UComp for Matlab/Octave

User guide

Authors: Nerea Urbina & Diego J. Pedregal

30 June 2020

Contents

1 Overview

UComp is a set of functions for the modelling, identification, validation and forecasting of time series based on univariate structural Unobserved Components models (UC), firstly proposed in a seminal work by [1] and expanded later by many others (see e.g., [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]). This software allows for a comprehensive analysis of time series with a few functions. The software includes a number of novelties with respect to other libraries, but the star novelty is the automatic identification of models along a wide range of possible combinations of UC models (up to 47 different combinations). The library includes cycles, exogenous variables and allows for automatic detection of outliers.

The core functions are written in C++ using Armadillo library for linear algebra. The rest are a number of wrapping functions written in Matlab and Octave that allows for the use of the library as a standard Matlab toolbox by the use of MEX functions.

The work-flow consists of creating an **UComp** object as a *struct* data type in Matlab/Octave, and then, working on the different fields it is composed of. Such objects may be created by functions `UCsetup`, `UCmodel` or `UC` (see detailed information in Section 5). Compulsory inputs to any of these functions are the time series data and its seasonal period (number of observations per year). The user would optionally be able to set the rest of parameters that belongs to any **UComp** object, such as outlier detection, forecast horizon, selecting the information criterion for model selection, etc. (see reference for `UCsetup`). The value of these input parameters will handle the behaviour of the rest of functions.

The remaining functions operate directly on **UComp** structures, modifying the properties through a number of functions that perform standard operations, like filtering and smoothing necessary for detrending, seasonal adjustment, signal extraction, etc.

2 List of files

The included files are divided into four groups: C++ files, Matlab/Octave functions, worked examples and additional files.

- C++ files

  `armaMex.hpp` Armadillo's wrapper for Matlab/Octave integration through MEX

  `armaMexOct.hpp` Armadillo's wrapper for Octave integration through MEX (`modified version of armaMex.hpp`)

  `ARMAmodel.h` Stationary ARMA models with zero mean

  `BSMmodel.h` Basic Structural models

  `DJPTtools.h` Several auxiliary functions for general purposes `optim.h` Quasi-Newton estimation with BFGS inverse Hessian approximation

`SSpace.h` State Space systems class

`stats.h` Statistical tests and other useful statistical functions

`UCompCMatlab.cpp` C++ wrapper for UCompC MEX function for Matlab

`UCompCOctave.cpp` C++ wrapper for UCompC MEX function for Octave

- Matlab/Octave functions

`UC.m` Runs all relevant functions for UC modelling

`UCmodel.m` Estimates and forecasts UC general univariate models

`UCsetup.m` Sets up UC general univariate models

`UCestim.m` Estimates and forecasts UC models

`UCvalidate.m` Shows a table of estimation and diagnosis results for UC models

`UCfilter.m` Runs the Kalman Filter for UC models

`UCsmooth.m` Runs the Fixed Interval Smoother for UC models

`UCcomponents.m` Estimates components of UC models

`UCdisturb.m` Runs the Disturbance Smoother for UC models

`mexUComp.m` Compiles and link the necessary files to build MEX files

- Data examples

`airpas.mat` Contains two time series to run functions

- Additional files

`README.txt` Important information about MEX files

`libblas.lib` precompiled BLAS library for Windows

`liblapack.lib` precompiled LAPACK library for Windows

3 Before using the toolbox

3.1 Information to consider

- Help about any function is available in the usual way, i.e., by typing `help function` (e.g., `help UCestim`).
- Adding the folder where the toolbox is located to the current path is recommended by using the command: `addpath(folderName)` (for example, `addpath('libs')`).
- Functions have an input format check, make sure to respect all data types.
- If the MEX function receives any parameter that does not conform to what is expected by the C++ files, an error not identified by Matlab/Octave will be issued and probably the program had a fatal crash.

3.2 How to build MEX files

MEX files are compiled with function `mexUComp.m`, intended to run as an automatic installer. However, in case it does not work, editing this file is recommended to see how to build MEX files by hand.

In most systems `mexUComp` only needs the first input that tells the folder where Armadillo library lives. If necessary, a second input tells where libraries LAPACK and BLAS or substitutes live. Folder `cpp` includes some pre-compiled versions of these libraries for Windows systems.

For a correct compilation Armadillo, LAPACK and BLAS libraries must be installed or accessible in some way in the system. In Windows, the user has several options: i) use the pre-compiled libraries included in `cpp` folder; ii) use the libraries provided by Armadillo; iii) use the libraries provided by Matlab/Octave; or iv) use some of those in the links in Section 3.3.

MAC/Linux users have more chances to get installer `mexUComp` working, but visiting the installation notes of Armadillo to get information about LAPACK/BLAS libraries would help as well.

The previous paragraphs show actually the road followed by the authors to build the MEX file. Nevertheless, it is highly recommended to visit the following link: `https://es.mathworks.com/help/matlab/matlab_external/before-you-run-a-mex-file.html?lang=en`. Also follow this steps:

1. Download Armadillo from `http://arma.sourceforge.net/download.html`

   The user will find information about LAPACK/BLAS libraries dependencies for each OS in the section named *Installation Notes* of the website.

2. Download a compatible compiler with Matlab: `https://es.mathworks.com/support/requirements/supported-compilers.html`

3. In the command window, type: `mex -setup cpp`. With this command the user should be able to choose a compiler.

4. Build the MEX file linking the source file with Armadillo and LAPACK / BLAS / OpenBLAS libraries. To link files and include folders, use `mexUComp` as stated above or the commands `-I[path]`, `-L[path]`, `-l[library name]`.

   The typical case is to write in the command window: `mex -Ipath_to_armadillo_include -Lpath_to_libraries -llapack -lblas file.cpp`. Depending on the platform, `file.cpp` is replaced by either `UCompCMatlab.cpp` or `UCompCOctave.cpp`, that has been previously copied and renamed into the main folder.

Octave and Mac users can use the lapack/blas libraries provided by their respective platforms (replace step 4 with the command: `mex -Ipath_to_armadillo_include -llapack -lblas UCompC.cpp`) or link to your own libraries as explained above.

3.3 Helpful links

- MEX files functions (Matlab): `https://es.mathworks.com/help/matlab/call-mex-file-functions.html?lang=en`
- MEX files functions (Octave): `https://octave.org/doc/v5.2.0/Mex_002dFiles.html#Mex_002dFiles`
- Building MEX files: `https://es.mathworks.com/help/matlab/matlab_external/build-c-mex-programs.html?lang=en`
- Invalid MEX file errors: `https://es.mathworks.com/help/matlab/matlab_external/invalid-mex-file-error.html?lang=en`
- Run MEX file you receive from someone else: `https://es.mathworks.com/help/matlab/matlab_external/before-you-run-a-mex-file.html?lang=en`
- MEX Platform Compatibility: `https://es.mathworks.com/help/matlab/matlab_external/platform-compatibility.html?lang=en`
- MEX Version Compatibility: `https://es.mathworks.com/help/matlab/matlab_external/version-compatibility.html?lang=en`
- Armadillo library: `http://arma.sourceforge.net/`
- LAPACK libraries: `http://www.netlib.org/lapack/`
- BLAS libraries: `http://www.netlib.org/blas/`
- Pre-compiled LAPACK and BLAS libraries for Windows platforms: `https://icl.cs.utk.edu/lapack-for-windows/lapack/`

4 Unobserved Components Models

The UC models aims at decomposing a time series into meaningful components. A common decomposition is shown in equation (1), where T_t, C_t, S_t, and I_t stand for a trend, cycle, seasonal, irregular components, respectively. The model allows also for linear relationships with k exogenous variables $x_{i,t}$ affected by a set of parameters β_i, $(i = 1, \ldots, k)$.

$$z_t = T_t + C_t + S_t + I_t + \sum_{i=1}^{k} \beta_i x_{i,t} \tag{1}$$

In many practical situations, a simplified version of this model is enough for a good representation of the data, see equation (2).

$$z_t = T_t + S_t + I_t \tag{2}$$

Structural methods take equations (1) or (2) as the base model (they are actually the observation equation of a State Space (SS) system, see below) and directly specify the dynamic models for each of the components, for which there is a wide range of possibilities. In general, all components are assumed to be stochastic, trends must be non-stationary by definition, seasonal and cyclical components must show sinusoidal behavior, and irregular components are generally specified as white or coloured noises. The particular models chosen in this paper for each component stem from a long tradition, see e.g., [1, 2, 3, 7].

4.1 Trend components

All trends considered in **UComp** are particular cases of the Generalised Random Walk model (or Damped Trend, DT) shown in equation (3), where T_t^* is usually referred to as the trend 'slope', $0 \leq \alpha \leq 1$, $\eta_{T,t}$ and $\eta_{T,t}^*$ are independent Gaussian white noise sequences with variances $\sigma^2_{\eta_T}$ and $\sigma^2_{\eta_T^*}$, respectively.

$$\begin{bmatrix} T_{t+1} \\ T_{t+1}^* \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & \alpha \end{bmatrix} \begin{bmatrix} T_t \\ T_t^* \end{bmatrix} + \begin{bmatrix} \eta_{T,t} \\ \eta_{T,t}^* \end{bmatrix} \tag{3}$$

This model subsumes the following particular cases: i) Random Walk (RW), setting $\alpha = 0$, $\sigma^2_{\eta_T^*} = 0$ and $T_1^* = 0$; ii) RW with drift, same as previous, but with $T_1^* \neq 0$; iii) Integrated Random Walk (IRW) with $\alpha = 1$ and $\sigma^2_{\eta_T} = 0$; iv) Local Linear Trend (LLT) with $\alpha = 1$.

4.2 Cyclical components

Cycles are taken from [1] and obey equation (4). Here, C_t^* is an additional state necessary to define the model; ρ is a damping factor taking values between 0 and 1; ω is the frequency of the cycle, namely $\omega = 2\pi/P$, where P is the period (the number of observations per one full oscillation); and η_t and η_t^* are mutually independent Gaussian white noises with common variance σ^2_η.

$$\begin{bmatrix} C_{t+1} \\ C^*_{t+1} \end{bmatrix} = \rho \begin{bmatrix} cos\omega & sin\omega \\ -sin\omega & cos\omega \end{bmatrix} \begin{bmatrix} C_t \\ C^*_t \end{bmatrix} + \begin{bmatrix} \eta_t \\ \eta^*_t \end{bmatrix} \quad (4)$$

4.3 Seasonal components

Seasonal components considered in this paper are of the trigonometric class proposed by [1]. The formulation is essentially the same as the cycle with $\rho = 1$ and adding all the harmonics of the fundamental frequency/period. Calling s the known seasonal period (the number of observations per year), the number of harmonics in general is $[s/2] = s/2$ for even s numbers, and $[s/2] = (s-1)/2$ for uneven s numbers.

The overall seasonal component is then the sum of all the sinusoidal harmonics $S_{j,t}$ in equation (5), where $\omega_j = 2\pi j/s$ is the frequency of each harmonic, $S^*_{j,t}$ is an additional state necessary for the specification, and $\eta_{j,t}$ and $\eta^*_{j,t}$ are independent white noises with common variance σ^2_j.

$$\begin{gathered} S_t = \sum_{j=1}^{[s/2]} S_{j,t} \\ \begin{bmatrix} S_{j,t+1} \\ S^*_{j,t+1} \end{bmatrix} = \begin{bmatrix} cos\omega_j & sin\omega_j \\ -sin\omega_j & cos\omega_j \end{bmatrix} \begin{bmatrix} S_{j,t} \\ S^*_{j,t} \end{bmatrix} + \begin{bmatrix} \eta_{j,t} \\ \eta^*_{j,t} \end{bmatrix} \end{gathered} \quad (5)$$

4.4 Irregular components

The irregular component in **UComp** is usually considered as a residual component obtained after the extraction of the rest of components. Very often, it is just serially independent white noise with constant variance σ^2_I. But sometimes it exhibits some remaining autocorrelation. In such cases, coloured irregular components may be considered in the form of $ARMA(p, q)$

$$I_t = \frac{(1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q)}{(1 + \phi_1 B + \phi_2 B^2 + \cdots + \phi_q B^p)} \eta_{I,t}$$

where $\eta_{I,t}$ is a Gaussian white noise with constant variance σ^2_I; B is the back-shift operator such that $B^l z_t = z_{t-l}$; and ϕ_i $(i = 1, 2, \ldots, p)$ and θ_j $(j = 1, 2, \ldots, q)$ are unknown parameters that ought to be estimated from the data. The roots of both numerator and denominator polynomials should be outside the unit circle to ensure stationarity and invertibility of the $ARMA$ process.

4.5 Input-output relations

The UC model may include relations with exogenous variables naturally. But this should be done with care, since identification problems may appear especially in cases where the inputs themselves are affected by trend or seasonality. Such problems usually do not appear when the inputs are stationary (i.e., they do not mingle with the trend component) and non-seasonal (i.e., there is no confusion with the seasonal component). Typically, deterministic variables, such as calendar variables, moving festivals, or general intervention variables to deal with outlying observations are ideal candidates to consider.

4.6 Overall model

Once any particular combination of the above components are chosen, the overall State Space system is composed of the block concatenation of the individual subsystems.

Consider the example shown in equation (6), that is composed of a trend, seasonal and irregular components. The equations in matrix form on top are the so called 'state equations' that establish the dynamic mechanism of the unobserved state vector (α_t), by relating it in two consecutive time stamps. The equation at the bottom is the 'observation equation' and is just a selection of state elements to replicate the model in equation (1).

$$
\begin{array}{c}
\alpha_{t+1} = \left[\begin{array}{c} T_{t+1} \\ \hline S_{1,t+1} \\ S^*_{1,t+1} \\ S_{2,t+1} \\ S^*_{2,t+1} \end{array} \right] = \left[\begin{array}{c|cccc} 1 & 0 & 0 & 0 & 0 \\ \hline 0 & cos\omega_1 & sin\omega_1 & 0 & 0 \\ 0 & -sin\omega_1 & cos\omega_1 & 0 & 0 \\ 0 & 0 & 0 & cos\omega_2 & sin\omega_2 \\ 0 & 0 & 0 & -sin\omega_2 & cos\omega_2 \end{array} \right] \left[\begin{array}{c} T_t \\ \hline S_{1,t} \\ S^*_{1,t} \\ S_{2,t} \\ S^*_{2,t} \end{array} \right] + \left[\begin{array}{c} \eta_{T,t} \\ \hline \eta_{1,t} \\ \eta^*_{1,t} \\ \eta_{2,t} \\ \eta^*_{2,t} \end{array} \right] \\
z_t = \left[\begin{array}{c|cccc} 1 & 1 & 0 & 1 & 0 \end{array} \right] \alpha_t + I_t
\end{array}
\tag{6}
$$

The trend is the first element of the state vector (T_t) and is affected by a Gaussian white noise ($\eta_{T,t}$) with constant variance σ^2_T dynamically as a random walk (RW). The rest of the elements in the state vector define the seasonal component, that is actually the sum of two terms ($S_{1,t} + S_{2,t}$, see how the observation equation collects these two elements from the state vector in one single component) and their definition involves two frequencies, ω_1 and ω_2 that are assumed to play the role of the frequency associated to the fundamental seasonal periodicity and one of its harmonics. The seasonal components involves four Gaussian noises ($\eta_{1,t}$, $\eta^*_{1,t}$, $\eta_{2,t}$, $\eta^*_{2,t}$) that in usual formulations are mutually independent with common variance σ^2_S. The final element is the irregular component (I_t) that is simply white noise with constant variance σ^2_I.

Given the previous system the main problem is to obtain optimal estimates of the state vector and its covariance matrix, conditioned to the particular model and to all available information. This is usually achieved by well-known recursive algorithms such as the Kalman filter, Fixed Interval and Disturbance Smoothers. Before running these algorithms, the system matrices must be known and therefore the estimation of the unknown parameters must be carried out in some way, usually by Maximum Likelihood.

There are many issues related to the identification and estimation of this class of models covered in many excellent references, such as [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14].

5 Function reference

There are two sets of functions included in **UComp**. On the one hand, functions that set up **UComp** objects from scratch by using a long list of options that control the way the toolbox is going to work. These functions are: `UCsetup`, `UCestim` and `UC`.

On the other hand, the rest of the functions work directly on the objects and allow to get different outputs of interest. The syntax of all these additional function is very simple, since they only require as an input an **UComp** object.

Table 1 summarises the functions available in **UComp**.

`UC`	Overall function that runs all the rest
`UCsetup`	Creates an UComp object and sets all input options controlling how the rest of functions work
`UCmodel`	Runs `UCsetup` and `UCestim`
`UCestim`	Identifies UC model, estimates it by Maximum Likelihood and computes forecasts
`UCvalidate`	Validates UC model estimated
`UCfilter`	Optimal Kalman filtering of UC models
`UCsmooth`	Optimal Fixed Interval Smoother
`UCdisturb`	Optimal Disturbance Smoother
`UCcomponents`	Components estimation

Table 1: Main functions of **UComp**.

5.1 UC

Description

Runs all relevant functions for UC modelling in this order: `UCsetup`, `UCestim`, `UCdisturb`, `UCvalidate` and `UCcomponents`.

Usage

```
m = UC(y, frequency)
m = UC(y, frequency, ...
       'optionalvar1', optvar1, ...
       'optionalvarN', optvarN)
```

Inputs

`y`	A time series to forecast. Required input.
`frequency`	Fundamental period (number of observations per year). Required input.
`periods`	Vector of fundamental period and its harmonics. If not entered as input, it will be calculated from frequency.
`u`	A matrix of input time series. If the output wanted to be forecast, matrix u should contain future values of inputs. Default value: [ ]
`model`	The model to estimate. It is a single string indicating the type of model for each component. It allows two formats 'trend/seasonal/irregular' or 'trend/cycle/seasonal/irregular'. The possibilities available for each component are: • Trend: ? / none / rw / irw / llt / dt • Seasonal: ? / none / equal /different • Irregular: ? / none / arma(0,0) / arma(p,q) - with p and q integer positive orders • Cycles: ? / none / combination of positive or negative numbers. Positive numbers fix the period of the cycle while negative values estimate the period taking as initial condition the absolute value of period supplied. Several cycles with positive or negative values are possible and if a question mark is included, the model test for the existence of the cycles specified. Default value: '?/none/?/?'
`outlier`	Critical level of outlier tests. If NaN, it does not carry out any outlier detection (default). A negative value indicates critical minimum t test for one run of outlier detection after identification. A positive value indicates the critical minimum t test for outlier detection in any model during identification. Default value: NaN
`stepwise`	Stepwise identification procedure (true/false). Default value: false

`tTest`	Augmented Dickey Fuller test for unit roots (true/false). The number of models to search for is reduced, depending on the result of this test. Default value: false
`p0`	Initial condition for parameter estimates. Default value: NaN
`h`	Forecast horizon. If the model includes inputs h is not used, the length of u is used instead. Default value: NaN
`criterion`	Information criterion for identification (‘aic’, ‘bic’ or ‘aicc’). Default value: ‘aic’
`verbose`	Intermediate results shown about progress of estimation (true/false). Default value: false
`arma`	Check for arma models for irregular components (true/false). Default value: true
`cLlik`	Reserved input.

Output

An object of class **UComp**. It is a structure with fields including all the inputs and the fields listed below as outputs:

After running `UCestim`:

`p`	Estimated parameters
`v`	Estimated innovations (white noise correctly specified models)
`yFor`	Forecasted values of output
`yForV`	Variance of forecasted values of output
`criteria`	Value of criteria for estimated model

After running `UCdisturb`:

`yFit`	Fitted values of output
`yFitV`	Variance of fitted values of output
`a`	State estimates
`P`	Variance of state estimates
`eta`	State perturbations estimates
`eps`	Observed perturbations estimates

After running `UCvalidate`:

`table`	Estimation and validation table
`v`	Residuals

After running `UCcomponents`:

`comp`	Estimated components in table form
`compV`	Estimated components variance in table form

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UC(log(airpas), 12);
m = UC(log(airpas), 12, 'model', 'llt/equal/arma(0,0)')
```

See also UCsetup, UCmodel, UCestim, UCvalidate, UCfilter, UCsmooth, UCdisturb, UCcomponents

5.2 UCsetup

Description

Sets up UC general univariate models with a number of control variables that govern the way the rest of functions will work.

Usage

```
m = UCsetup(y, frequency)
m = UCsetup(y, frequency, ...
            'optionalvar1', optvar1, ...
            'optionalvarN',optvarN)
```

Inputs

Same as `UC`.

Outputs

Same as `UC`.

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCsetup(log(airpas), 12);
m = UCsetup(log(airpas), 12, 'model', 'llt/equal/arma(0,0)');
m = UCsetup(log(airpas), 12, 'outlier', 4);
```

See also UC, UCmodel, UCestim, UCvalidate, UCfilter, UCsmooth, UCdisturb, UCcomponents

5.3 UCmodel

Description

Function for modelling and forecasting univariate time series with UC models. It sets up the model with a number of control variables that govern the way the rest of functions in the package will work. It also estimates the model parameters by Maximum Likelihood and forecasts the data.

Usage

```
m = UCmodel(y, frequency)
m = UCmodel(y, frequency, ...
            'optionalvar1', optvar1, ...
            'optionalvarN', optvarN)
```

Inputs

Same as `UC`.

Output

An object of class **UComp**. It is a structure with fields including all the inputs and the fields listed below as outputs:

`p`	Estimated parameters
`v`	Estimated innovations (white noise correctly specified models)
`yFor`	Forecasted values of output
`yForV`	Variance of forecasted values of output
`criteria`	Value of criteria for estimated model

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCmodel(log(airpas), 12);
m = UCmodel(log(airpas), 12, 'model', 'llt/equal/arma(0,0)');
```

See also UC, UCsetup, UCestim, UCvalidate, UCfilter, UCsmooth, UCdisturb, UCcomponents

5.4 UCestim

Description

Estimates and forecasts a time series using UC models

Usage

```
sys = UCestim(sys)
```

Input

`sys`	Structure of type **UComp** created with `UCsetup` or `UCmodel`

Output

The same input structure with the appropriate fields filled in, in particular:

`p`	Estimated parameters
`v`	Estimated innovations (white noise correctly specified models)
`yFor`	Forecasted values of output
`yForV`	Variance of forecasted values of output
`criteria`	Value of criteria for estimated model

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCsetup(log(airpas), 12);
m = UCestim(m);
```

See also UC, UCsetup, UCmodel, UCvalidate, UCfilter, UCsmooth, UCdisturb, UCcomponents

5.5 UCvalidate

Description

Shows a table of estimation and diagnosis results for UC models

Usage

```
sys = UCvalidate(sys)
```

Input

`sys`	Structure of type **UComp** created with `UCmodel`

Output

The same input structure with the appropiate fields filled in, in particular:

`table`	Estimation and validation table
`v`	Estimated innovations

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCmodel(log(airpas), 12);
m = UCvalidate(m);
```

See also UC, UCsetup, UCmodel, UCestim, UCfilter, UCsmooth, UCdisturb, UC-components

5.6 UCfilter

Description

Runs the Kalman filter for UC models

Usage

```
sys = UCfilter(sys)
```

Input

sys	Structure of type **UComp** created with `UCmodel`

Output

The same input structure with the appropiate fields filled in, in particular:

`yFit`	Fitted values of output
`yFitV`	Variance of fitted values of output
`a`	State estimates
`P`	Variance of state estimates

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCmodel(log(airpas), 12);
m = UCfilter(m);
```

See also UC, UCsetup, UCmodel, UCestim, UCvalidate, UCsmooth, UCdisturb, UCcomponents

5.7 UCsmooth

Description

Runs the Fixed Interval Smoother for UC models

Usage

```
sys = UCsmooth(sys)
```

Input

`sys`	Structure of type **UComp** created with `UCmodel`

Output

The same input structure with the appropiate fields filled in, in particular:

`yFit`	Fitted values of output
`yFitV`	Variance of fitted values of output
`a`	State estimates
`P`	Variance of state estimates

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCmodel(log(airpas), 12);
m = UCsmooth(m);
```

See also UC, UCsetup, UCmodel, UCestim, UCvalidate, UCfilter, UCdisturb, UCcomponents

5.8 UCdisturb

Description

Runs the Disturbance Smoother for UC models

Usage

```
sys = UCdisturb(sys)
```

Input

`sys`	Structure of type **UComp** created with `UCmodel`

Output

The same input structure with the appropiate fields filled in, in particular:

`yFit`	Fitted values of output
`yFitV`	Variance of fitted values of output
`a`	State estimates
`P`	Variance of state estimates
`eta`	State perturbations estimates
`eps`	Observed perturbations estimates

Authors

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCmodel(log(airpas), 12);
m = UCdisturb(m);
```

See also UC, UCsetup, UCmodel, UCestim, UCvalidate, UCfilter, UCsmooth, UCcomponents

5.9 UCcomponents

Description

Estimates components of UC models

Usage

```
sys = UCcomponents(sys)
```

Input

`sys`	Structure of type **UComp** created with `UCmodel`

Output

The same input structure with the appropiate fields filled in, in particular:

`comp`	Estimated components in table form
`compV`	Estimated components variance in table form

Author

Nerea Urbina & Diego J. Pedregal

Examples

```
load data/airpas
m = UCmodel(log(airpas), 12);
m = UCcomponents(m);
```

See also UC, UCsetup, UCmodel, UCestim, UCvalidate, UCfilter, UCsmooth, UCdisturb,

6 Examples

Some simple examples are shown here to illustrate **UComp** working in practice. Mind that the examples are kept to a minimum simplicity to enhance the easiness of use of the toolbox. Much more complicated examples involving many more parameters coming from cycles, outliers, exogenous inputs, etc. are possible.

All examples run on the well-known US airpassengers data from [15], included in file `data/airpas.mat` and shown in Figure 1. The data consists of 144 monthly observations taken between 1949 and 1961.

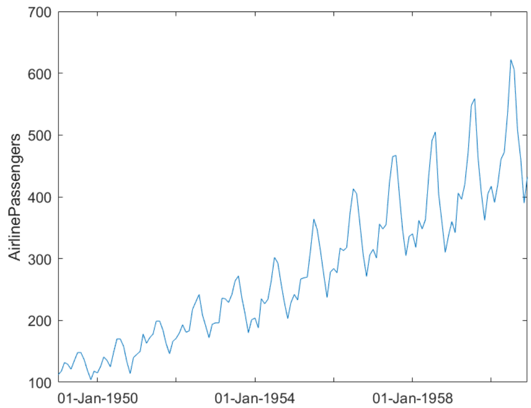

Figure 1: Airline Passengers time series

The following listing shows two ways to set up the object on the logs of this time series. There are only two compulsory inputs to the functions, namely the time series itself and the frequency of the data (actually the number of observations per year, 12 in this particular case, since the data is monthly). A full list of inputs and outputs are listed in the reference chapter 5.

```
% A call with compulsory inputs
m1 = UCsetup(log(airpas), 12);
% Another call with verbose output
m2 = UC(log(airpas),12,'verbose',true);
```

The `UCsetup` just creates an **UComp** object and sets up all the options for the future way the toolbox is working. `UCmodel` runs `UCsetup` internally and `UCestim`, i.e., it creates the object and estimates and forecasts the time series. Since no particular model is supplied, the full identification algorithm is run on the data, selecting the most appropriate model according to the Akaike Information Criterion (default option). The truncated output is shown in the following listing.

```
------------------------------------------------------------
Identification started WITHOUT outlier detection
------------------------------------------------------------
       Model                      AIC      BIC     AICc
------------------------------------------------------------
    none/none/none/arma(0,0):   1.2554   1.2760   1.2554
        none/none/equal/none:   3.3487   3.5756   3.3626
   none/none/equal/arma(0,0):   1.7350   1.9825   1.7489
    none/none/different/none:   0.8451   1.1545   0.8659
                               ...
 dt/none/different/arma(0,0):  -2.8969  -2.4844  -2.8552
llt/none/different/arma(1,0):  -2.8915  -2.4790  -2.8498
------------------------------------------------------------
Identification time:    1.61030 seconds
------------------------------------------------------------
```

The input parameters to `UCsetup`, `UCmodel` or `UC` undergo an input check of data type, meaning that if a wrong format is entered, the command is not executed. The following listing shows a wrong example.

```
%Example of wrong format input
m3 = UCsetup(log(airpas), 12, 'model', "llt/equal/arma(0,0)");
```

The output is:

```
Error using UCsetup (line 114)
The value of 'model' is invalid. It must satisfy the function: ischar
```

The model selected depends on the information criterion. The following listing shows that the model selected by the Akaike Information Criterion (AIC) is different to the one selected by the Bayesian or the corrected AIC (BIC and AICc, respectively). The best model according to AIC is `llt/different/arma(0,0)`, while the rest of more parsimonious criteria select `llt/equal/arma(0,0)`.

```
mAIC  = UC(log(airpas), 12, 'criterion', 'aic');
mBIC  = UC(log(airpas), 12, 'criterion', 'bic');
mAICc = UC(log(airpas), 12, 'criterion', 'aicc');
```

`UCvalidate` function shows the parameter estimates and some diagnositc checking on the innovations, that should be Gaussian white noise. The table itself is returned in field `table` of the output. For example, for the model identified with the AIC criterion

```
mAIC = UCvalidate(mAIC);
```

The output is

```
-------------------------------------------------------------
Concentrated Maximum-Likelihood
Model: llt/none/different/arma(0,0)
Periods:  12.0 /  6.0 /  4.0 /  3.0 /  2.4
Q-Newton: Function convergence
(*)  concentrated out parameter
(**) constrained parameters during estimation
-------------------------------------------------------------
                  Param         |T|    P-value     |Grad|
-------------------------------------------------------------
Level:        2.34e-004         nan        nan  2.52e-005
Slope:           0.0000**       nan        nan        nan
Seas(12.0):   1.10e-005         nan        nan  2.53e-005
Seas(6.0):    5.18e-006         nan        nan  4.27e-006
Seas(4.0):       0.0000**       nan        nan        nan
Seas(3.0):    2.19e-006         nan        nan  1.18e-005
Seas(2.4):    1.22e-006         nan        nan  1.32e-005
Irregular:    3.48e-004*        nan        nan        nan
-------------------------------------------------------------
     AIC:   -2.9035     BIC:    -2.5116   AICc:   -2.8618
        Log-Likelihood:   228.0492
-------------------------------------------------------------
Summary statistics:
-------------------------------------------------------------
Missing data points:
Q( 1):       0.0189     Q( 4):       1.3358
Q( 8):       2.9641     Q(12):       8.1584
Bera-Jarque: 5.0290   P-value:       0.0809
H(  46):     0.5498   P-value:       0.0452
-------------------------------------------------------------
```

This model shows some pecularities: i) it does not include the harmonic corresponding to the Nyquist frequency; ii) the trend is actually a RW with drift because the variance of the second estate equation is constrained to zero during estimation; iii) variances for all harmonics are different and the one corresponding to period 4 is actually constrained to zero in the estimation process.

Once the model is considered adequate it can be used for testing the forecasts, estimate the components, detrend the data, signal extraction, etc. Figure 2 shows both actual and forecasted data, stored in the `yFor` field of the output object.

`UCcomponents` estimates the unobserved components of the model, usually the trend, cycles, seasonal, irregular (shown in Figure 3). In case exogenous variables or outliers are considered, they also appear as components. All the components are stored in the fields `mAIC.comp` and their variances in field `mAIC.compV`. The call to this function is

```
mAIC = UCcomponents(mAIC);
```

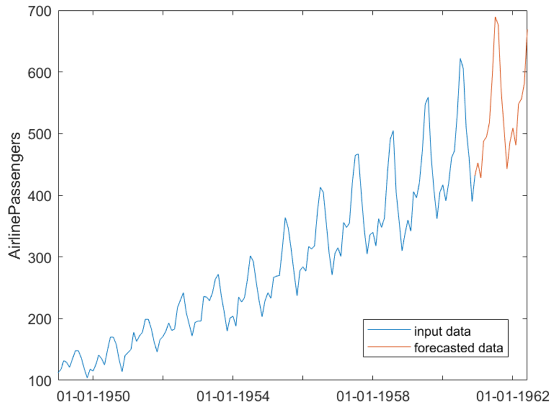

Figure 2: Airline Passengers time series with forecasted data

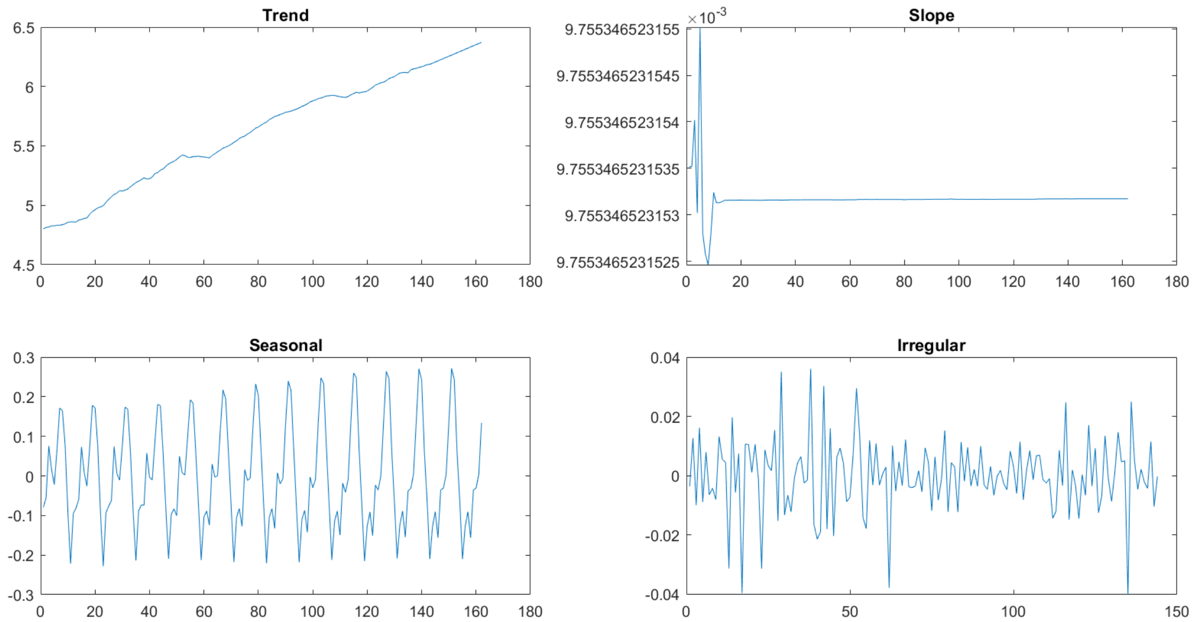

Figure 3: Components of Airline Passengers

UComp also deals with exogenous inputs as regression terms. The following listing shows how to estimate with artificial inputs added. In this case the code is simplified by running `UC` directly that produces all the outputs in one run.

```
u = zeros(3, 144);
u(1, 100:120) = 1;
u(2, 50) = 1;
u(3, 30) = 1;
m4 = UC(log(airpas), 12, ‘u’, u, ‘verbose’, true);
```

Outliers may be automatically detected by turning on the qualifier `'outlier'` in the call to `UC`. Such qualifier is assigned a positive value indicating the minimum value for the corresponding dummy variable t-test to be considered an outlier. A value of 4 is recommended, though lower values may be used with care, because the number of outliers increases as this constant decreases.

```
m5 = UC(log(airpas), 12, 'verbose', true, 'outlier', 4);
```

The output is similar to the previous one, but with a heading indicating that outliers are detected for each model. The output is

```
-------------------------------------------------------------
Identification started WITH outlier detection
-------------------------------------------------------------
         Model                        AIC      BIC     AICc
-------------------------------------------------------------
    none/none/none/arma(0,0):   1.2554   1.2760   1.2554
        none/none/equal/none:   3.3487   3.5756   3.3626
   none/none/equal/arma(0,0):   1.7350   1.9825   1.7489
                               ...
  dt/none/different/arma(0,0):  -2.8969  -2.4844  -2.8552
llt/none/different/arma(1,0):  -2.8915  -2.4790  -2.8498
-------------------------------------------------------------
Identification time:    5.54656 seconds
-------------------------------------------------------------
```

In all the previous examples the standard identification algorithm has been run, meaning that all possible have been estimated and the best selected with a particular criterion value. However, execution time can be reduced in situation where computation times are important, by selecting the stepwise or stepwise with unit roots algorithms.

```
m6 = UC(log(airpas), 12, 'verbose', true, 'outlier', 4, ...
        'stepwise', true);
```

The output is

```
-------------------------------------------------------------
Identification started WITH outlier detection
-------------------------------------------------------------
         Model                       AIC      BIC     AICc
-------------------------------------------------------------
        none/none/equal/none:   3.3487   3.5756   3.3626
   none/none/equal/arma(0,0):   1.7350   1.9825   1.7489
                                ...
  dt/none/different/arma(0,0):  -2.8969  -2.4844  -2.8552
llt/none/different/arma(1,0):  -2.8915  -2.4790  -2.8498
-------------------------------------------------------------
Identification time:    3.77010 seconds
-------------------------------------------------------------
```

No outliers are detected and therefore models m5, m6 and mAIC are the same. Two artificial outliers at observations 40 and 100 are added to test whether **UComp** is able to detect them, see Figure 4 and the following listing. The components are shown in Figure 5, where the two outliers can be easily seen.

```
yMod = y;
yMod(40) = 400;
yMod(100) = 600;
m7 = UC(log(airpasMod), 12, 'verbose', true, 'outlier', 4);
```

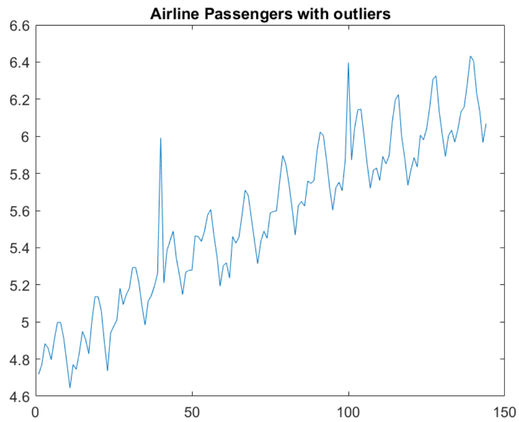

Figure 4: Airline Passengers with outliers

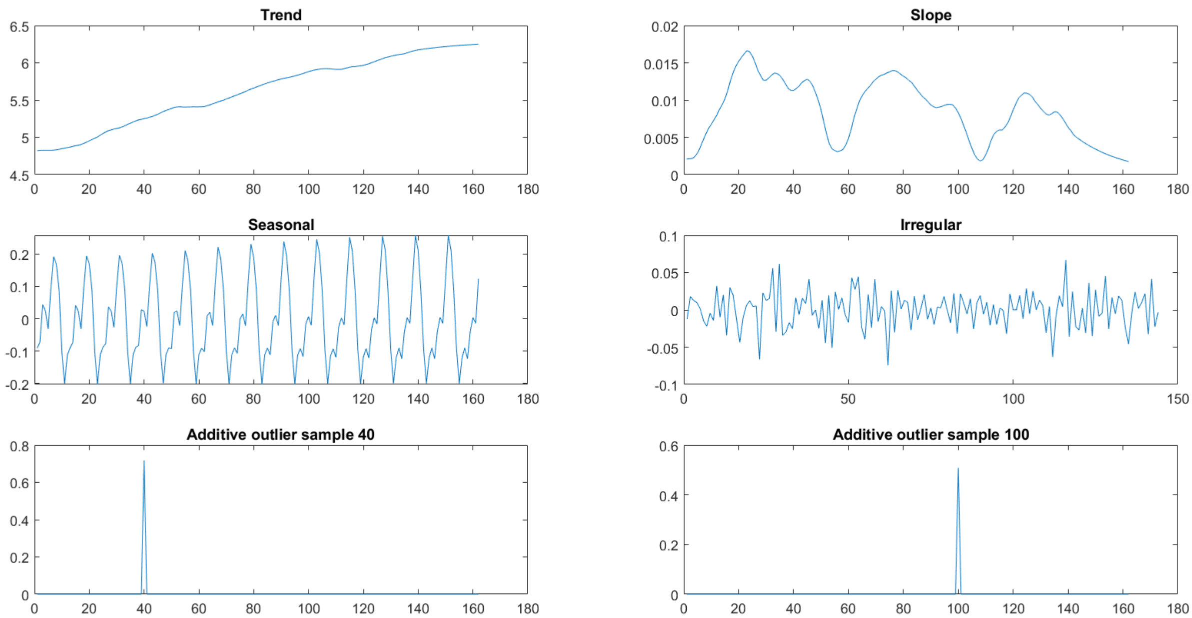

Figure 5: Components of Airline Passengers with outliers

Cycles may be introduced as an additional component to the model (as the second component). If a question mark is included the cycle existence is tested as the rest of components. Positive or negative numbers may be added using it as a fixed period (if positive) or as an initial value to start the search (if negative). For example, the following are valid model specifications: '?/?/?/?' (cycle is tested with initial period set by the toolbox), '?/24/?/?' (all models will include a cycle of 24 observations per cycle), '?/-24/?/?' (cycle will include a cycle with an estimated period starting on 24 observations per cycle), '?/24?/?/?' (a 24 period cycle is tested among the rest of components), '?/-24?/?/?' (a cycle is tested with unknown cycle starting on 24 period), '?/-48+24/?/?' (two cycles are present in all models, one with a period of 24 observations and another one with unknown period estimated by the toolbox starting on 48 observations per cycle). Cycles must be specified in the input `model`, as shown in the followint example.

```
m8 = UC(log(airpas),12, 'u', u, 'verbose', true, 'outlier', 4, ...
        'model', 'llt/-48?/eq/?');
```

References

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

[2] P de Jong. The diffuse kalman filter. *Annals of Statistics*, 19:1073–83, 1991.

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

[4] D J Pedregal and P C Young. Statistical approaches to modeling and forecasting time series. In *A Companion to Economic Forecasting*, pages 69–104. Blackwell Publishing Ltd, 2002.

[5] 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.

[6] T. Proietti. Structural time series models for business cycle analysis. In Mills T. C. and Patterson K., editors, *Handbook of Econometrics*, pages 385–433. Palgrave Macmillan, London, 2009.

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

[8] T. Proietti and A. Luati. Maximum likelihood estimation of time series models: the kalman filter and beyond. In *Handbook of research methods and applications in empirical macroeconomics*, pages 334–362. Edward Elgar Publishing Ltd., 2013.

[9] M Pelagatti. *Time Series Modelling with Unobserved Components*. Chapman-Hall / CRC, 2015.

[10] J Casals, A Garcia-Hiernaux, M Jerez, S Sotoca, and A Trindade. *State-Space Methods for Time Series Analysis: Theory, Applications and Software*. Chapman-Hall / CRC Press, 2016.

[11] T Proietti and E. Hillebrand. Seasonal changes in central england temperatures. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 180(3):769–791, 2017.

[12] M A Villegas and D J Pedregal. Sspace: A toolbox for state space modelling. *Journal of Statistical Software*, 87-5:1–26, 2018.

[13] Marco A. Villegas and Diego J. Pedregal. Automatic selection of unobserved components models for supply chain forecasting. *International Journal of Forecasting*, 35(1):157 – 169, 2019. Special Section: Supply Chain Forecasting.

[14] D J Pedregal. State space modeling for practitioners. *Foresight*, 54:21–25, 2019.

[15] 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.