Time Series Modelling with MATLAB: the SSpace toolbox*

Diego J Pedregal[1][0000−0003−4958−0969], Marco A Villegas[1][0000−0002−8491−3231], Diego A Villegas[1], and Juan R Trapero[2][0000−0002−5879−3133]

[1] Industrial Engineering Politecnic, University of Castilla-La Mancha, 13071 Ciudad Real, Spain

[2] Faculty of Chemical Sciences and Technologies, University of Castilla-La Mancha, 13071 Ciudad Real

Abstract. SSpace is a MATLAB toolbox for State Space modelling that provides the user with tools for linear Gaussian, non-linear and non-Gaussian systems with the most advanced and up-to-date features available in any State Space framework. Great flexibility is achieved because each model is coded on a standard MATLAB function, thence having absolute control on particular parameterizations, parameter constraints, time variation of parameters or variances, arbitrary non-linear relations with inputs, time aggregation, nested models, system concatenation, etc. The toolbox may be used by specifying State Space systems from scratch or by using ready-to-use templates for standard methods (like VARMAX, exponential smoothing, unobserved components, dynamic linear regression, etc.). The toolbox is freely available via a public code repository with full documentation and help system. This chapter demonstrates the toolbox's potential with several examples.

Keywords: MATLAB · State Space systems · Kalman filter · smoother algorithm · Maximum Likelihood.

1 Introduction

SSpace is a MATLAB toolbox that implements linear, non-linear and non-Gaussian State Space (SS) systems in a very flexible and powerful way. It is mainly based on the work of Peter C. Young and collaborators [14, 15] along many years seasoned with many other elements, mainly found in the books of Andrew C. Harvey, James Durbin and Siem J. Koopman [5, 7]. Though SS systems may be considered a 'classical' tool nowadays (especially in engineering and

* This work is published Valenzuela O., Rojas F., Pomares H., Rojas I. (eds.) Theory and Applications of Time Series Analysis. ITISE 2018. Contributions to Statistics. Springer, Cham. This work was supported by the European Regional Development Fund and Spanish Government (MINECO/FEDER, UE) under the project with reference DPI2015-64133-R and by the Vicerrectorado de Investigación y Política Científica from UCLM by DOCM 31/07/2014 [2014/10340].

economics), the approach is still remarkably alive as an area of active research, judging by the amount of research articles and books on this topic.

There are also numerous packages available in the marketplace, some of them available for free, and many others available commercially. There are already several toolboxes written in MATLAB including some supplied with the core program (like Signal Processing, Control, etc.), but others exist, such as CAPTAIN [14], SSM [12], SSMMATLAB [6] and E4 [2]. Two packages worth mentioning because of their relevance are STAMP [9] and SSfPack [10]. Some further examples are listed in volume 41, 2011 of the Journal of Statistical Software [3]. Some others are written either in R [13], RATS [4], gretl [11], etc. Among the commercial programs, the following incorporate SS routines with different degrees of complexity: Eviews , SAS , Stata , etc.

In a broad sense, SSpace provides the user with the most advanced and up-to-date features available in any State Space framework, e.g., the capability of dealing with both univariate and multivariate models, exact Kalman filter initialization, univariate treatment of multivariate time series, non-linear and non-gaussian modelling, alternative objective functions in parameter optimization (not only maximum likelihood), straightforward modelling of non-linear input-output relationships, etc.

The flexibility and easiness of use is reflected in the fact that SSpace was designed keeping in mind the final user and the usability of the library, by selecting easy-to-remember function names, and more importantly, by allowing a direct correspondence between the analytical expression of models and the corresponding definition in MATLAB code. In addition, users are also provided with a set of model-templates for approaching many standard models with maximum simplicity. A full help system and documentation for each function is included in both HTML and MATLAB format, complemented with eight step-by-step demos to demonstrate the use of the toolbox with standard well-known examples and others much less standard.

A final advantage of SSpace is that it is freely available via Internet at https://bitbucket.org/predilab/sspace-matlab/, where potential users are encouraged to push their own contributions and suggestions.

2 General State Space framework

SSpace supports multivariate linear and non-linear Gaussian models, and univariate non-Gaussian models. The linear Gaussian version is shown in Equation (1).

$$\begin{aligned} \text{State Equations:} \quad \alpha_{t+1} &= T_t\alpha_t + \Gamma_t + R_t\eta_t \\ \text{Observation Equations:} \quad y_t &= Z_t\alpha_t + D_t + C_t\epsilon_t \end{aligned} \tag{1}$$

In these equations, α_t is the state vector of length n; y_t are the $m \times 1$ vector of output data; $\eta_t \sim N(0, Q_t)$ and $\epsilon_t \sim N(0, H_t)$ are the state and observational vectors of Gaussian noises, with dimensions $r \times 1$ and $h \times 1$, respectively; both

noises are allowed to be correlated by a system matrix $S_t = Cov(\eta_t, \epsilon_t)$ of dimension $r \times h$; Γ_t and D_t are two matrices included to deal with input-output models in a flexible way. The remaining elements in (1) are the rest of system matrices with appropriate dimensions. The system is completed by making assumptions about the stochastic properties of the initial state vector, i.e., $\alpha_1 \sim N(a_1, P_1)$, where a_1 and P_1 are its mean and covariance matrix, respectively.

The non-linear models in SSpace are shown in Equation (2).

$$\begin{aligned} \alpha_{t+1} &= T_t(\alpha_t) + \Gamma_t + R_t(\alpha_t)\eta_t \\ y_t &= Z_t(\alpha_t) + D_t + C_t(\alpha_t)\epsilon_t \end{aligned} \tag{2}$$

Functions $T_t(\alpha_t)$ and $Z_t(\alpha_t)$ provide non-linear transformations of the state vector into vectors of size $n \times 1$ and $m \times 1$, respectively. Matrices Q_t and H_t may also depend on the state vector, but $S_t = 0$.

Finally, the non-Gaussian SS set up is shown in Equation (3):

$$\begin{aligned} \alpha_{t+1} &= T_t\alpha_t + \Gamma_t + R_t\eta_t \\ y_t &\sim p(y_t \mid \theta_t) + D_t \\ \theta_t &= Z_t\alpha_t \end{aligned} \tag{3}$$

Here θ_t is known as the *signal*. This representation allows stochastic volatility models (i.e., $y_t = exp(\frac{1}{2}\theta_t)\epsilon_t + D_t$); exponential family models (where $p(y_t \mid \theta_t) = exp[y_t'\theta_t - b_t(\theta_t) + c_t(y_t)], -\infty < \theta_t < \infty$); and models in which the observations are generated by the relation $y_t = \theta_t + \epsilon_t, \epsilon_t \sim p(\epsilon_t)$ (with $p(\bullet)$ being a distribution of the exponential family).

Given any of the previous systems, the estimation problem consists of finding the first and second order moments (i.e., mean and covariance) of the state vector, conditional on all the data in a sample. The tools that allow this operation to be performed in linear Gaussian systems are the well-known Kalman filter, fixed interval and disturbance smoothers. These algorithms may be adapted to deal with non-linear systems by running them on a Taylor linear expansion of the original non-linear system (extended Kalman filter and smoothers). Things become rather more complicated for non-Gaussian systems, which require simulation based methods that imply running the recursive algorithms repeatedly with extra computational burden. An excellent exposition of all these filtering and smoothing techniques may be found in [5], see also SSpace documentation.

The application of the recursive algorithms requires knowledge of all the system matrices. The normal situation is that part of the system matrices are known a priori and part are unknown. The unknowns are estimated in SSpace by time domain Exact Maximum Likelihood (ML) optimization, though less common procedures are also available in SSpace, e.g., estimation by minimization of several-step ahead forecast errors.

There are plenty of issues not commented in this chapter because of space constraints. For further reading, refer to SSpace documentation and [15, 7, 14, 5].

3 SSpace overview

The feature that gives SSpace its real power is the possibility of specifying models in MATLAB coded functions. Such functions follow a fixed structure that is supplied with the toolbox in a set of templates that should be used to avoid coding bugs. The general template is a standard MATLAB function called `SampleSS` shown below. This function has an input argument `p`, that is a vector of unknown parameters and that will be estimated later on. The system matrix names are easily identifiable and the template will work properly as long as the user does not remove anything from it, and just adds meaningful MATLAB code.

```
function model = SampleSS(p)
  model.T = []; model.Gam = []; model.R = [];
  model.Z = []; model.D   = []; model.C = [];
  model.Q = []; model.H   = []; model.S = [];
```

Take as an example the AR(1) process in equation (4) with $var(\eta_t) = \sigma_\eta^2$.

$$y_t = \phi y_{t-1} + \eta_t \tag{4}$$

A straightforward SS representation of this model consists of equation (4) playing the role of the state equation with $y_t = \alpha_t$ as the observation equation. By comparing this particular case with the general linear Gaussian case in equation (1), the system SS matrices are inferred as $T_t = \phi$, $R_t = Z_t = 1$, $C_t = 0$, $Q_t = \sigma_\eta^2$, $H_t = 0$. Γ_t and D_t do not exist because the model has no inputs. The filled-in version of `SampleSS` for this particular case is listed below, where the function is renamed as `ar1` to keep the original version of `SampleSS` intact for future use.

```
function model = ar1(p)
  model.T = p(1);     model.Gam = []; model.R = 1;
  model.Z = 1;        model.D   = []; model.C = 0;
  model.Q = 10.^p(2); model.H   = 0;  model.S = [];
```

The input `p` is in this case a vector of two elements, namely ϕ and σ_η^2. Beware that the system matrix $Q = \sigma_\eta^2$, identified as the second element of the input argument `p`, should be positive or zero. Thence, Q is defined as any positive or negative power of 10.

Once the model is fully specified, the way it is handled should be told to SSpace by means of a number of fundamental functions shown in the following listing.

```
>> sys = SSmodel('y', data, 'model', @ar1);
>> sys = SSestim(sys);      % Estimation
>> sys = SSvalidate(sys);   % Validation
>> sys = SSsmooth(sys);     % Smoothing
```

The first command builds a new SSpace object, called `sys`, consisting of a model written in `ar1.m` that will be applied to the data stored in memory in a

variable called `data`. The second command estimates the model by exact ML. The third shows the results in tabular form with diagnostic statistics to check model validity. Finally, the last command provides the smoothed estimates of states and their covariance matrices. With each command, the system object `sys` is filled in with the relevant output information that may be used later on.

One caveat is that any model implemented has to be transformed to SS form as a previous step before it may be used in SSpace. However, this limitation is readily overcome, because the toolbox is provided with a number of predefined templates for a set of common methods. Table 1 lists all the available functions to deal with an existing SSpace system and the templates included, see details in the documentation.

Table 1. Main functions and templates included in SSpace.

Main functions:	
`SSmodel`	Creates SSpace model object or adds properties to an existing one
`SSestim`	Estimation of a SSpace model
`SSvalidate`	Validation of a SSpace model
`SSfilter`	Optimal kalman filtering of SSpace model
`SSsmooth`	Optimal fixed interval smoothing of SSpace model
`SSdisturb`	Optimal disturbance smoother
`SSdemo`	Run SSpace demos 1 to 8
Templates:	
Linear and Gaussian models	
`SampleSS`:	General SS template
`SampleARIMA`:	ARIMA models with eXogenous variables
`SampleVARMAX`:	VARMAX models
`SampleBSM`:	Basic Structural Model
`SampleDHR`:	Dynamic Harmonic Regression
`SampleDLR`:	Dynamic Linear Regression
`SampleES`:	Exponential Smoothing with eXogenous variables
Non-Gaussian models	
`SampleNONGAUSS`:	General non-Gaussian models
`SampleEXP`:	Non-Gaussian exponential family models
`SampleSV`:	Sochastic volatility models
Non-linear models	
`SampleNL`:	General non-linear models
Other templates	
`SampleAGG`:	Models with time aggregation
`SampleCAT`:	Concatenation of State Space systems
`SampleNEST`:	Nesting in inputs State Space systems

For example, the same AR(1) model may be implemented with the aid of the `SampleARIMA` template. The advantage of using the `SampleARIMA` template is

that the model is directly defined in terms of the ARIMA specification, instead of using its SS representation. Therefore, the user does not even need to know the SS representation of an AR model. The listing below shows the `SampleARIMA` template prepared to deal with a much more complicated ARIMA model, in which all the references to the system matrices of a SS model are replaced by alternative references to the backshift operator polynomials typical of an ARIMA model (as algebraic vectors), as is the norm in other MATLAB toolboxes.

```
function model= SampleARIMA(p)
Sigma = 10.^p(1);      % Noise variance
DIFFpoly= [1 -1]';     % Differences
ARpoly = conv([1 p(2)], [1 zeros(1, 11) p(3)])';
MApoly = [1 -p(2)]';   % AR and MA polynomials
D = p(4);              % Input variables (constant)
```

The model implemented is an ARIMA$(1,1,1)\times(1,0,0)_{12}$ with a constant (in matrix `D`) and a parameter constraint consisting of setting the AR(1) parameter as the negative of the MA(1). This is a constraint that will be ludicrous in many real situations but is introduced here solely as an example of how easy it is to implement parameter constraints in SSpace.

One last point worth mentioning is that Table 1 includes a template list for non-linear and non-Gaussian templates. It also includes other templates to carry out useful operations with time series, namely time aggregation (`SampleAGG`), concatenation of SS systems (`SampleCAT`) and nesting SS systems in inputs (`SampleNEST`).

4 Examples

4.1 Example 1: Regression

Regression may be introduced in SSpace models in many different ways, and this worked example is included here as an illustration of SSpace flexibility when implementing this sort of models (see also demo number 5 of SSpace).

Consider 300 samples from a simulation of the model in equation (5), where B stands for the backshift operator such that $B^l y_t = y_{t-l}$; a_t is a Gaussian white noise serially independent with mean zero and variance 0.25 and e_t is another Gaussian white with zero mean and variance 1. This case may be seen as a regression with three inputs, namely a constant, an AR(1) process and a cosine wave. A simulated response of equation (5) is depicted in Figure 1.

$$\begin{aligned} y_t &= 15 + 4u_t + 2cos(2\pi t/50) + e_t \\ u_t &= \tfrac{1}{(1-0.8B)} a_t \end{aligned} \tag{5}$$

Because of the simplicity of this model, one sees immediately that this regression with three inputs may be viewed as a simplified version of a SS system in which the state equation does not exist and the observation equation does not relate to the states in any way, i.e., $T_t = 0$, $R_t = Z_t = 0$, $C_t = 1$, $Q_t = 0$,

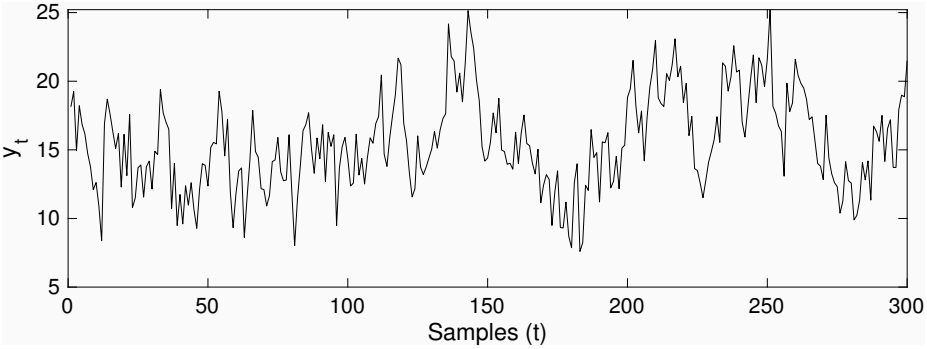

Fig. 1. A simulation sample of model in equation (5).

$H_t = 1$, Γ_t does not exist and $D_t = \beta$ is a vector of three time invariant parameters affecting each input. The user model file based on the general `SampleSS` template is shown below, where matrix `D` plays the role of β parameters of the regression.

```
function model= regression1(p)
model.T = 0; model.Gam = []; model.R = 0; model.Z = 0;
model.D = [p(1) p(2) p(3)];  model.C = 1; model.Q = 0;
model.H = 1; model.S = 0;
```

Since the observation noise variance is not included in the model, it should be estimated by concentrated ML. The execution of the estimation function would explicitly require the use of the concentrated ML function. Assuming that the inputs have been included in a $3 \times T$ MATLAB matrix called `u`, the command to estimate the model is

```
>> sys=SSmodel('y',y, 'u', u, 'model',@regression1,...
                'OBJ_FUNCTION_NAME', @llikc);
```

In this listing the output-input data are `y` and `u`, respectively; the model is in function `regression1` above; and the objective function is `llikc`, with the latter 'c' indicating the concentrated ML optimization.

Estimation results truncated to save space for the time series in Figure 1 are shown when using the `SSvalidate` function. Parameter estimates are highly significant and close to their theoretical values.

```
-------------------------------------------------------
          Param     S.E.     T-test   P-value |Gradient|
-------------------------------------------------------
 p(1)   15.0710   0.0623   241.9089    0.0000    0.000000
 p(2)    3.9847   0.0646    61.7117    0.0000    0.000000
 p(3)    1.7903   0.0835    21.4530    0.0000    0.000000
-------------------------------------------------------
```

A different way to implement this regression model consists of defining matrix D as a time varying matrix, and adding the model inputs as a second input in the user function. The next listing shows a variety of alternative definitions of matrix D.

```
function model= regression2(p, u)
...
model.D = [p(1) p(2) p(3)] * u;
...
model.D = filter(1, [1 -p(5)], u);
...
model.D = 1 ./ (exp(-p(5) * u));
...
model.D = p(5) * u;
ind = find(y > 0);
model.D(1, ind) = p(6) * u(1, ind);
```

The first case is just a redefiniton of the linear model through a time varying D matrix; the second one defines the input-output relation as a transfer function model, that may be generalized to any order; the third case is a general non-linear function; the fourth case is a linear piece wise relation, depending on whether variable y is positive or negative (in this case y should be supplied as an additional input to the user model function). It is important to note that, as all the previous specifications only affect the definition of matrix D for modelling input-output relationships, they may be introduced in any sort of model.

The call to estimate this model below is somewhat different to the previous calls, because the additional input to function **regression2** ought to be told explicitly.

```
>> sys=SSmodel('y',y, 'model',@regression2,...
                'user_inputs', u, 'OBJ', @llikc);
```

Any of the previous regression versions have an advantage that allows the use of SSpace in a completely novel and even 'mischievous' way, consisting of interpolating missing values of input variables at the same time the model parameters are estimated (note that missing values in output variables are not a problem in SS systems in contrast to missing values in input variables). This is simply solved by including all the missing values as additional parameters to estimate. For example, assuming there are two missing consecutive values in the second input at observations 200 and 201, the following line of code should be introduced prior to the definition of matrix D.

```
...
u(2, 200)= p(4); u(2, 201)= p(5);
...
```

Another way to deal with regressions is by specifying them as Dynamic Linear Regressions in which the parameters are assumed to vary over time as either Random Walks or Integrated Random Walks. In the case of three inputs the model may be written as in equation (6).

$$y_t = u_t \beta_t + \epsilon_t \tag{6}$$

This model may be fit into equation (1) easily if the state vector is just the time varying parameter and it is assumed to follow an independent random walk process ($\beta_{t+1} = \beta_t + \eta_t$). Then $\beta_t = \alpha_t$, $T_t = R_t = I$ (an identity matrix), $Z_t = u_t$, $C_t = 1$, $Q_t = Q$ (diagonal), $H_t = 1$. In this case, Z_t is a time varying system matrix.

`SampleDLR` helps the user to specify this type of models correctly. In particular, model in equation (5) is listed below.

```
function model = dlr(p, u)
D = [1 1 1];
Q = diag(10.^p(1:3));
H = 1;
```

Here, variable `D` indicates, with 1's, which of the inputs are affected by time varying parameters; `Q` is the diagonal covariance matrix noises affecting the time varying parameters in such a way that big diagonal values imply big time variations, while values close to zero imply constant parameters; finally, `H` is the variance of the observed noise, that in this specification is concentrated out from the likelihood function (i.e., `H=1`). Such specification may be used in two different ways: i) specifying zero variances in `Q` matrix is effectively telling the recursive algorithms that the model is a time constant regression, and the filtered states are their least squares recursive estimation; and ii) estimating `Q` matrix a time varying regression is estimated.

The code in the next listing produces the results below, where SSpace automatically detects that the system is a time varying regression and therefore shows the final states as the estimates of the time varying parameters. Variances of parameters (`10.^ [p(1) p(2) p(3)]`) are clearly zero, implying that the regression parameters (`[State(1) State(2) State(3)]`) are constant and close to the simulated ones. Further regression discussions are included in demo number 5 of SSpace.

```
sys= SSmodel('y', y, 'model', @dlr, ...
             'user_input', u', 'OBJ', @llikc);
sys= SSestim(sys);
sys= SSvalidate(sys);
```

```
-----------------------------------------------------
            Param      S.E.     T-test  P-val    |Grad|
-----------------------------------------------------
    p(1) -11.7780   62.4750     0.1885 0.8506 0.000000
    p(2) -14.8145   38.7453     0.3824 0.7025 0.000000
    p(3) -10.1355   45.8441     0.2211 0.8252 0.000000
State(1)  15.0710    0.0626   240.6985 0.0000         -
State(2)   3.9847    0.0649    61.4029 0.0000         -
State(3)   1.7903    0.0839    21.3454 0.0000         -
-----------------------------------------------------
```

4.2 Example 2: Time aggregation in a Basic Structural Model with trigonometric seasonality

The well-known air passengers data from [1], but with the first five years transformed to quarterly aggregated data is shown in Figure 2.

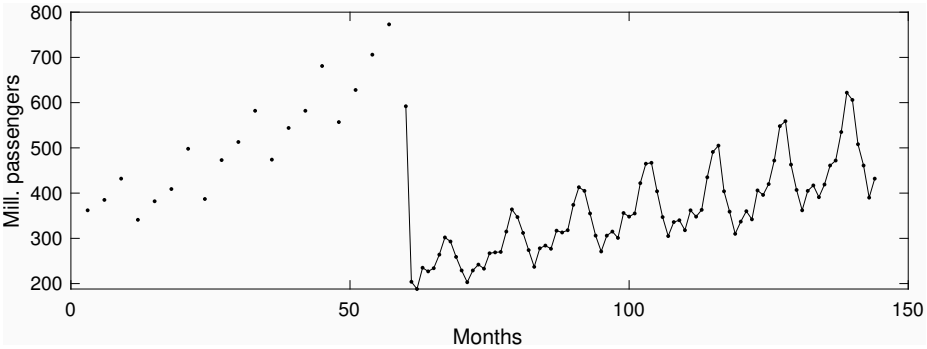

Fig. 2. Air passengers data of [1] with time aggregation.

An appropriate model for this time series is a Basic Structural Model [7] with trigonometric seasonality that may be implemented in SSpace easily with the help of `SampleBSM` template. A version of this model may be seen in the listing below (user function `airpasBsm`).

```
function model= airpasBsm(p)
% TREND MODEL (Local Linear Trend)
TT = [1 1;0 1];
ZT = [1 0];
RT = [1 0;0 1];
QT = diag(10.^(p(3:4)));
% TRIGONOMMETRIC SEASONAL MODEL with common variance
Periods = [12 6 4 3 2.4 2];
Rho     = [1 1 1 1 1 1];
Qs      = repmat(10.^(p(1)), 1, 6);
% IRREGULAR (observed noise)
H = 10.^(p(2));
```

However, in order to handle time aggregation, this function ought to be called inside another one based on `SampleAGG` template, which contains just a correct call to the user function. This extra function needs as an extra input the output data to locate exactly where the time aggregation takes place, signaled in the output data as standard MATLAB NaN (Not-a-Number) values.

```
function model= airpasBsm_agg(p, y)
model1 = airpasBsm(p);
```

Smoothed trend and seasonal components obtained with this model with and without time aggregation are shown in Figure 3. Differences are very small, meaning that the interpolation is rather appropriate.

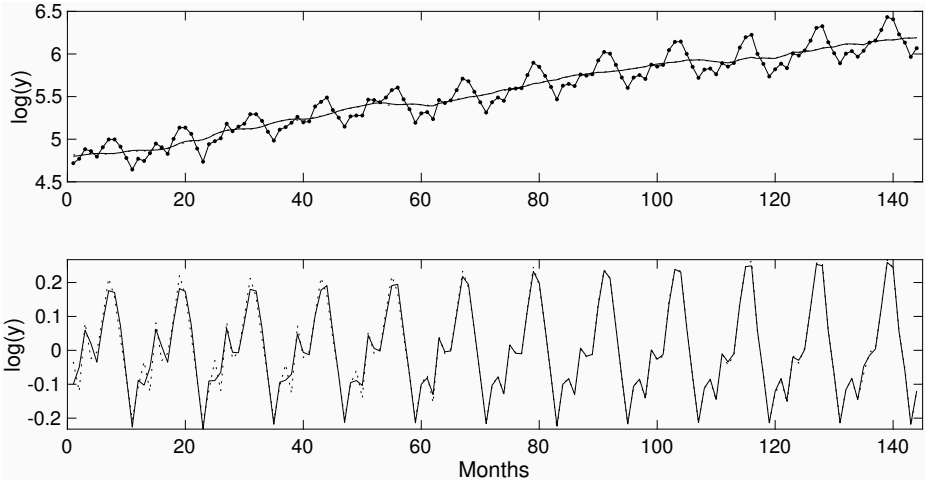

Fig. 3. Trend and log of airpassengers (top panel) and seasonal component. Estimated components with time aggregation (solid lines) and without time aggregation (dotted) are shown.

4.3 Example 3: Demand forecasting comparisons

The robustness of SSpace is evaluated in this example, in which a thorough experiment is carried out. The data consists of 517 consecutive daily sales of 261 products from a Spanish franchise, specialized in selling dishes made from natural products. Figure 4 shows some typical examples of the time series in the dataset.

Although all time series are composed of integer values, the units are much larger in top panel of Figure 4. In fact, all observations in the bottom panel are below 12. It is well known that in such cases the Gaussian approximation is not appropriate and other discrete distributions (mainly Poisson) are superior in many respects (see e.g., [5]). This suggests that the sample should be split into two groups of series, i.e., those that may be treated as *continuous* with higher values per day, and the remainder, which hereafter will be referred to as *discrete* time series. There are 166 time series in the *continuous* group, i.e., 63.6% of the total, and 95 (36.4%) in the *discrete* category.

The experimental setup of this example consists of the automatic identification and estimation of all forecasting methods for each time series using the initial 414 daily observations. 1 to 14 days ahead forecasts are then produced in a rolling experiment that advances the forecast origin one day at a time on the

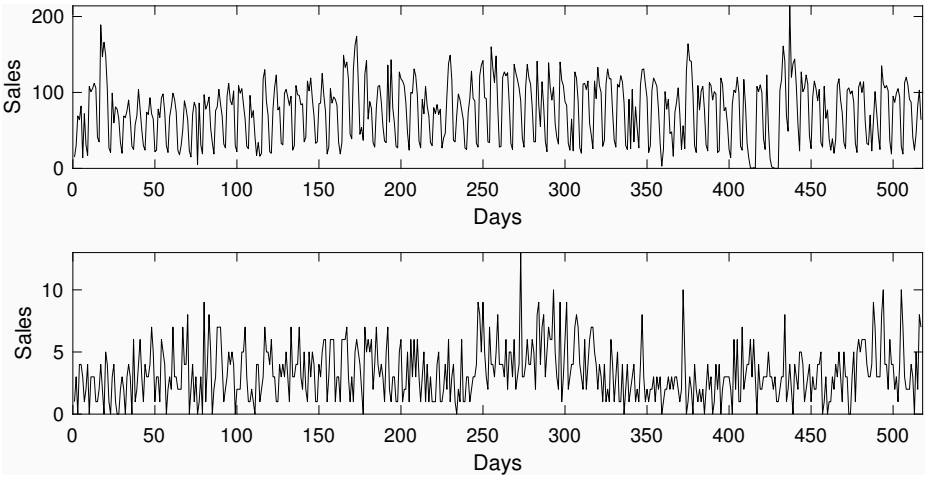

Fig. 4. Two examples of demand time series

remaining out-of-sample observations. Thus, 90 rounds of daily forecasts were done for each product.

The methods used are:

- **NAIVE**: random walk.
- **AR**: pure autorregresive models with order identified with Bayesian Information Criterion (BIC).
- **ARIMA**: identified as in [8].
- **ETS**: ExponenTial Smoothing, identified as in [8].
- **UC**: Unobserved Components based on the identification of several trend, seasonal and irregular components, based on the minimization of the BIC estimated with SSpace (using template `SampleBSM`).
- **UCp**: Poisson Unobserved Components used only for discrete time series based on the minimization of the BIC estimated with SSpace (using template `SampleEXP`).
- **MEAN**: mean combination of ARIMA, ETS, UC and UCp (in the case of discrete time series).
- **MEDIAN**: median combination of ARIMA, ETS, UC and UCp (in the case of discrete time series).

Mean of Mean Absolute Errors across all time series and all methods are shown in Table 2 at different forecast horizons (from 1 to 14), with the best method for each forecasting horizon highlighted in boldface. The table shows clearly that UC is the method with less errors in continuous time series, while the UCp is the best in the case of discrete time series. MEAN and MEDIAN are often the second best. But, what is more important from the point of view of this chapter is that the results shown require repeated runs of SSpace subroutines, that worked robustly in this long experiment.

Table 2. Mean of Mean Absolute Errors for continuous and discrete time series. Best method for each forecasting horizon is highlighted in boldface.

Continuous time series						
	1	**2**	**3**	**4**	**7**	**14**
NAIVE:	0.4695	1.0833	1.7437	2.4153	4.0597	8.3784
AR:	0.3484	0.7198	1.0967	1.4816	2.6518	5.5116
ARIMA:	0.3210	0.6634	1.0133	1.3695	2.4550	5.0598
ETS:	0.3308	0.6849	1.0455	1.4117	2.5241	5.2015
UC:	**0.3164**	**0.6539**	**0.9989**	**1.3500**	**2.4225**	**4.9901**
MEAN:	0.3209	0.6635	1.0127	1.3691	2.4563	5.0740
MEDIAN:	0.3216	0.6655	1.0163	1.3728	2.4590	5.0774
Discrete time series						
	1	**2**	**3**	**4**	**7**	**14**
NAIVE:	1.0128	2.0892	3.1674	4.2484	7.3912	14.9254
AR:	0.8304	1.6736	2.5218	3.3707	5.9269	11.9996
ARIMA:	0.8285	1.6634	2.5020	3.3425	5.8658	11.8557
ETS:	0.8338	1.6762	2.5220	3.3694	5.9136	11.9533
UC:	0.8147	1.6331	2.4542	3.2784	5.7589	11.6495
UCp:	**0.8056**	**1.6161**	**2.4284**	**3.2456**	**5.7119**	**11.5940**
MEAN:	0.8089	1.6242	2.4428	3.2642	5.7368	11.6101
MEDIAN:	0.8104	1.6259	2.4445	3.2660	5.7378	11.6103

5 Conclusions

This chapter has presented SSpace, a new MATLAB toolbox for taking full advantage of the State Space framework. SSpace is a toolbox for State Space modelling that provides the user with the possibility to model linear Gaussian, non-linear and non-Gaussian systems with the most advanced and up-to-date features available in any State Space framework, following mainly [7, 14, 5]. In addition, all system matrices are potentially time varying and may be multivariate, several estimation methods are implemented, inputs to the system may be introduced explicitly, etc.

Further advantages are that a few functions are necessary to carry out a comprehensive analysis of time series, always used with a fixed pattern and with function names carefully chosen following mnemonic rules. However, what makes SSpace flexible, powerful and transparent is that the user implements models directly by coding MATLAB functions. This feature makes extensions of models with non-standard properties possible, like time varying parameters or variances, non-linear input-output relations, etc.

The toolbox is supplied with a number of templates to carry out the time series analysis by some standard methods to avoid forcing the user to remember their respective SS form. All these advantages, in addition to robustness, are illustrated with several worked examples taken from real data.

References

1. Box, G., Jenkins, G., Reinsel, G., Ljung, G.: Time series analysis: forecasting and control. John Wiley & Sons (2015)
2. Casals, J., Garcia-Hiernaux, A., Jerez, M., Sotoca, S., Trindade, A.: State-Space Methods for Time Series Analysis: Theory, Applications and Software. Forthcoming by Chapman and Hall/CRC (2016)
3. Commandeur, J., Koopman, S., Ooms, M.: Statistical software for state space methods. Journal of Statistical Software **41**(1), 1–18 (2011). https://doi.org/10.18637/jss.v041.i01
4. Doan, T.: State space methods in rats. Journal of Statistical Software **41**(9), 1–16 (2011). https://doi.org/10.18637/jss.v041.i09
5. Durbin, J., Koopman, S.: Time series analysis by state space methods. No. 38, Oxford University Press (2012)
6. Gómez, V.: Ssmmatlab: A set of matlab programs for the statistical analysis of state space models. Journal of Statistical Software **66**(9), 1–37 (2015). https://doi.org/10.18637/jss.v066.i09
7. Harvey, A.: Forecasting, structural time series models and the Kalman filter. Cambridge university press (1989)
8. Hyndman, R.J., Khandakar, Y.: Automatic Time Series Forecasting: The Forecast Package for R. Journal of Statistical Software **3**(27), 1–22 (2008)
9. Koopman, S., Harvey, A., Doornik, J., Shephard, N.: STAMP 8.2: Structural Time Series Analyser and Modeller and Predictor. Timberlake Consultants Limited (2009)
10. Koopman, S., Shephard, N., Doornik, J.: Statistical Algorithms for Models in State Space Form: SsfPack 3.0. Timberlake Consultants Press (2008)
11. Lucchetti, R.: State space methods in gretl. Journal of Statistical Software **41**(11), 1–22 (2011). https://doi.org/10.18637/jss.v041.i11
12. Peng, J., Aston, J.: The state space models toolbox for matlab. Journal of Statistical Software **41**(6), 1–26 (2011)
13. Petris, G., Petrone, S.: State space models in r. Journal of Statistical Software **41**(4), 1–25 (2011). https://doi.org/10.18637/jss.v041.i04
14. Taylor, C., Pedregal, D., Young, P., Tych, W.: Environmental time series analysis and forecasting with the captain toolbox. Environmental Modelling & Software **22**(6), 797–814 (2007)
15. Young, P., Pedregal, D., Tych, W.: Dynamic harmonic regression. Journal of forecasting **18**(6), 369–394 (1999)