: [20, 9, 21, 22], Schwarz criterion on squares.
- Frequency domain tools: cumulative periodogram, smoothed or raw periodogram, AR-spectrum [23, 24].

The second useful GUI is the so called `toolFORECAST`, that is designed to show graphically the forecasting output of the model functions in **ECOTOOL** and to print out tables with plenty of error metrics (see examples in section 3).

Function `toolFORECAST` allows to plot one or several forecasts and forecasting errors for each time series in turn, selecting the output to be displayed interactively by menus. For long time series several pushbuttons permit to slide side-wise along the time series (as in `toolTEST`). Tabular results may be displayed in three different forms, i) actual values, forecasts, errors and error metrics for each time series and each forecasting step separately; ii) overall error metrics for one time series but many methods altogether to do fast comparisons; and iii) Diebold-Mariano test, sign test and Wilcoxon signed-rank test for testing statistical significant differences among several forecasting methods [25, 26].

Table 2: Naïve methods implemented in function `modelNAIVE`.

Method	Forecast calculation ($\hat{y}_{T+l}$)
Mean	$\sum_{t=1}^{T} y_t/T$
Random Walk (RW)	y_T
Seasonal RW	$y_{T+l-(\lfloor (l-1)/s \rfloor+1)s}$
Mean Seasonal RW	$\sum_{k=l-s+1}^{l} \hat{z}_{T+k}$, where $\hat{z}_{T+k}$ are the *seasonal RW* forecasts
Drift	$y_t + (y_T - y_1) \times l/(T-1)$
Mean Drift	$y_t + (x_T - y_1) \times l/(T-1)$, where x_T is the mean of the data in the last season available

2 ECOTOOL models and methods

ECOTOOL offers the possibility of identifying, estimating, testing and forecasting time series by methods with different degrees of complexity, from pure univariate to full multivariate systems. The next sub-sections show the full list of methods implemented.

2.1 Naïve methods

The simplest methods, commonly used as benchmarks, are a set of naïve methods, i.e., mere rules of thumb that needs just simple computations or no computations at all. They are strictly correct under some severe restricting assumptions, but usually most of them have no meaning at all for many time series.

Let's call $y_t, t = 1, 2, \ldots, T$ a time series; s the seasonal period; l the forecast horizon and $\lfloor x \rfloor$ the integer part of x. Then, the l steps ahead forecasts conditional on all information available up to time T (i.e., $\hat{y}_{T+l}$) is estimated by each of the naïve methods as shown in Table 2. It is assumed that if more sophisticated methods are better in forecasting terms, they should outperform all these naïve options. *Mean* forecasts are just the mean of the in-sample data; *RW* forecasts are just the last observation; *seasonal RW* is the last seasonal cycle available; *mean seasonal RW* is the mean of the last seasonal cycle of data; *drift* is a linear forecast with a slope calculated by joining the last and the first observations; *mean drift* is similar but the slope is based on the *mean seasonal RW* forecast.

All these naïve methods are implemented in the function `modelNAIVE`.

2.2 ARIMA

The ARIMA models implemented in **ECOTOOL** in the function `modelTF` are rather general [1], since it allows for multiple seasonal polynomials. This is what some authors have called multi-seasonal ARIMA models (similar to the more restricted multi seasonal ARI models of [27]). The general formulation is in equation (1), where z_t is a stationary time series; B is the back-shift operator such that $B^l z_t = z_{t-l}$; $\theta_{Q_i}(B^{s_i})$ and $\phi_{P_i}(B^{s_i})$ are moving average and autoregressive polynomials of order Q_i and P_i, respectively, in which the exponent of the backshift operator in each summand is a multiple of the seasonal frequency, s_i $(i = 1, 2, \ldots, k)$; and a_t is a Gaussian serially independent white noise with zero mean and constant variance.

$$z_t = \frac{\theta_{Q_0}(B)}{\phi_{P_0}(B)} \frac{\theta_{Q_1}(B^{s_1})}{\phi_{P_1}(B^{s_1})} \frac{\theta_{Q_2}(B^{s_2})}{\phi_{P_2}(B^{s_2})} \cdots \frac{\theta_{Q_k}(B^{s_k})}{\phi_{P_k}(B^{s_k})} a_t \tag{1}$$

Time series z_t in (1) is assumed stationary, but usually is the result of applying the differencing operators to a non stationary time series y_t, as in (2).

$$z_t = (1-B)^{d_0}(1-B^{s_1})^{d_1} \ldots (1-B^{s_k})^{d_k} y_t \tag{2}$$

ECOTOOL offers an automatic identification procedure inspired in [2] applied to multi seasonal ARIMA models. It is effectively a much more complex process than [27] in the sense that identification is fully automatic and that it allows for moving average terms. As far as the author is concerned, this is the first time such identification procedure is implemented successfully in a time series package for multi seasonal models (see section 3).

2.3 Unobserved Components (UC)

ECOTOOL allows for UC models known as a Basic Structural Model of Harvey in the function `modelUC` [28]. The formulation of this model is shown in equation (3), where a time series y_t is decomposed into a long term trend (T_t), a seasonal component (S_t) and an irregular component (I_t).

$$y_t = T_t + S_t + I_t \tag{3}$$

Particular formulations are possible by selecting different alternatives of each component. For trend models, the possibilities are condensed in equation (4), where T_t^* is referred to as the trend 'slope', α is a parameter between zero and one, and η_t and η_t^* are independent white noise sequences with variances σ_η^2 and $\sigma_{\eta^*}^2$, respectively.

$$\begin{bmatrix} T_{t+1} \\ T_{t+1}^* \end{bmatrix} = \begin{bmatrix} \alpha & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} T_t \\ T_t^* \end{bmatrix} + \begin{bmatrix} \eta_t \\ \eta_t^* \end{bmatrix} \tag{4}$$

This model is called Generalised Random Walk Trend and subsumes the following specific cases implemented in **ECOTOOL**: i) Random Walk (RW), by eliminating the second equation and $\alpha = 1$; ii) Integrated Random Walk (IRW) with $\alpha = 1$ and $\sigma_\eta^2 = 0$; iii) Smoothed Random Walk (SRW) with $\sigma_\eta^2 = 0$; and iv) Local Linear Trend (LLT) with $\alpha = 1$, see e.g., [28], [29] and [30].

Seasonal components are included as the so called dummy seasonality [28], that depends on a single parameter, namely the variance of white noise ω_t, see equation (5).

$$\begin{bmatrix} S_{1,t+1} \\ S_{2,t+1} \\ S_{3,t+1} \\ \vdots \\ S_{s-1,t+1} \end{bmatrix} = \mathbf{\Psi} \begin{bmatrix} S_{1,t} \\ S_{2,t} \\ S_{3,t} \\ \vdots \\ S_{s-1,t} \end{bmatrix} + \begin{bmatrix} \omega_t \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$
$$\text{with } \mathbf{\Psi} = \begin{bmatrix} -1 & -1 & -1 & \dots & -1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{5}$$

The full UC model is built by assembling all the State Space models for the components. As a matter of fact, equation (3) is the observation equation of a State Space system and the block concatenation of all models for the trend (4) and seasonal (5) form the transition equation. The Kalman Filter and other recursive algorithm provides the optimal solution to state estimation (see details in [28, 29, 30]).

Table 3: ETS type of components in `modelETS`.

Code	Model
NN	$y_t = l_{t-1} + e_t$
	$l_t = l_{t-1} + \alpha e_t$
AN	$y_t = l_{t-1} + b_{t-1} + e_t$
	$l_t = l_{t-1} + b_{t-1} + \alpha e_t$
	$b_t = b_{t-1} + \beta e_t$
DN	$y_t = l_{t-1} + b_{t-1} + e_t$
	$l_t = l_{t-1} + b_{t-1} + \alpha e_t$
	$b_t = \phi b_{t-1} + \beta e_t$
NA	$y_t = S_{t-s} + e_t$
	$S_t = S_{t-s} + \gamma e_t$
AA	$y_t = l_{t-1} + b_{t-1} + S_{t-s} + e_t$
	$l_t = l_{t-1} + b_{t-1} + \alpha e_t$
	$b_t = b_{t-1} + \beta e_t$
	$S_t = S_{t-s} + \gamma e_t$
DA	$y_t = l_{t-1} + b_{t-1} + S_{t-s} + e_t$
	$l_t = l_{t-1} + b_{t-1} + \alpha e_t$
	$b_t = \phi b_{t-1} + \beta e_t$
	$S_t = S_{t-s} + \gamma e_t$

2.4 ExponenTial Smoothing (ETS)

The ETS models implemented in function `modelETS` are taken from [31], see Table 3 for the full list of possibilities. Multiplicative forms are possible by using the log transformation.

In Table 3, l_t, b_t, S_t and e_t stand for the level, slope, seasonal and irregular components respectively; and α, β, γ and ϕ are unknown parameters that should be estimated. The code is composed of two letters and defines the model components, the first letter specifies the trend and the second letter is reserved for the seasonal component. 'N' indicates that the component is not present; 'A' implies that the component is additive; 'D' implies a damped component, applicable to trends only (with $0 < \phi < 1$). When specifying an ETS model with a seasonal component, the two letters ought to be followed by a number indicating the fundamental period in samples per cycle.

2.5 Transfer function (TF)

Multiple Input Single Output TF models are implemented in **ECOTOOL** by means of the `modelTF` function. The model may be written as in equation (6).

$$y_t = \sum_{i=1}^{h} \frac{\omega_{n_i}(B)}{\delta_{m_i}(B)} u_{i,t} + N(B)a_t \tag{6}$$

In this equation $\omega_{n_i}(B) = (\omega_0 + \omega_1 B + \omega_2 B^2 + \ldots + \omega_{n_i} B^{n_i})$ and $\delta_{m_i}(B) = (1 + \delta_1 B + \delta_2 B^2 + \ldots + \delta_{m_i} B^{m_i})$ are polynomials in the backshift operator of order n_i and m_i, respectively; and $N(B)a_t$ is a general representation for any of the univariate alternatives in **ECOTOOL**, namely ARIMA, UC, and ETS. Mind that linear regression is a particular case of model (6) if all numerator and denominator polynomials are of order zero.

When the noise model in equation (6) is specified as an AR, ARMA or ARIMA model, it is possible to transform the TF model into an ARX, ARMAX or ARIMAX, by setting appropriate constraints on the polynomials. This operation is straightforward in **ECOTOOL** (see section 3).

While linear regression or TF models with ARIMA noise have been used abundantly for a long time (since the first edition of [1] in 1976), ETS or UC combined with TF models is a unique feature of **ECOTOOL** toolbox, as far as the author is concerned. Taking advantage of this feature, function `modelTF` also allows for the automatic detection of four types of outliers in ARIMA, UC and ETS models (see section 3). Once more, automatic identification of outliers in UC or ETS models is a unique feature of **ECOTOOL**.

2.6 VARX

VARX models in equation (7) are the multivariate option implemented in **ECOTOOL** by means of the function `modelVARX`, where boldface letters indicate either matrices or vectors.

$$\begin{aligned}(\mathbf{I} + \mathbf{\Phi}_1 B + \ldots + \mathbf{\Phi}_p B^p)\mathbf{z}_t = \\ (\mathbf{\Omega}_0 + \mathbf{\Omega}_1 B + \ldots + \mathbf{\Omega}_q B^q)\mathbf{u}_t + \mathbf{a}_t\end{aligned} \tag{7}$$

In this equation, $\mathbf{z}_t$ stands for a vector of p stationary outputs (generally obtained by appropriate differencing of corresponding vector of non-stationary time series $\mathbf{y}_t$); $\mathbf{u}_t$ represent a set of m inputs; $\mathbf{\Phi}_i$ are a set of squared matrices of dimension $p \times p$; $\mathbf{\Omega}_j$ are a set of matrices of dimension

$p \times m$; and $\mathbf{a}_t$ are a vector of p white noises with zero mean and non-diagonal covariance matrix $\mathbf{\Gamma}$.

Optimal estimation of unconstrained VARX models is easy, since equation by equation estimation by least squares renders consistent and efficient estimates. **ECOTOOL** allows for imposing constraints on the coefficients for which iterative generalised least squares are used to reach efficient estimation with a number of iterations controlled by the user [23].

3 ECOTOOL examples

The present section considers three examples and one case study chosen to illustrate **ECOTOOL** working on real data. The proposed examples are provided as mere illustrations of the methods included in **ECOTOOL** and by no means are meant to be polished analysis of each case or the best way to analyze any of them. In addition, not all the function capabilities are shown and the output of many routines are truncated to save space. For fuller descriptions and outputs see the demos included in the toolbox, the help for each particular function and the toolbox documentation.

3.1 Example 1: Univariate models

The well-known monthly measured CO_2 concentration at Mauna Loa is represented in Figure 2 (`ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt`) and is used to illustrate how to implement univariate models in **ECOTOOL**. The data consists of sixty years of monthly data collected from March 1958 to May 2018. The data show a growing trend during the full period associated with long run emissions and a seasonal component rather stable related to the global net uptake and release of CO_2 in summer and winter.

The next listing shows how to load the data, select the raw CO_2 data (stored in the fifth column of the matrix data downloaded from the official web page), avoid the first and last two years of monthly observations and select two parameters that will be used later on, namely the number of forecasts (`nofs`) and the seasonal period (`s`). The last line is useful for those users who want to try the variance stabilizing Box-Cox transformation of the data. It can also be accessed by the 'Box-Cox Transformation' option in the 'Tests' menu of the `toolTEST` GUI. By this command the λ parameter of the transformation is estimated according to [16] and returns the transformed variable (`ty`) and λ (`lambda`).

```
load maunaLoa.data                      % Load data
y = maunaLoa(25 : end - 24, 5);         % Select-in sample CO2
nofs = 24;                              % Number of forecasts
s = 12;                                 % Seasonal period
[ty, lambda] = vboxcox(y);              % Box-Cox transform
```

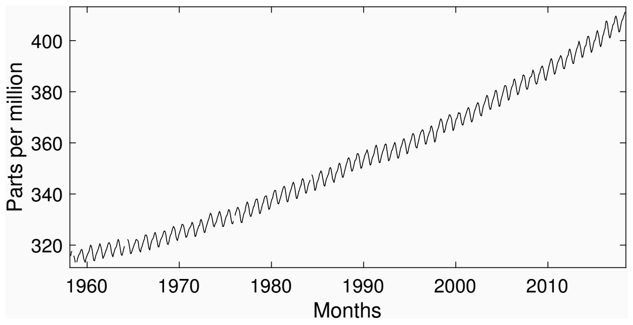

Figure 2: The Mauna Loa CO_2 concentration data from March 1958 to May 2018

Naïve models

The simplest choice of all univariate methods are the naïve ones shown in Table 2. There are several ways to run these methods, shown in next listing.

```
m1 = modelNAIVE(ty, nofs, s);               % Run methods
m1b = modelNAIVE(ty, 'nofs', nofs, 's', s);
m1c = modelNAIVE(ty, 's', s, 'nofs', nofs);
```

The first call just sets all the inputs to the function in strict order according to the documentation, default values are assumed when an empty matrix is provided. The second call allows to select the inputs by tuples. One advantage of this syntax is that the inputs may be sorted in any order (as the third call shows) and only the necessary inputs may be invoked, leaving default values for the rest.

The inputs to function `modelNAIVE` are just the data, the forecast horizon and the seasonal period. The output is an 'object' (a MATLAB structure) with several fields containing all the relevant information about the model and the results, namely, the type of model (`m1.type`), all the inputs to the function used (`m1.nofs`, `m1.nofs`, `m1.s`) and the forecasts (`m1.py`), i.e., a six column matrix `nofs`$\times$6 containing the forecasting results for the six methods in Table 2.

ARIMA model

Fitting an ARIMA model implies an analysis of stationarity of the time series and identification of a model in the first place. This can be done in two ways, i) by the 'Tests' menu of the `toolTEST` GUI or ii) by calling directly the functions actually executed by the options available in that menu. Running the tests through the `toolTEST` menu is encouraged when analyzing individual time series, though direct use of functions is convenient in programming contexts, when many tests ought to be run automatically. Functions are used often in the listings below to make the exposition clearer.

Assuming that one regular and one seasonal difference are necessary to induce stationarity, the identification of an ARIMA model may be performed with the following code:

```
z = vdif(ty, [1 1], [1 s]);     % Difference series
vident(z);                      % Identification
```

The differenced series is calculated in the first line and stored in a new variable called `z`. Function `vdif` has three inputs, namely the time series to differentiate, the differencing orders applied multiplicatively, and the lag of each difference applied element-wise to the previous input, i.e., $z_t = (1 - B)(1 - B^s)ty_t$. Finally, the second line of code estimates the simple and partial autocorrelation functions of `z` that is assumed to be stationary in the first two moments. The same result would have been achieved by running `toolTEST(z)` and selecting the option 'Univariate ACF and PACF' in the 'Tests' menu. This function produces two outputs, a self explanatory plot (see Figure 3) and a table with complementary statistical information, shown below.

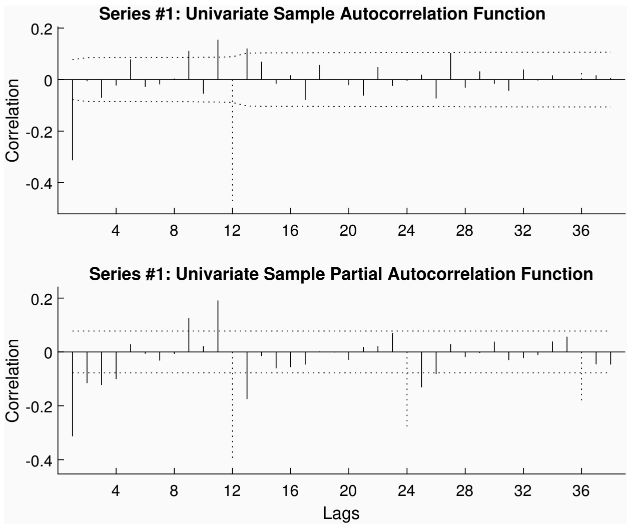

Figure 3: Simple and Partial Autocorrelation functions for the Mauna Loa differenced data. Dotted vertical lines signal the seasonal lags.

```
---------------------------------------------------------------
 LB: Ljung-Box Test
 MT: Monti (1994) Test
---------------------------------------------------------------
 H0: Process is white noise
---------------------------------------------------------------
   Lag     SACF           LB P-value   SPACF          MT P-value
---------------------------------------------------------------
     1  -0.3133  -  65.2749   0.00  -0.3133  -  65.2749  0.00
     2  -0.0062  .  65.3009   0.00  -0.1158  -  74.2001  0.00
     3  -0.0710  .  68.6634   0.00  -0.1230  -  84.2937  0.00
     4  -0.0228  .  69.0117   0.00  -0.1006  -  91.0605  0.00
...
```

Figure 3 suggests that a sensible model would be an ARIMA$(0, 1, 1) \times (0, 1, 1)_{12}$. Estimation code is shown in the next listing, in which the model is specified as a string variable almost exactly as it would be written analytically, and it is estimated by `modelTF` function. Once more, two different though equivalent ways to call this function are shown.

```
model = '(1+ma1*B)(1+ma12*B12)/(1-B)(1-B12)';
m2 = modelTF(ty, [], model, nofs);
m2b = modelTF(ty, 'model', model, 'nofs', nofs);
```

By default, the model is estimated by Exact Maximum Likelihood with no automatic detection of outliers, etc. All these options may be customized by the user with a longer function call (see section 3.2). The output is a structure with all the relevant information, most important are the residuals (`m2.e`) and the forecasts (`m2.py`). Function `modelTF` also produces and stores a table output with information about the model estimation and diagnostic checking, see next listing.

```
--------------------------------------------
          PARAM    STERR    T-TEST  P-value
--------------------------------------------
 ma1   -0.38349  0.03965   9.67208  0.00000
 ma12  -0.88282  0.01964  44.94003  0.00000
--------------------------------------------
               AIC: -15.4919
               SBC: -15.4783
               HQC: -15.4866
Objective Function: 0.0001229
      Corrected R2: 0.99985
 Residual Variance: 1.8592e-07
--------------------------------------------
```

A final step is model validation, i.e., checking whether all the statistical assumptions on which the model is based are met. The ideal way to deal with this requirement is to run `toolTEST` on the residuals and pass through most of the test options available on the 'Tests' menu. In this way, several formal tests and graphical heuristics are produced, like plots, scatter and seasonal plots (if applicable); summary statistics, including a test for zero mean; gaussianity tests; remaining autocorrelation; heteroskedasticity tests; unit root tests; non-linear tests; frequency domain tests.

Figure 4 shows the output of a few of such tests applied to the model residuals: a histogram with the theoretical Gaussian distribution on top, the cumulative periodogram to test for whiteness, the CUSUM and CUSUMSQ to check constancy of mean and variance and the autocorrelation fuctions.

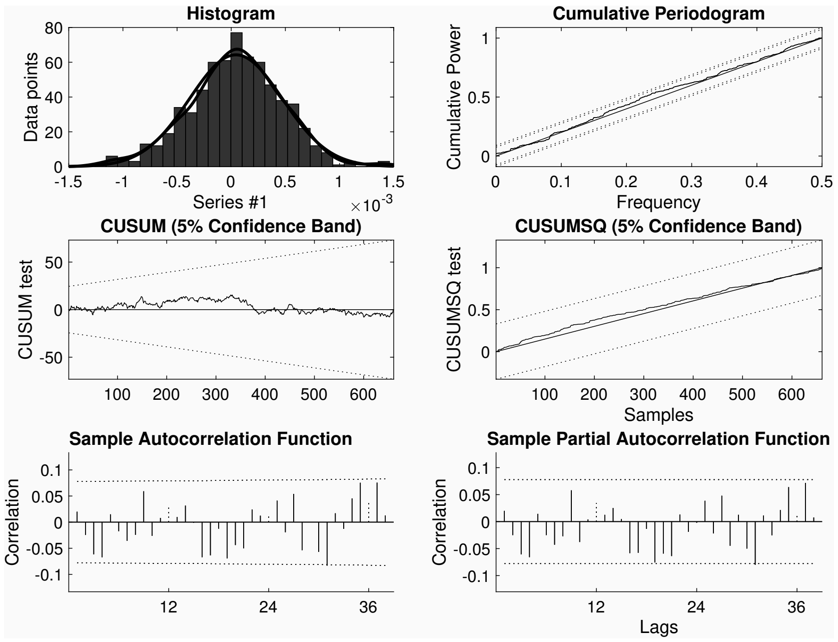

Figure 4: Tests for residuals of ARIMA model run on the CO_2 data.

A simpler way to deal with the identification of ARIMA models is by running an automatic identification algorithm. A variant of [2] procedure (avoiding unit root testing) may be run in **ECOTOOL**:

```
m3 = modelAUTO(y, [], [1 s]);
```

After doing a search on several ARIMA models in incremental order, the same model as before is finally identified. This function may be custimized in different ways, check function help and documentation.

UC model

From an UC perspective, a model that is appropriate for trending data with a clear seasonal component is the Basic Structural Model of Harvey [28]. This model is run with the single command:

```
m4 = modelUC(ty, [], 'LLT12', nofs);
```

The model is specified in the third input as a string in which `LLT` stands for 'Local Linear Trend', and it is followed by the seasonal period. Other trend

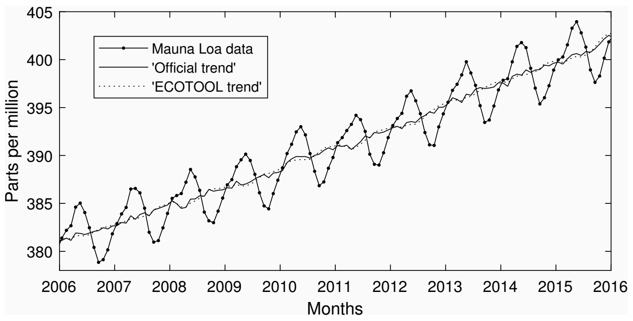

Figure 5: Detail of Mauna Loa data with the 'official' and `modelUC` trends.

models implemented are RW, IRW, SRW. A final possibility is applying a Hodrick-Prescott filter to the series with an arbitrary smoothing constant λ by selecting the model as `HP#`, where `#` is the chosen λ. Additional inputs allows for switching between conditional and exact Maximum Likelihood, and automatic outlier detection.

The output is a structure with many fields, from which the most relevant in this case are the innovations of the model (`m4.e`, white noise for a correctly specified model); forecasts (`m4.py`); and the estimated components (`m4.comp`: a three column matrix containing the trend seasonal and irregular). Figure 5 shows a detail of the trend reported in the official web and the trend estimated by **ECOTOOL**, once it is converted back to its original scale by function `vboxcoxinv`. Both trends, though not exactly equal, are quite consistent with each other.

ETS model

The sintax for ETS models is very similar to the UC models. The function to use is `modelETS`, the inputs and outputs are the same but the model string includes six possibilities, by combining level and seasonal options followed by the seasonal period ('N' for none, 'A' for additive and 'D' for damped). The following code runs a Holt-Winters model on the data and store the model output in `m5` structure. Once more, exact likelihood estimation and automatic outlier detection is available, by calling `modelETS` with additional inputs.

```
m5 = modelETS(ty, [], 'AA12', nofs);
```

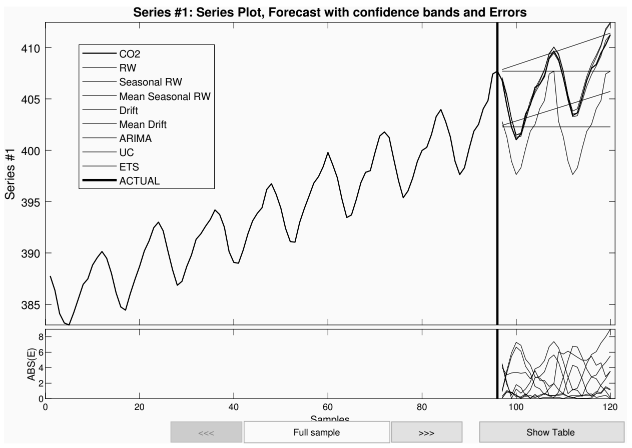

Figure 6: Forecasts produced with all univariate methods.

Univariate forecasting comparisons

So far all sort of univariate models available in **ECOTOOL** have been applied to the CO_2 data. Forecast comparisons among all of them in graphical and tabular form may be done with the `toolFORECAST` function. A possible call for making comparisons in the original scale is shown in the next listing, where the actual values are stored on variable `ACTUAL`, and `FORECASTS` is a matrix with all the univariate forecasts except the mean (that is exceptionally wrong). All of them are converted back to the original scale by the `vboxcoxinv` function.

```
ACTUAL = maunaLoa(end - 119 : end, 5);
aux = [m1.py(:, 2 : end) m2.py m4.py m5.py];
FORECASTS = vboxcoxinv(aux, lambda);
toolFORECAST(ACTUAL, FORECASTS)
```

The output is shown in Figure 6, where it may be seen easily that some of the forecasts are pretty bad. The 'Show Table' button at the bottom is useful for showing the tabular output and make comparisons on the basis of error metrics. The output table is different depending on whether one single or all the forecasts are shown in the plot.

The output table shows a comparison among methods based on a rich variety of error metrics, as shown below, where it is clear that all the naive methods (up to method #5) are much worse than the more sophisticated, being ETS the best of all across all error metrics.

```
---------------------------------------------------------------
 Forecasting Results
---------------------------------------------------------------
 ME:          Mean Error
 MAE:         Mean Absolute Error
 MASE:        Mean Absolute Scaled Error
 MAPE(\%):     |F - y| / y (Absolute Percentage Error)
 MAPEf(\%):    |F - y| / F (APE with respect to forecast...)
 sMAPE(\%):    |F - y| / (y + F) / 2
---------------------------------------------------------------
       N      ME    MAE   MASE  MAPE(\%)  MAPEf(\%)  sMAPE(\%)
---------------------------------------------------------------
 F#1  24   1.465  2.564  2.154    0.633     0.629        0.631
 F#2  24  -3.945  3.945  3.314    0.971     0.982        0.977
 F#3  24  -3.959  4.124  3.464    1.011     1.025        1.018
 F#4  24   3.389  3.398  2.855    0.840     0.830        0.835
 F#5  24  -2.164  2.661  2.236    0.653     0.658        0.655
 F#6  24   0.501  0.533  0.448    0.131     0.131        0.131
 F#7  24   0.356  0.462  0.388    0.113     0.113        0.113
 F#8  24  -0.053  0.298  0.250    0.073     0.073        0.073
---------------------------------------------------------------
```

3.2 Example 2: Outlier detection and intervention analysis

The work by [32] analyzes the effect of the seat belt law on road accidents in Great Britain, based on monthly casualties from January 1969 to December 1984. The car drivers killed and seriously injured data has been used in several references as an illustration of UC models [28, 30]. The interest of this data is that there are some breaks in the mid-seventies, probably due to oil crisis and in February 1983 after the introduction of the seat belt law, as can be seen by eye in Figure 7. One full year of missing observations has been artificially added to make this example more appealing.

In order to show the **ECOTOOL** capabilities when analyzing this sort of data, UC and ETS models have been estimated combined with automatic detection of outliers and estimation of calendar effects altogether. As far as the author is concerned, this is the first time a procedure of this kind

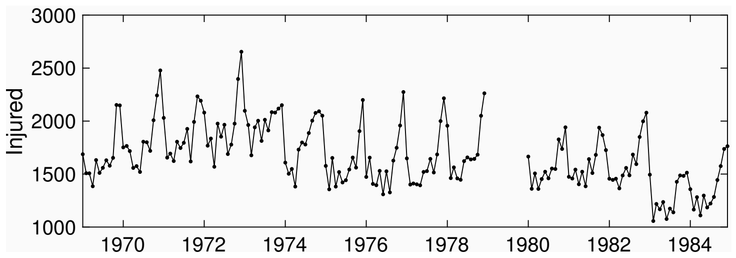

Figure 7: Car drivers killed and seriously injured in the UK between 1969 to 1984.

is proposed on UC and ETS models. The calendar dummy variables are produced by a number of additional functions and condensed in a single matrix `u`, shown in the next listing.

```
load victims
y = vboxcox(victims);                  % Box-Cox transformation
n = length(y);                         % Number of data points
Ieaster = easter([1969 1], n);         % Easter dummy variable
Itrading = trading([1969 1], n);       % Trading day dummy
Ileap = leapy([1969 1], n);            % Leap-year dummy
Idays = days([1969 1], n);             % Days per month dummy
u = [Ieaster Itrading Ileap Idays];    % Matrix of all dummies
```

The calendar functions are:

- Function `easter`: outputs a dummy variable taking into account the fact that Easter is a moving holiday some years celebrated in March, some in April and some in between. There are a number of different configurations possible, based on a reference day, the days before the reference affecting the output variable and the weights for each of such days. Default values are Easter Sunday as the reference day and 5 days before with equal weights.

- Function `trading`: allows the introduction of corrections for business or trading days vs weekend and festive days to take into account the different proportion of working and non-working days in each month. It is possible to set it up in two ways: i) just weekdays vs weekends (on the basis of 5 weekdays for each 2 weekend days or 6/1) or ii) treating all days differently and measuring a distinct effect for each day.

- Function `leapy`: it takes into account the calendar distortion due to leap years by specifying a dummy variables with ones on every February of every leap year.
- Function `days`: it includes a correction in days by the fact that each month has different number of days.

The following listing shows how UC and ETS models may be estimated.

```
model = {'LLT12', 'EASTER', 'TRADING', 'LEAP', 'DAYS'};
m1 = modelUC(y, u, model, [], 'EML', 3);        % UC model
model{1} = 'AA12';
m2 = modelETS(y, u, model, [], 'EML', 3);       % ETS model
```

The overall model is in a TF form like equation (6). The model is inputted as a cell variable `model` in which the first element is the noise model (either UC or ETS) and the rest are the transfer function terms for the input variables. In particular, all the transfer functions in these models are just parameters with zero delays, equivalent to linear regression terms. To state it more clearly, the second cell element in `model` (`EASTER`), for example, is just the name of the parameter affecting the first input of the model. It could have been specified as a dynamic transfer function model like $(\omega_0 + \omega_1 B + \omega_2 B^2)/(1 + \delta_1 B + \delta_2 B^2)$, for second order numerator and denominator polynomials. This model would be included in the second place of cell variable `model` as `'(w0+w1*B+w2*B2)/(1+d1*B+d2*B2)'`.

The fifth input to `modelUC` and `modelETS` selects Exact Maximum Likelihood as the estimation method. The last input in each function call tells `ECOTOOL` to check whether any residual out of a number of standard errors bound is an outlier (3 in the previous listing). In addition, for those identified as outliers it checks which type of outlier is among additive, innovative, level shift or tansitory change (AO, IO, LS, TC, respectively; see [3, 4]). The outlier input can be customize in several ways, see the help and documentation.

Notice that the syntax of both functions (`modelUC` and `modelETS`) is exactly the same, with the only difference that the first element in the model is either an UC or an ETS model. The (truncated) estimation table is shown in the next listing for the UC model.

```
-----------------------------------------------------------
               PARAM      SCORE    STERR   T-TEST  P-value
-----------------------------------------------------------
 Level       0.00409   -2.38865  1.13593  2.10281  0.01843
 Slope       0.00004   -4.38056  0.50443  8.68414  0.00000
 Seasonal    0.00000  -10.42722  0.00000      Inf  0.00000
 Irregular   0.65201   -0.18575  0.05065  3.66701  0.00016
 EASTER      0.44307          -  0.31367  1.41252  0.07976
 TRADING    -0.04160          -  0.01668  2.49377  0.00677
 LEAP       -0.79427          -  2.28481  0.34763  0.36426
 DAYS        1.13635          -  2.33420  0.48683  0.31349
 LS170      -3.01089          -  0.55131  5.46133  0.00000
 LS59       -2.56870          -  0.57405  4.47468  0.00001
 LS71       -2.28330          -  0.53700  4.25193  0.00002
-----------------------------------------------------------
```

This table is divided in several sections:

- Variance estimation of all the components involved with their respective score parameters. For convenience in the estimation procedure, the likelihood is directly defined on the scores and transformed back to their variance counterparts (see e.g., [28, 29, 30]). Any variance close to zero, implies a deterministic or very stable component.
- Calendar effects. Easter is positive, close to the significance boundary, meaning that there are more injured drivers during these holidays. `TRADING` is clearly significant and negative implying that there are more accidents during weekends.
- Outliers automatically identified. Three highly significant level shifts detected on November 1973, November 1974 and February 1983.

Notice that the UC components are returned in `m1.comp` variable. Because of all the dummy and outliers effects, this matrix contains trend, seasonal, irregular and all the rest of effects in the same order that appears in the output table. Level shifts are usually associated to long run movements, they are usually added to the trend. Figure 8 shows all the estimated components.

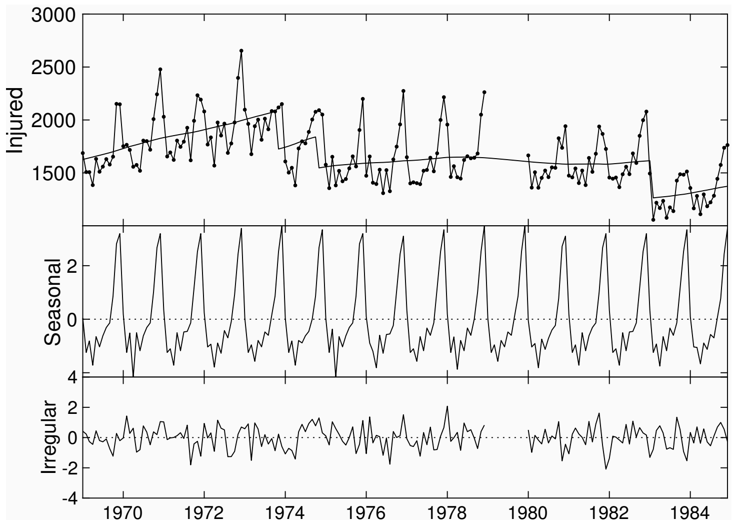

Figure 8: Car drivers killed Top: series and trend with level shifts. Middle: seasonal component. Bottom: irregular.

3.3 Example 3: Transfer function and multivariate models

This example shows how to deal with dynamic relations between advertising and sales with **ECOTOOL**. The annual data for the Lydia Pinkham's vegetable compound between 1907 and 1960 is shown in Figure 9 and will be used here following [23] indications, which closely follows the orthodox Box-Jenkins methods for dynamic modeling. One first step is some sort of exploratory analysis of the data, that may be done following the next listing.

```
load Lydia_Pinkham
u= Lydia_Pinkham(:, 1) / 1000;
y= Lydia_Pinkham(:, 2) / 1000;
z = [u y];
toolTEST(z, 'names', 'Advert Sales');
```

The call to `toolTEST` allows for many prior analyses through the 'Tests' menu, like graphical representations of several sorts, reporting a statistics summary, performing unit root tests, etc. Both time series are non-stationary according to unit root tests, and the identification in first differences suggest an AR(1) model for sales and AR(2) for advertising. Such models are estimated and tested in the following listing.

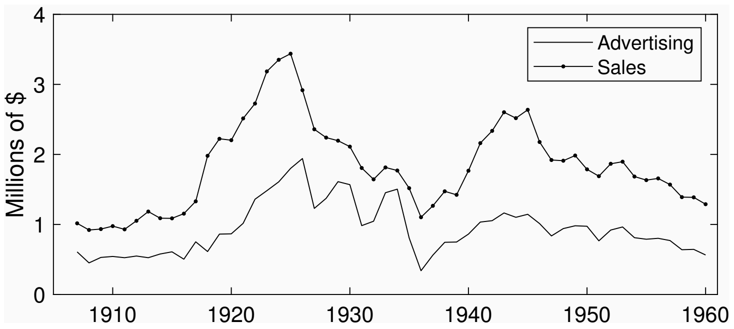

Figure 9: Lydia Pynkham annual data.

```
% Identification of first differences
vident(vdif(z))
% Univariate model and forecasts of input (prewhitening)
m1 = modelTF(u, [], '(1)/(1-B)(1+ar1*B+ar2*B2)', 6);
alpha = m1.e;
toolTEST(alpha)
% Univariate model and forecasts of output
m2 = modelTF(y, [], '(1)/(1-B)(1+ar1*B)', 6);
toolTEST(m2.e)
```

Both calls to `toolTEST` will allow for an exhaustive statistical testing on the residuals checking all the options available in the 'Tests' menu. The next step consists of finding the cross correlation between input and output. However, to avoid spurious correlations, both ought to be filtered by the input model [1]. The prewithened input is actually the residuals from the AR(2) model above (`alpha`), and the prewhitened output may be found easily with the `filter` standard MATLAB function.

```
% Filtering output with input model
beta = filter([1 m1.par'], 1, vdif(y));
% CCF between filtered input and output
vident2([alpha beta(4 : end)], 8)
```

Mind that the model input parameters (`m1.par`) are recovered from the previous call to `modelTF` applied to the input. The last line computes the multivariate sample autocorrelation and partial lag autocorrelation functions (the partial autoregressions may be also computed by changing the default

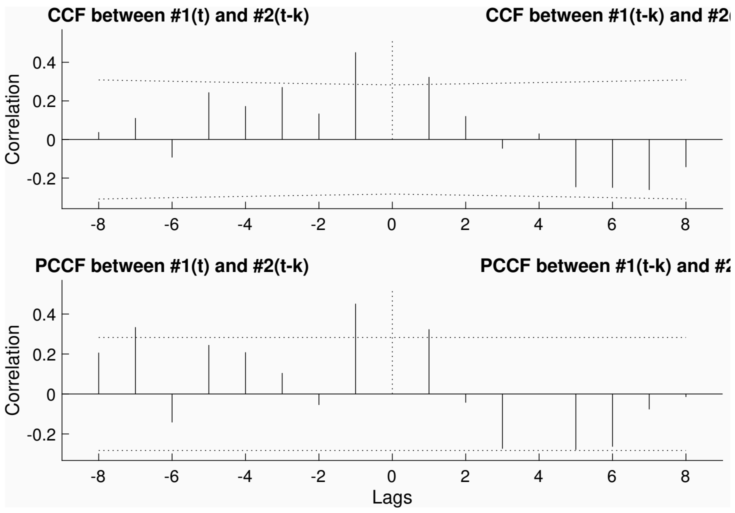

Figure 10: Cross Correlation Function between advertising (#1) and sales (#2) in the Lydia Pynkham data.

options). It also may be accessed by a call to the `toolTEST` function by selecting 'Vector ACF and PACF' option in the 'Tests' menu. The diagonal elements of such multivariate functions provide the autocorrelations of each individual time series, while the off-diagonal elements provide the two directional cross-correlation coefficients. The output to this functions are twofold, one table (shown later) and one plot.

The graphs display the auto- and cross- correlations among all variables involved, organized on windows that may be accessed by the 'Series' menu, see Figure 10 for the output corresponding to the cross correlation function. This figure shows clearly that there is a simultaneous relation between both variables, since there is a decaying cross correlation function at both sides of zero lag. However, concentrating exclusively on the right hand side with positive lags for the moment and for illustrative purposes, two possible models are identified by [23], either a first lag transfer function, i.e., $y_t = (\omega_0 + \omega_1 B)u_t + e_t$; or a first order transfer function, i.e., $y_t = \omega_0/(1 + \delta_1 B)u_t + e_t$.

The code to estimate and forecast the first order transfer function model is shown in the next listing.

```
% Transfer function modeling (deterministic input)
model = {'1/(1+ar1*B)(1-B)', 'w0/(1+delta1*B)'};
m3 = modelTF(y, u, model, m1.py);
% Transfer function modeling (stochastic input)
m4 = modelTF(y, u, model, [m1.polys 6]);
```

The model is composed of the noise model identified as an AR(1) and the second element that is the first order transfer function, both input and output are differenced once. Two calls to `modelTF` are included, the first returns forecasts treating the input forecasts as deterministic (for which the input forecasts are supplied as `m1.py` estimated previously), while the second call returns the output forecasts assuming the input forecasts are stochastic. In this latter case the input model estimated previously (`m1.polys`) and the forecasting horizon are supplied, instead. Results are shown in the next listing and the difference between the two are only the forecast standard errors.

```
---------------------------------------------
            PARAM    STERR   T-TEST  P-value
---------------------------------------------
 ar1     -0.22989  0.15017  1.53085  0.13199
 w0       0.55117  0.12697  4.34101  0.00007
 delta1  -0.33637  0.17524  1.91951  0.06052
---------------------------------------------
               AIC: -3.2876
               SBC: -3.174
               HQC: -3.2442
---------------------------------------------
```

A final model option is a multivariate one. The accompanying table to Figure 10 is shown in the next listing and provides clear evidence for simultaneous relations. As a matter of fact, a second order vector autorregression model seems clearly correct, since both Akaike's and Swartz's criterion reach the minimum at lag 2 and there is no single significant coefficient for higher lags.

```
----------------------------------------------------------------
 Multivariate Simple and Partial Lag Autocorrelation Functions
----------------------------------------------------------------
k= 0
++ | .. |    1.00000  0.51897
++ | .. |    0.51897  1.00000
k=  1  AIC= -6.4901  SBC= -6.2606  Chi= 24.3787  P-Chi= 0.0001
.+ | .+ |    -0.00744   0.45184  |    -0.00744   0.45184
++ | ++ |     0.32393   0.51357  |     0.32393   0.51357
k=  2  AIC= -6.4583  SBC= -6.0758  Chi=  5.4468  P-Chi= 0.2444
.. | .. |     0.13312   0.13403  |    -0.09353  -0.05501
.+ | .. |     0.12097   0.43847  |    -0.04347   0.22500
k=  3  AIC= -6.4408  SBC= -5.9054  Chi=  5.6297  P-Chi= 0.2286
.. | .. |    -0.08708   0.27119  |    -0.15054   0.10498
.. | .. |    -0.04738   0.21431  |    -0.27418  -0.08146
k=  4  AIC= -6.4278  SBC= -5.7395  Chi=  5.3679  P-Chi= 0.2516
.. | .. |     0.25611   0.17286  |     0.26815   0.20877
.. | .. |     0.03081   0.08083  |     0.00027  -0.10706
----------------------------------------------------------------
```

Granger causality tests and the code to estimate a VAR(2) model (by `modelVARX` function) are shown in the next listing. Two versions of the VAR model are shown, the first one with full parameter matrices and a second one imposing zeros on the elements that where not significant in the first place. In order to set up the zero constraints, a full polynomial is supplied as a model with `NaN` values for the coefficients to be estimated and zeros for the constrained.

```
% Granger causality
vgranger(vdif(z), 1 : 3)
% VAR model
m5 = modelVARX(z, [], [1 2], 6, 1, 1);
% Estimating constrained model
model= [1 0 NaN NaN NaN 0;
        0 1   0 NaN NaN 0];
m6 = modelVARX(z, [], model, 6, 1, 1);
```

The output of Granger causality tests based on VAR(3) models is shown in the following listing and indicates clearly the simultaneous relation between advertising and sales.

```
---------------------------------------------------------------
Based on a VAR(3) model
---------------------------------------------------------------
        H0                      F-Stat  P-value  Chi-stat P-value
---------------------------------------------------------------
 Advert does not cause Sales  3.0967   0.0304    9.2901  0.0257
 Sales does not cause Advert  6.4329   0.0005   19.2988  0.0002
---------------------------------------------------------------
```

The final constrained VAR model is shown in next listing.

```
-------------------------------------------------------------
 Generalised Least Squares Estimation (1 iterations)
-------------------------------------------------------------
              AIC: -6.1284
              SBC: -5.3106
              HQC: -5.8139
              Corrected R2: 0.360696  0.225808
              Residual Variance: 0.029594  0.040870
-------------------------------------------------------------
95% |  COEFFICIENTS    | T-STAT COEFFs      |   S.E. COEFF.
-------------------------------------------------------------
 A1:
 +- |  0.2370  -0.5347  |  2.1419  -4.7932  |  0.1107  0.1116
 @- |  0.0000  -0.4873  |       -  -3.6987  |       -  0.1318
 A2:
 +@ |  0.5122   0.0000  |  5.1957        -  |  0.0986       -
 .@ |  0.2685   0.0000  |  1.9199        -  |  0.1399       -
-------------------------------------------------------------
```

Further adequacy tests of the VAR model may be checked by the following code (see also Figure 11), that represents the inverse of the roots of the VAR polynomial in the unit circle (model is stationary if all the inverse of the roots are inside the unit circle), and its impulse response function with 10% confidence bands based on Montecarlo simulations.

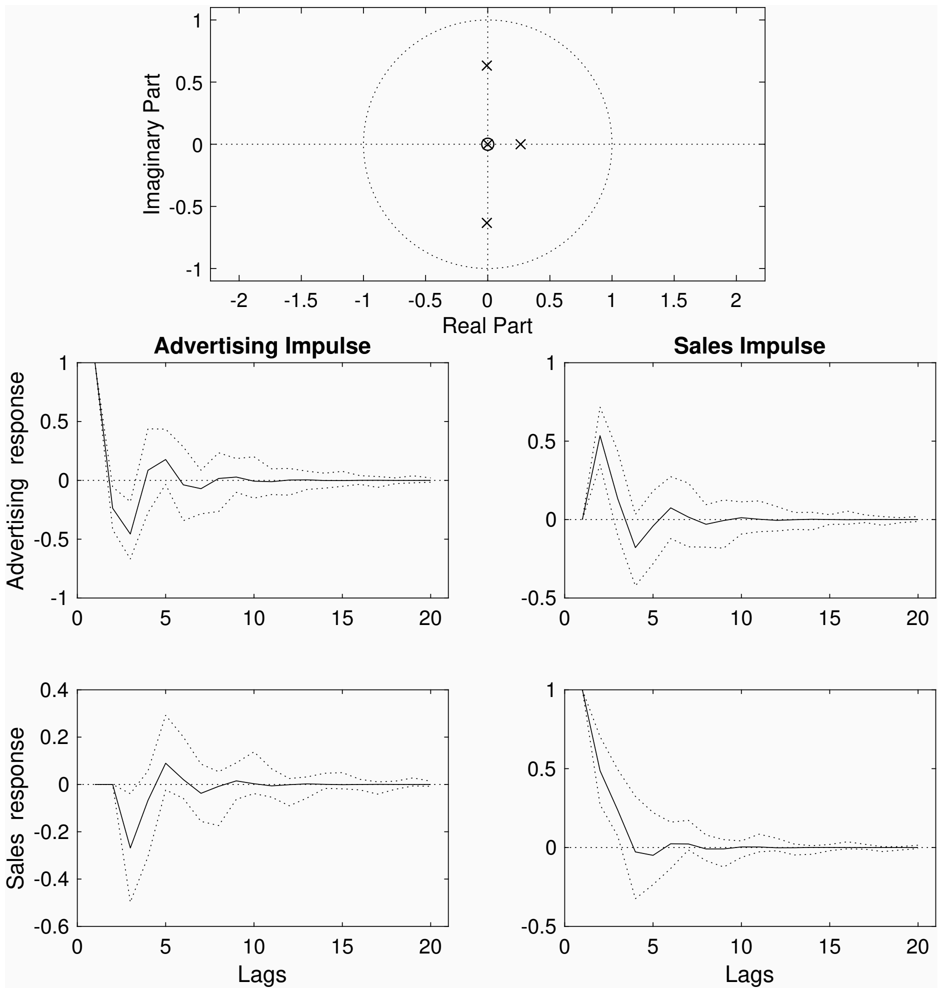

Figure 11: Module of roots (top panel) and impulse response function for VAR(2) model with 10% confidence interval.

Table 4: Sales forecasts of univariate, transfer function and multivariate models.

	AR(1)		Transfer Function			VAR(2)	
h	Point	S.E.	Point	S.E. Determin.	S.E. Stochastic	Point	S.E.
1	1.247	0.216	1.257	0.184	0.277	1.240	0.202
2	1.229	0.377	1.265	0.292	0.595	1.237	0.286
3	1.221	0.514	1.270	0.375	0.931	1.246	0.415
4	1.218	0.631	1.265	0.444	1.283	1.244	0.530
5	1.216	0.733	1.262	0.504	1.675	1.240	0.619
6	1.216	0.824	1.264	0.558	2.111	1.240	0.695

```
zplane(1, vRoots(m6.ARpoly));
ARp= [m6.ARpoly - 1.64 * m6.seARpoly; ...
                  m6.ARpoly + 1.64 * m6.seARpoly];
varstep(m6.ARpoly, 20, 0, ARp, 5000);
```

Table 4 shows the output forecasts for all models estimated and their standard errors.

3.4 Case study: electricity load demand forecasting in Spain using multi seasonal models

This case study is introduced to show three capabilities of **ECOTOOL**, namely the possibility to estimate multiseasonal ARIMA models, the automatic identification of such complex models and the robustness of the routines, demonstrated in a long automatic experiment. The data is the hourly Spanish electricity load demand, publicly available at the Iberian Energy Market Operator web page (OMIE: `http://www.omie.es`). It is continuously updated since 29 June 2001. Figure 12 shows a portion of such data, from 1 January 2017 to 12 June 2018.

These data are characterised by a number of periodic components superimposed that have to be dealt with, if a comprehensive model wants to be fitted. Firstly, the data exhibits a clear annual cycle with two peaks in winter and summer, respectively, closely related to temperatures. Secondly, a strong diurnal cycle, with different profiles depending on the season of the year. Thirdly, a weekly cycle is present with lower demand during weekends,

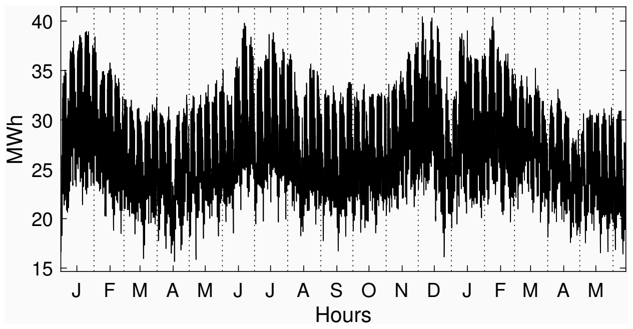

Figure 12: Spanish hourly electricity load demand from January 2017 to June 2018.

mainly due to the absence of industrial activity. Finally, the data is affected by a number of special days, special events, moving festivals and holidays, etc.

In this context is where **ECOTOOL** offers an important innovation, since automatic identification of ARIMA models is extended for these complex type of databases, namely the multi-seasonal ARIMA model, i.e., models that include as many multiplicative seasonal factors as necessary. As far as the author is concerned, this is the first time that an automatic algorithm is developed for such complex cases. Certainly, in this case the model is composed of the multiplication of three ARMA factors: regular, daily and weekly periodicities. The general specification is in equation (8), written with the same nomenclature as equation (1).

$$z_t = \frac{\theta_{Q_0}(B)}{\phi_{P_0}(B)} \frac{\Theta_{Q_1}(B^{24})}{\Phi_{P_1}(B^{24})} \frac{\Theta_{Q_2}(B^{168})}{\Phi_{P_2}(B^{168})} a_t \tag{8}$$

$\theta_{Q_0}(B)/\phi_{P_0}(B)$ is a ratio of polynomials in the back-shift operator of appropriate orders; $\Theta_{Q_1}(B^{24})/\Phi_{P_1}(B^{24})$ is a ratio with orders Q_1 and P_1 in multiples of 24 hours per day; and $\Theta_{Q_2}(B^{168})/\Phi_{P_2}(B^{168})$ is similarly defined, though for 168 hours per week. The multi-seasonal extension of the 'airline' model used later is shown in equation (9).

$$y_t = \frac{(1+\theta B)}{(1-B)} \frac{(1+\Theta_{24}B^{24})}{(1-B^{24})} \frac{(1+\Theta_{168}B^{168})}{(1-B^{168})} a_t \tag{9}$$

The output variable y_t in equation (9) is the undifferenced time series because the differences are included explicitly in the model denominators.

One week ahead forecasts for load demand with an ‘airline’ model (9) and with the automatic identified models using the last three months of data available may be carried out with the following code

```
load Edemand
y = log(Edemand(end - 24 * 31 + 1 : end));
model= '(1+a1*B)(1+a24*B24)(1+a168*B168)/(1-B)(1-B24)(1-B168)';
m1 = modelTF(y, [], model, 168);
m2 = modelAUTO(y, [], [1 24 168], 1, 0, 168);
```

Automatic identification suggests ARIMA$(1, 1, 2) \times (0, 0, 3)_{24} \times (0, 1, 1)_{168}$, rather different to the ‘airline’ model in many respects. Perhaps the most important is that daily differences are not necessary, implying that the ‘airline’ model in equation (9) is strictly wrongly specified, at least because of over-differentiation.

These two models were estimated and used to forecast a week ahead along a full year (from July 2017 to June 2018) with a rolling forecast origin every 24 hours and samples 60 days long. Thence, 365 rounds of 168 hours-ahead forecasts were calculated with each model. The window size (60 days) allow the models to adapt for the changing profile of the periodic components over the year. A full year of data was reserved as the test set to give a better overall idea of forecasting performance, since such performance varies with the season of the year. One additional model was also trained to add a benchmark for comparisons, namely a weekly Naïve of period 168 hours per cycle (a daily naïve with period 24 was also tried, but was discarded because the results were systematically worse).

Two error metrics (included in `toolFORECAST`) are used to evaluate the forecasting accuracy of each model. Both have become most important in the recent forecasting literature, because they have proven very useful in many applications and are free from some inconveniences. They are the symmetric Mean Absolute Percentage Error (sMAPE) and the Mean Absolute Scaled Error (MASE), see equations (10), (11), [33] and [34]. The sMAPE metric avoids the distortions of the standard non-symmetric MAPE criterion and problems for values close to zero. The MASE metric compares the out-of-sample performance of the model with the in-sample performance of a simple seasonal RW (see Table 2), i.e., assuming that the model for the data is $y_t = y_{t-s} + a_t$.

$$\text{sMAPE}_l = l^{-1} \sum_{i=1}^{l} \frac{2 \mid y_{T+i} - \hat{y}_{T+i} \mid}{\mid y_{T+i} \mid + \mid \hat{y}_{T+i} \mid} \times 100 \tag{10}$$

$$\text{MASE}_l = l^{-1} \sum_{i=1}^{l} \frac{\mid y_{T+i} - \hat{y}_{T+i} \mid}{(T-s)^{-1} \sum_{r=s+1}^{T} \mid y_r - y_{r-s} \mid} \tag{11}$$

Results for sMAPE and MASE metrics are shown in Table 5 for a selection of forecast horizons. Some relevant observations follow. Firstly, the forecast performance deteriorates with the forecast horizon for all methods. Secondly, Naïve is the worst model for all horizons, with the only exception of very long horizons (above 5 days) according exclusively to the sMAPE metric. Thirdly, the *Auto* method is the best for all horizons, meaning that the automatic identification implemented in **ECOTOOL** makes sense in terms of forecasting performance.

Table 5: Median sMAPE and MASE metrics for different models on the electricity demand data.

	sMAPE			MASE		
Steps	Airline	Auto	Naïve	Airline	Auto	Naïve
1	2.364	2.390	4.507	0.287	0.291	0.512
2	2.732	2.701	4.277	0.463	0.450	0.773
3	2.951	2.943	4.710	0.602	0.585	0.960
4	3.117	3.159	5.076	0.716	0.693	1.111
5	3.248	3.301	5.478	0.783	0.792	1.249
6	3.338	3.382	5.595	0.851	0.876	1.379
7	3.357	3.329	5.564	0.921	0.919	1.446
8	3.529	3.601	5.510	0.992	1.007	1.494
9	3.624	3.756	5.426	1.068	1.074	1.591
10	3.682	3.733	5.339	1.122	1.115	1.670
11	3.745	3.721	5.435	1.166	1.180	1.777
12	3.675	3.693	5.388	1.197	1.219	1.871
1 day	3.618	3.520	5.455	1.581	1.584	2.380
2 days	4.472	4.167	5.434	2.043	2.024	2.915
3 days	5.112	4.608	5.854	2.426	2.350	3.249
4 days	5.451	4.924	5.927	2.705	2.618	3.460
5 days	5.917	5.130	6.019	2.931	2.803	3.667
6 days	6.470	5.263	5.904	3.161	2.943	3.807
7 days	6.678	5.549	5.788	3.455	3.007	3.980

4 ECOTOOL Reference

4.1 Graphical User Interface functions and demos

ECOTOOLdemos

ECOTOOLdemos   ECOTOOL demos

```
ECOTOOLdemos(number)
```

INPUTS:

`number`: a number from 1 to 11
(1) Overview on Monthly Spanish electricity and gas demand
(2) Overview on hourly Spanish electricity demand and prices
(3) ARIMA modelling
(4) Unemployment in Spain
(5) Nile data
(6) US GDP and the Hodrick-Prescott filter
(7) Steel consumption in the UK
(8) SCC example
(9) Lydia Pynkham advertisement sales data
(10) Data interpolation and signal extraction
(11) Gas furnace data

toolFORECAST

---

toolFORECAST   Forecasting tool

`toolFORECAST(y, py, stdpy, origin)`

This is a Graphical User Interface for forecast representation, error calculation and forecasting comparisons. A 'Series' menu appears when time series are multivariate and an additional 'Forecasts' menu unfolds when several forecasts are provided for each time series.

Some additional parameters of this GUI may be customized:

- Save position and size of windows (File → Save Windows Positions).
- Reset position and size of windows (File → Reset Windows Positions).
- Information about this GUI (File → About Test Tool).

All the output in tables appear in a standard MATLAB figure window (called Text Output Window) that allows for certain useful customization: In addition, some additional parameters of this GUI may be customized:

- Copy the table to the clipboard (Edit → Copy Text to Clipboard).
- Send the table to the command window (Edit → Send to Command Window).
- Increase font size (Edit → Increase Font Size).
- Decrease font size (Edit → Decrease Font Size).

INPUTS:

`y`:	(*) Vector of actual data (m x T or T x m)
`py`:	(*) Vector of forecast data (n*m x any or any x n*m)
`stdpy`:	Confidence bands (2*n*m x any or any x 2*n*m) This is the typical output stdpz of modelTF or modelVARX
`origin`:	Forecast origin (1 x 1)

It is possible to plot as many forecasts as desired for any set of time series.

The length of the actual values and the forecasts do not need to be the same and they are adjusted by means of the input ‘origin’.

See Also: modelTF, modelVARX, toolTEST

toolTEST

---

toolTEST Graphical User Interface to access diagnostic tools

`tooltest(y, 'names', stringnames, 'fun1', opt1, ...)`

This is a Graphical User Interface ideal for all sort of statistical operations, like displaying descriptive information about variables, identification tools, diagnostic checking of models, etc. All this information may be accessed through a 'Tests' menu, together with a 'Series' menu (only appears if time series are multivariate) and 'Options' menu. All the options available in the 'Tests' menu may be run also by calling appropriate functions (see description below).

Some additional parameters of this GUI may be customized:

- Save position and size of windows (File → Save Windows Positions).
- Reset position and size of windows (File → Reset Windows Positions).
- Information about this GUI (File → About Test Tool).

All the output in tables appear in a standard MATLAB figure window (called Text Output Window) that allows for certain useful customization: In addition, some additional parameters of this GUI may be customized:

- Copy the table to the clipboard (Edit → Copy Text to Clipboard).
- Send the table to the command window (Edit → Send to Command Window).
- Increase font size (Edit → Increase Font Size).
- Decrease font size (Edit → Decrease Font Size).

INPUTS:

y: (*) Vector time series data (m x T or T x m)
The rest of inputs are optional and have to be supplied by pairs of variables, the first indicate a function name, the second the inputs to that function. opt1, opt2, ... has to be

cell variables
Valid Function names are (see the help of all
these functions to know what the pair input can be):
plotband, vboxplot, vtboxplot, vscatter, sumstats, vhist,
vqqplot, gausstest, vident, vident2, varsbc, vgranger,
vcusum, stdmean, testhet, vboxcox, adftest, vphillips,
johansentest, NLtests, vtsay, varsbc2, vbds,
vcumperiod, vspectrum, varspectrum

'names' is used to supply a set of names for the variables in
input y
(may be a string or a string matrix)
All the options for any function may be changed within the
GUI, making the first input the only one compulsory

Examples:
```
y= randn(300, 3);
toolTEST(y);
toolTEST(y, 'names', 'Series1 Series2 Series3');
toolTEST(y, 'vident', 50, 4, 1, 'varspectrum', 11);
```

See Also: toolFORECAST

4.2 Time Series modeling functions

modelAUTO

modelAUTO Automatic identification of ARIMA models

```
model= modelAUTO(y, u, s, diffs, boxcox, nofs, outlier, method)
```

```
model= modelAUTO(y, 'property1', v1, 'property2', v2, ...)
```

INPUTS or Property values (string variables):

`y`:	(*) Output time series data (1 x T or T x 1)
`u`:	Vector of input time series data (Nu x T or T x Nu)
`s`:	Seasonal periods including 'regular' period, e.g. for regular and monthly operators on monthly data s = [1 12]
`diffs`:	Test for differencing on/off in the identification procedure (1/0)
`boxcox`:	Transform data with boxcox transformation before running identification
`nofs`:	Forecast information. If model is univariate ARIMA, nofs is just the number of forecasts ahead (1 x 1) If model is TF with deterministic inputs, nofs are the forecasts for all the inputs into the future (any x k) If model is TF with stochastic inputs, nofs is the model for each input, the format is the same as the output 'polys' to this function (a cell of vectors with the polynomial values). Forecasts for the inputs and standard deviations are computed inside the function
`outlier`:	Automatic detection of Additive (AO), Innovative (IO), Level Shift (LS) and Transitory Change (TC) outliers (1 $\times$ n, n=1 ... 7). First element is the constant reference (default 3) Second to fifth are [AO IO LS TC] on/off outliers detection Sixth element is the denominator for TC outliers Seventh element is the number of iterations (default is 3)

`method`:	Either a single string or a cell 1×2 method1 may be either 'EML' or -1 for Exact Maximum Likelihood, or 'CML' or 0 for Conditional Sum of Squares (Default) 'NONE' for no estimation, then par0 are used as parameters. method2 are the initial values (out of sample) of the input variables u

OUTPUT:

model is a structure with a number of fields, including all the inputs and the following ones, that are considered outputs to the function.

`e`:	Residuals of model
`py`:	Forecasts of output variable
`stdpy`:	Standard deviations of forecasts
`model`:	String containing the model identified (input to modelTF)
`par`:	Vector of parameter estimates
`covpar`:	Covariance matrix of estimates
`y0`:	Output series with outliers removed and interpolation of missing values
`tfout`:	Transfer funciton responses for all inputs
`uout`:	All inputs in the final model
`table`:	Estimation table of final model

See Also: **modelTF, modelVARX, modelETS, modelUC, modelNAIVE**

modelETS

---

modelETS Exponential Smoothing modelling

```
model = modelETS(y, u, model, nofs, method, outlier, par0)

model = modelETS(y, 'property1', v1, 'property2', v2, ...)
```

INPUTS:

`y`:	(*) Output time series data ($1 \times$ T or T $\times 1$)
`u`:	Vector of input time series data (Nu $\times$ T or T $\times$ Nu)
`model`:	Model Structure. Cell of strings with k+1 elements containing the ratio of polynomials written as a function of the backwardshift operator B. The first element is the ES model for the noise. The rest are the k TF functions. Types of ES models are: 'NN': No trend, no seasonal 'AN': Additive trend, no seasonal 'DN': Damped trend, no seasonal 'NA#': No trend, additive seasonal of period # 'AA#': Additive trend, additive seasonal of period # 'DA#': Damped trend, additive seasonal of period #
`nofs`:	Forecast information. If model is univariate ES, nofs is just the number of forecasts ahead (1×1) If model is TF with deterministic inputs, nofs are the forecasts for all the inputs into the future (any x k)
`method`:	'EML' or -1 for Exact Maximum Likelihood 'CML' or 0 for Conditional Sum of Squares (Default)
`outlier`:	Automatic detection of Additive (AO), Innovative (IO), Level Shift (LS) and Transitory Change (TC) outliers (1×1, 1×2 or 1×3). First element is the constant reference (default 4) Second element is the number of iterations (default is 3) Third element is the denominator for TC outliers
`par0`:	Vector of inital estimates to start the search

OUTPUT:
model is a structure with a number of fields, including all the inputs and the following ones, that are considered outputs to the function.

`e`:	Residuals of model
`py`:	Forecasts of output variable
`stdpy`:	Standard deviations of forecasts
`comp`:	Level, seasonal, irregular, and outlier components
`par`:	Vector of parameter estimates
`covpar`:	Covariance matrix of estimates
`y0`:	Output series with outliers removed and interpolation of missing values
`table`:	Estimation table of final model

Examples:

```
m1 = modelETS(z, [], 'AA12', 24, 4);
m1 = modelETS(z, u, 'AA12', 'w0/(1+d1B)', ones(12, 1));
```

See Also: modelVARX, modelTF, modelUC, modelAUTO, modelNAIVE

modelNAIVE

modelNAIVE	Forecasting with naive methods (mean, naive, naive seasonal, mean naive, mean change, mean mean change)

```
model = modelNAIVE(y, nofs, s)
```

```
model = modelAUTO(y, 'property1', v1, 'property2', v2)
```

INPUTS:

`y`:	(*) Output time series data (1 x T or T x 1)
`nofs`:	Number of forecasts
`s`:	Seasonal period

OUTPUT:

model is a structure with a number of fields, including all the inputs and the following ones, that are considered outputs to the function.

`py`:	Forecasts of output variable
	Column 1: Mean method
	Column 2: Naive method
	Column 3: Naive seasonal method
	Column 4: Mean naive method
	Column 5: Drift method
	Column 6: Mean Drift method

See Also: **modelVARX, modelETS, modelUC, modelTF, modelAUTO**

modelTF

modelTF	MISO Transfer Function analysis

$$y_t = \frac{\omega_{r1}}{\delta_{d1}} u_{1,t} + \frac{\omega_{r2}}{\delta_{d2}} u_{2,t} + \ldots + \frac{\omega_{rk}}{\delta_{dk}} u_{k,t} + \frac{\theta_q}{\phi_p} a_t$$

```
model = modelTF(y, u, model, nofs, method, outlier, par0)

model = modelTF(y, 'property1', v1, 'property2', v2, ...)
```

INPUTS:

`y`:	(*) Output time series data (1 x T or T x 1)
`u`:	Vector of input time series data (Nu x T or T x Nu)
`model`:	Model Structure. Cell of strings with k+1 elements containing the ratio of polynomials written as a function of the backwardshift operator B The first element is the model for the noise. The rest are the k TF functions. See examples below
`nofs`:	Forecast information. If model is univariate ARIMA, nofs is just the number of forecasts ahead (1 x 1) If model is TF with deterministic inputs, nofs are the forecasts for all the inputs into the future (any x k) If model is TF with stochastic inputs, nofs is the model for each input, the format is the same as the output 'polys' to this function (a cell of vectors with the polynomial values). Forecasts for the inputs and standard deviations are computed inside the function
`method`:	Either a single string or a cell 1x2 method1 may be either 'EML' or -1 for Exact Maximum Likelihood, or 'CML' or 0 for Conditional Sum of Squares 'NONE' for no estimation, then par0 are used as parameters method2 are the initial values (out of sample) of the input variables u
`outlier`:	Automatic detection of Additive (AO), Innovative (IO), Level Shift (LS) and Transitory Change (TC) outliers

	(1xn, n=1 ... 7). First element is the constant reference (default 3) Second to fifth are [AO IO LS TC] on/off outliers detection Sixth element is the denominator for TC outliers Seventh element is the number of iterations (default 3)
`par0`:	Vector of inital estimates to start the search

OUTPUT:

model is a structure with a number of fields, including all the inputs and the following ones, that are considered outputs to the function.

`e`:	Residuals of model
`py`:	Forecasts of output variable
`stdpy`:	Standard deviations of forecasts
`polys`:	Numerical cell array with estimated polynomials
`par`:	Vector of parameter estimates
`covpar`:	Covariance matrix of estimates
`y0`:	Output series with outliers removed and interpolation of missing values
`tfout`:	Transfer funciton responses for all inputs
`uout`:	All inputs in the final model
`modelout`:	Full model, including models for outliers
`table`:	Estimation table of final model

Examples:

```
model = '(1+ma1B)(1+ma12B12)/(1-B)(1-B12)';
m1= modelTF(z, [], model, 24, 4);
model = {'(1+ma1B)(1+ma12B12)/(1-B)(1-B12)',
         'w0/(1+d1B)'};
m1= modelTF(z, u, model, ones(12, 1));
```

See Also: modelVARX, modelETS, modelUC, modelNAIVE, modelAUTO

modelUC

---

modelUC   Unobserved Components modelling

```
model = modelUC(y, u, model, nofs, method, outlier, par0)

model = modelUC(y, 'property1', v1, 'property2', v2, ...)
```

INPUTS:

`y`:	(*) Output time series data (1 x T or T x 1)
`u`:	Vector of input time series data (Nu x T or T x Nu)
`model`:	Model Structure. Cell of strings with k+1 elements containing the ratio of polynomials written as a function of the backwardshift operator B. The first element is the UC model for the noise. The rest are the k TF functions Types of UC models are: 'HP#': Hodrick-Prescott filter with lambda # If lambda is omitted, it is estimated 'RW': Random Walk trend, no seasonal 'IRW': Integrated Random Walk trend, no seasonal 'LLT': Local Linear trend, no seasonal 'SRW': Smoothed trend, no seasonal 'RW#': Random Walk trend, seasonal of period # 'IRW#': Integrated Random Walk trend, period # 'LLT#': Local Linear trend, seasonal of period # 'SRW#': Smoothd Random Walk trend, period #
`nofs`:	Forecast information. If model is univariate UC, nofs is just the number of forecasts ahead (1 x 1). If model is Regression terms, nofs are the forecasts for all the inputs into the future (any x k)
`method`:	'EML' or -1 for Exact Maximum Likelihood 'CML' or 0 for Conditional Sum of Squares (Default)
`outlier`:	Automatic detection of Additive (AO), Innovative (IO), Level Shift (LS) and Transitory Change (TC) outliers (1x1, 1x2 or 1x3). First element is the constant reference (default 4)

	Second element is the number of iterations (default is 3)
	Third element is the denominator for TC outliers
`par0`:	Vector of inital estimates to start the search

OUTPUT:
model is a structure with a number of fields, including all the inputs and the following ones, that are considered outputs to the function.

`e`:	Residuals of model
`py`:	Forecasts of output variable
`stdpy`:	Standard deviations of forecasts
`comp`:	Level, seasonal, irregular, and outlier components
`par`:	Vector of parameter estimates
`covpar`:	Covariance matrix of estimates
`y0`:	Output series with outliers removed and interpolation of missing values
`comps`:	Standard deviation of components (third column correspond to output)
`table`:	Estimation table of final model

Examples:

```
m1= modelUC(z, [], {[nan nan], [12 6 4 3 2.4 2],
           [1 1 1 1 1 1], nan, nan}, 24);
m1= modelUC(z, u, {[nan nan], [12 6 4 3 2.4 2],
          [1 1 1 1 1 1], nan, nan, [nan nan]},
          ones(12, 1));
```

See Also: modelVARX, modelTF, modelETS, modelNAIVE, modelAUTO

modelVARX

---

modelVARX VAR model with eXogenous variables analysis

$$\mathbf{\Phi}_p(B)\mathbf{y}_t = \mathbf{\Omega}_r(B)\mathbf{u}_t + \mathbf{e}_t$$

```
model = modelVARX(y, u, model, nofs, diffs, s, iter)
```

```
model = modelVARX(y, 'property1', v1, 'property2', v2, ...)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`u`:	Vector of input time series data (Nu x T or T x Nu)
`model`:	(*) Model orders. Cell of vectors or matrices 1 x 1 or 1 x 2 The first element correspondes to polynomial ARpoly(B) The second element corresponds to polynomial Xpoly(B) Each of these element may be a vector of specific orders or a matrix A matrix is the multivariate polynomial (m x any) in which the positions to estimate are marked with a NaN First block of coefficients in polynomial ARpoly(B) must be the identity matrix First block of coefficients in polynomial Xpoly(B) corresponds to lag 0, i.e. contemporaneous relation between inputs and outputs See examples below
`nofs`:	Forecast information. If model has no inputs, nofs is just the number of forecasts ahead (1 x 1) If model includes inputs, nofs nofs are the forecasts for all the inputs (any x k)
`diffs`:	Difference orders for outputs (the same as input 'order' to vdif function)
`s`:	Types of differences (the same as 's' to vdif function)
`iter`:	Maximum number of iterations when constraints

are imposed upon the model (1 x 1)

OUTPUT:

model is a structure with a number of fields, including all the inputs and the following ones, that are considered outputs to the function.

`e`:	Residuals of model
`py`:	Forecasts of output variables
`stdpy`:	Standard deviations of forecasts
`ARpoly`:	Estimated AR polynomial
`Xpoly`:	Estimated eXogenous polynomial
`seARpoly`:	Standard Error of AR polynomial
`seXpoly`:	Standard Error of eXogenous polynomial
`sigma`:	Residual covariance matrix
`table`:	Estimation table of final model

Examples:

```
m1 = modelvarx(z, [], [1 6], 12);
model = {[1 0 0 0 nan nan; 0 1 0 0 0 nan]}
m1 = modelvarx(z, [], model, 12);
m1 = modelvarx(z, u, {[1 6], [0 3]}, ones(12, 1));
model = {eye(2), [nan nan; 3 nan]}
m1 = modelvarx(z, u, model, ones(12, 1));
```

See Also: modelTF, modelETS, modelUC, modelAUTO, modelNAIVE

4.3 Calendar effects functions

days

days	Creates dummy variable to account for the number of days in period in monthly or quarterly data

```
I = days(starting_date, N, s)
```

INPUTS:

`starting_date`:	(*) Starting date of data [yyyy mm] or [yyyy q]
`N`:	(*) Number of monthly or quarterly observations to simulate
`s`:	Seasonal period of data (12 or 4) (12)

OUTPUTS:

`I`:	Dummy variable Nx1

Examples:

```
I= days([1990 6], 148);
```

See Also: easter, trading, leapy

easter

easter	Creates dummy Easter variables to test for Easter effects on monthly or quarterly data

`I = easter(starting_date, N, define, ttype, s)`

INPUTS:

`starting_date`:	(*) Starting date of data [yyyy mm] or [yyyy q]
`N`:	(*) Number of monthly or quarterly observations to simulate
`define`:	([0 5]) Matrix 1x2 1xANY ANYxANY that defines the type of easter intervention reference_day is a positive or negative integer to specify the reference day. 0: Easter Sunday [reference_day Number_of_days_to_and_including_Sunday] all years equal [reference_day W1 W2 W3 W4 ...] sum of weights should be 1. All years equal [r_d1 W1 W2 W3 W4; r_d2 A1 A2 0 0; ...] One row for each year
`ttype`:	0 percentage of easter days in March/April 1 idem but compensating the increase in one month with a decrease in the other
`s`:	Seasonal period of data (12 or 4) (12)

OUTPUTS:

`I`:	Dummy Easter variable of length N

Examples:

```
I= easter([1990 6], 148, [0 8]);
I= easter([1990 6], 148, [0 8; -3 5]);
I= easter([1990 6], 148, [0 1/3 1/3 1/3 0 0;
          3 1/5 1/5 1/5 1/5 1/5]);
```

See Also: trading, days, leapy

leapy

---

leapy   Dummy variable for leap year intervention in either monthly

---

```
I = leapy(starting_date, N, ttype, S)
```

INPUTS:

`starting_date`:	(*) Starting date of data [yyyy mm] or [yyyy q]
`N`:	(*) Number of monthly or quarterly observations to simulate
`ttype`:	0 mark all februaries with 29 days 1 consider year of 365.25 days
`S`:	Seasonal period of data (12 or 4) (12)

OUTPUTS:

`I`:	Dummy variable Nx1

Examples:

```
I= leapy([1990 6], 148);
```

See Also: easter, trading, days

trading

trading	Creates dummy Trading day variables

```
I = trading(starting_date, N, ttype, feasts, S)
```

INPUTS:

`starting_date`:	(*) Starting date of data [yyyy mm] or [yyyy q]
`N`:	(*) Number of monthly or quarterly observations to simulate
`ttype`:	Type of dummies (0): 0 for just (5 weekday) vs (2 weekend days), 1 for just (6 weekday) vs (1 weekend day), 2 when all days are different
`feasts`:	Nx1 (if ttype= 0 or 1) or Nx6 (if ttype= 2) of moving feastivals in weekdays
`S`:	Seasonal period of data (12 or 4) (12)

OUTPUTS:

`I`:	Dummy variable Nx1 (ttype= 0 or ttype= 1) or Nx6 (ttype= 2)

Examples:

```
I = trading([1990 6], 148);
```

See Also: easter, days, leapy

4.4 General purpose functions

acft

acft	Theoretical Autocorrelations for ARMA(p, q) processes

$$y_t = \frac{\theta_q(B)}{\phi_p(B)} a_t$$

```
tab= acft(MApoly, ARpoly, ncoef, s)
```

INPUTS:

`MApoly`:	MA polynomial ($[1, \theta_1, \theta_2, \ldots, \theta_q]$).
`ARpoly`:	AR polynomial ($[1, \phi_1, \phi_2, \ldots, \phi_p]$).
`ncoef`:	Number of ACF and PACF coefficients (38).
`s`:	Samples in the season (12).

OUTPUT:

`tab`:	[lag, ACF, PACF].

Examples:

```
acft(1, [1 -0.8]);
acft(conv([1 0.8], [1 zeros(1, 11) -0.8]), [1 -0.8]);
```

ADFtest

ADFtest Augmented Dickey-Fuller Unit Roots tests

`ADFtest(y, lags)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`lags`:	Maximum lag (1 x 1) or vector of lags (any x 1) to include in test regression

Examples:

```
adftest(y);
adftest(y, 10);
adftest(y, (5 :  10));
```

See Also: vphillips, Johansentest

corrmatrix

---

corrmatrix Builds correlation matrix from covariance matrix

`corrmat= corrmatrix(covmat)`

INPUTS:

`covmat`:	(*) Covariance matrix, squared and semipositive definite (any x any)

OUTPUTS:

`corrmat`:	Correlation matrix

See Also: nancorr, nancov, nanprod

delay

---

delay   Lags a vector of variables

```
ylag = delay(y, lags)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`lags`:	(*) Vector or matrix of lags (1 x any)

OUTPUTS:

`ylag`:	Matrix of lagged variables

Examples:

```
y= repmat((1 :  10)', 1, 3);
delay(y, [0 1 2])
delay(y(:, 1), [0 2 4])
```

gausstest

---

gausstest   Gaussianity tests

```
gausstest(y, dec)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`dec`:	Number of decimals for table output (1 x 1)

See Also: vhist, vqqplot

Johansentest

Johansentest	Johansen cointegration tests

```
johansentest(y, lags)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m) For univariate time series the test is a unit root test
`lags`:	Maximum lag (1 x 1) or vector of lags (any x 1) to include in test regression

Examples:

```
johansentest(z);
johansentest(z, 10);
johansentest(z, (5 :  10), 3);
```

See Also: ADFtest, vphillips

nancorr

nancorr	Correlation matrix of vector variables with NaN values

```
corry= nancorr(y)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)

OUTPUTS:

`corry`:	Correlation matrix

See Also: nancov, nancumsum, nanprod

nancov

nancov	Covariance matrix of vector variables with NaN values

```
covy = nancov(y)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)

OUTPUTS:

`covy`:	Covariance matrix

See Also: nancorr, nancumsum, nanprod

NLtests

NLtests	Non Linear tests based on squares

```
NLtests(y, ncoef)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`ncoef`:	Number of autocorrelation coefficients (1 x 1)

See Also: vtsay, varsbc2, vbds

plotband

---

plotband   Vector Time Series plot with confidence bands

```
plotband(y, s, standard, fixedscale)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`s`:	Seasonal period for subplots (1 x 1)
`standard`:	Standard plot on/off (1 / 0)
`fixedscale`:	Fixed vertical scale on/off (1 / 0)

See Also: vboxplot, vtboxplot, vscatter, sumstats

stdmean

---

stdmean   Standard Deviation vs Mean type of plots

```
stdmean(y, s, opt, numbers)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`s`:	Seasonal period (1 x 1)
`opt`:	0 for Standard deviation-Mean; 1 for MEDA-Median; 2 for Range-Mean
`numbers`:	numbers for each data point on/off (1/0)

See Also: vcusum, testhet, vboxcox, vboxcoxinv

sumstats

sumstats	Summary descriptive statistics

`sumstats(y, dec)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`dec`:	Number of decimals for table output (1 x 1)

See Also: plotband, vboxplot, vtboxplot, vscatter

varsbc

varsbc	Identification criteria on VAR models

`varsbc(y, ncoef)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`ncoef`:	Number of autocorrelation coefficients (1 x 1)

See Also: vident, vident2, vgranger

varsbc2

---

varsbc2 Non-linearity test based on Schwarz Criterion of squares

---

`varsbc2(y, ncoef)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`ncoef`:	Number of autocorrelation coefficients (1 x 1)

See Also: NLtests, vtsay, vbds

vspectrum

---

varspectrum AR spectrum

`vspectrum(y, smooth, nfreq, xlog)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`smooth`:	Length of smoothing truncating window (1 x 1, default is 1)
`nfreq`:	Number of frequencies to estimate the spectrum
	(1 x 1, use 0 or T/2 for periodogram frequencies)
	xlog: Frequency and Power axis log on/off (1 x 2)
	[0 0]: linear scale for both axes
	[1 0]: log scale for frequency axis
	[0 1]: log scale for power axis
	[1 1]: log scale for both axes

See Also: vcumperiod, varspectrum

varstep

varstep	Var impulse and step function analysis

```
[z, zmax, zmin] = varstep(ARpoly, n, typefun, sdAR, Nsimul)
```

INPUTS:

`ARpoly`:	(*) Vector AR polynomial (m x Na*m) Leading matrix coefficient may different from identity matrix (structural models)
`n`:	Maximum lag for impulse or step analysis (1 x 1)
`typefun`:	Impulse or step function (0/1)
`sdAR`:	Minimum and Maximum values for AR parameters for impulse or step error bounds by a Montecarlo simulation (2*m x Na*m)
`Nsimul`:	Number of runs for Montecarlo simulation for error bounds (1 x 1)

OUTPUTS:

`z`:	Impulse or step function (m x m x n)
`zmax`:	Maximum boundary for impulse or step function for Montecarlo simulation (m x m x n)
`zmin`:	Minimum boundary for impulse or step function for Montecarlo simulation (m x m x n)

vboxcox

vboxcox   Box-Cox homokedasticity family of transformations

```
[yt, lambdas,lambdasv]=vboxcox(y, lambdas, group, axislambdas)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`lambdas`:	Lambdas for each time series (1 x n, with n¡= m) If is empty or not supplied the lamdas are estimated by ML
`group`:	Number of elements for grouping for Guerrero (1993)
`axislambdas`:	Lambda axis for graphical output (1 x any)

OUTPUTS:

`yt`:	Transformed data according to lambdas (either supplied by the user or estimated)
`lambdas`:	Estimated lambdas (univariate estimation for each time series)
`lambdasv`:	Estimated lambdas (multivariate estimation)

Examples:

```
vboxcox(z);
zt= (z);
zt= (z, 0);
```

See Also: vcusum, stdmean, testhet, vboxcoxinv

vboxcoxinv

---

vboxcoxinv Inverse of Box-Cox transformation

```
yt = vboxcoxinv(y, lambdas, const)
```

INPUTS:

`y`:	(*) Vector time series data previously transformed with vboxcox (m x T or T x m)
`lambdas`:	Lambdas for each time series (1 x n, with n<= m)
`const`:	Constants that were added to the original series in order to avoid negative values (1 x m)

OUTPUTS:

`yt`:	Transformed data according to lambdas (either supplied by the user or estimated)

See Also: vcusum, stdmean, testhet, vboxcox

vboxplot

---

vboxplot Vector Time Series boxplot

```
vboxplot(y, s)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`s`:	Seasonal period for subplots (1 x 1)

See Also: plotband, vtboxplot, vscatter, sumstats

vConv

vConv Multiplication of vector polynomials

```
C = vConv(A, B, cA, cB);
```

INPUTS:

`A`:	(*) Vector polynomial
`B`:	(*) Vector polynomial
`cA`:	Number of columns that define block parameters of poly A
`cB`:	Number of columns that define block parameters of poly B

OUTPUTS:

`C`:	Multiplication of vector polynomials A and B

See Also: vdif, vfilter, vRoots

vcumperiod

vcumperiod Cumulative periodogram

```
vcumperiod(y)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)

See Also: vspectrum, varspectrum

vcusum

---

vcusum CUSUM and CUSUM of squares tests

`vcusum(y)`

INPUTS:

`y`: (*) Vector time series data (m x T or T x m)

See Also: stdmean, testhet, vboxcox, vboxcoxinv

vdif

---

vdif Differencing vector variables

`yd = vdif(y, order, s)`

INPUTS:

`y`: (*) Vector time series data (m x T or T x m)

`order`: Difference orders for multiplicative polynomials (1 x any, default is 1)

`s`: Seasonal parameter for each polynomial (1 x any, default is 1)

OUTPUTS:

`yd`: Differenced data (m x ¡T)

Examples:

```
y= randn(500, 3);
vdif(y, [2 1], [1 12])
vdif(y, [2 1; 1 0], [1 12])
vdif(y, [2 1; 1 0; 2 1], [1 12])
vdif(y, [1 1 1], [1 24 168])
```

See Also: vfilter, vConv, vRoots

vgranger

---

vgranger Granger causality tests on VAR models

`vgranger(y, lags, tests)`

INPUTS:

`y`: (*) Vector time series data (m x T or T x m)

`lags`: Vector of lags to include in VAR model (block constraints are possible, 1 x any)

`tests`: Type of tests to perform (string)
Variables are indicated by letters in alphabetical order, each test is separated by one space, '-' separates endogenous from exogenous.
`'a-b b-a'`: two tests, i.e. second variable causes first and viceversa
`'a-bc bc-a'`: two tests, i.e. second and third variable cause the first one and first variable causes second and third

Examples:

```
vgranger(y);
vgranger(y, [3 10]);
vgranger(y, [1 :  10], 'a-b ab-c');
```

See Also: vident, vident2, varsbc

vhist

vhist	Histogram with gaussian distribution and non-parametric estimate

`vhist(y, bins)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`bins`:	Number of bins for histograms (1 x 1)

See Also: vqqplot, gausstest

vident

vident	Summary descriptive statistics

`vident(y, ncoef, s, Npar)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`ncoef`:	Number of autocorrelation coefficients (1 x 1)
`s`:	Seasonal period for plots (1 x 1)
`Npar`:	Number of estimated parameters (if z is a set of residuals, 1 x 1)

See Also: vident2, varsbc, vgranger

vident2

---

vident2 Multivariate sample autocorrelation functions

`vident2(y, ncoef, s, Npar)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`ncoef`:	Number of autocorrelation coefficients (1 x 1)
`s`:	Seasonal period for plots (1 x 1)
`Npar`:	Number of estimated parameters (if z is a set of residuals, 1 x 1)

See Also: vident, varsbc, vgranger

vphillips

---

vphillips Phillips Perron Unit Roots tests

`vphillips(y, lags, ncovs)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`lags`:	Maximum lag (1 x 1) or vector of lags (any x 1) to include in test regression
`ncovs`:	Number of autocovariances of residuals to include in the tests

Examples:

```
vphillips(z);
vphillips(z, 10);
vphillips(z, (5 :  10), 3);
```

See Also: ADFtest, Johansentest

vqqplot

vqqplot	Histogram, gaussian distribution and non-parametric estimate

```
vqqplot(y)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)

See Also: vhist, gausstest

vRoots

vRoots	Calculates the roots of a vector polynomial

```
r = vRoots(A)
```

INPUTS:

`A`:	(*) Vector polynomial (m x Na*m)

OUTPUTS:

`r`:	Roots of A

See Also: vdif, vfilter, vConv

vscatter

---

vscatter Scatter plots

`vscatter(y, bins)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`bins`:	Number of bins for histograms (1 x 1)

See Also: plotband, vboxplot, vtboxplot, sumstats

vspectrum

---

vspectrum Spectrum

`vspectrum(y, smooth, nfreq, xlog)`

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`smooth`:	Length of smoothing truncating window (1 x 1, default is 1)
`nfreq`:	Number of frequencies to estimate the spectrum (1 x 1, use 0 or T/2 for periodogram frequencies)
`xlog`:	Frequency and Power axis log on/off (1 x 2) `[0 0]`: linear scale for both axes `[1 0]`: log scale for frequency axis `[0 1]`: log scale for power axis `[1 1]`: log scale for both axes

See Also: vcumperiod, varspectrum

vtsay

---

vtsay   Tsay non-linearity tests

```
vtsay(y, ncoef)
```

INPUTS:

`y`:	(*) Vector time series data (m x T or T x m)
`lags`:	Maximum lag lags for regression tests (1 x 1)

See Also: NLtests, varsbc2, vbds

References

[1] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 5th edition, 2015.

[2] R. J. Hyndman and Y. Khandakar. Automatic Time Series Forecasting: The Forecast Package for R. *Journal of Statistical Software*, 3(27):1–22, 2008.

[3] R.S. Tsay. Time series model specification in the presence of outliers. *Journal of the American Statistical Association*, 81(393):132–141, 1986.

[4] V. Gómez and A. Maravall. Automatic Modeling Methods for Univariate Series. In *A Course in Time Series*, pages 171–201. John Wiley & Sons, Inc., 2001.

[5] L. Broze and G. Mlard. Exponential smoothing: Estimation by maximum likelihood. *Journal of Forecasting*, 9(5):445–455, 1990.

[6] Hubert W. Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402, 1967.

[7] Anil K. Bera and Carlos M. Jarque. Efficient tests for normality, homoscedasticity and serial independence of regression residuals: Monte carlo evidence. *Economics Letters*, 7(4):313 – 318, 1981.

[8] G. M. Ljung and G. E. P. Box. On a measure of lack of fit in time series models. *Biometrika*, 65(2):297–303, 1978.

[9] Anna Clara Monti. A proposal for a residual autocorrelation test in linear models. *Biometrika*, 81(4):776–780, 1994.

[10] H. Akaike. A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[11] G. E. Schwarz. Estimating the Dimension of a Model. *Annals of Statistics*, 6:461–464, 1978.

[12] E.J. Hannan and B.G. Quinn. The determination of the order of an autoregression. *Journal of the Royal Statistical Society, Series B*, 41:190195, 1979.

[13] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969.

[14] R. L. Brown, Durbin J., and Evans J. M. Techniques for testing the constancy of regression relationships over time (with discussion). *Journal of the Royal Statistical Society B*, 37:149–192, 1975.

[15] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964.

[16] Vctor M. Guerrero. Timeseries analysis supported by power transformations. *Journal of Forecasting*, 12(1):37 – 48, 1993.

[17] David A. Dickey and Wayne A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74(366a):427–431, 1979.

[18] Peter C. B. Phillips and Pierre Perron. Testing for a unit root in time series regression. *Biometrika*, 75(2):335–346, 1988.

[19] Sren Johansen. Estimation and hypothesis testing of cointegration vectors in gaussian vector autoregressive models. *Econometrica*, 59(6):1551–1580, 1991.

[20] A.I. McLeod and Li W.K. Diagnostic checking arma time series models using squared residual autocorrelations. *Journal of Time Series Analysis*, 4:269–273, 1983.

[21] Daniel Pea and Julio Rodriguez. Detecting nonlinearity in time series by model selection criteria. *International Journal of Forecasting*, 21(4):731 – 748, 2005. Nonlinearities, Business Cycles and Forecasting.

[22] R. S. Tsay. Nonlinearity tests for time series. *Biometrika*, 73(2):461–466, 1986.

[23] W.S. Wei. *Time Series Analysis–Univariate and Multivariate Methods.* Temple University. USA, 2006.

[24] P. C. Young, D. J. Pedregal, and W. Tych. Dynamic Harmonic Regression. *Journal of Forecasting*, 18(6):369–394, 1999.

[25] Francis X. Diebold and Roberto S. Mariano. Comparing predictive accuracy. *Journal of Business & Economic Statistics*, 13(3):253–263, 1995.

[26] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[27] Jinduan Chen and Dominic L. Boccelli. Real-time forecasting and visualization toolkit for multi-seasonal time series. *Environmental Modelling & Software*, 105:244 – 256, 2018.

[28] A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge university press, 1989.

[29] C. J. Taylor, D. J. Pedregal, P. C. Young, and W. Tych. Environmental Time Series Analysis and Forecasting with the Captain Toolbox. *Environmental Modelling & Software*, 22(6):797–814, 2007.

[30] J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2nd edition, 2012.

[31] R. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder. *Forecasting with Exponential Smoothing: the State Space Approach*. Springer Science & Business Media, 2008.

[32] A. C. Harvey and J. Durbin. The effects of seat belt legislation on british road casualties: A case study in structural time series modelling. *Journal of the Royal Statistical Society. Series A (General)*, 149(3):187–227, 1986.

[33] S. Makridakis and M. Hibon. The m3-competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476, 2000.

[34] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679 – 688, 2006.