SSpace: A toolbox for State Space modelling

Marco A. Villegas  
Universidad de Castilla-La Mancha

Diego J. Pedregal  
Universidad de Castilla-La Mancha

Abstract

SSpace is a MATLAB toolbox for State Space modelling. State Space is in itself a powerful and flexible framework for dynamic system modelling, and SSpace is conceived in a way that try to enhance such flexibility to its maximum. One of the most salient features is that users implement their models by coding a MATLAB function. In this way, users have complete flexibility when specifying the systems, having absolute control on parameterisations, constraints among parameters, etc. Besides, the toolbox allows for some ways to implement either non-standard models or standard models with non-standard extensions, like heteroskedasticity, time varying parameters, arbitrary non-linear relations with inputs, transfer functions without the need of using explicitly the State Space form, etc. The toolbox may be used on the basis of scratch State Space systems, but is supplied with a number of templates for standard widespread models. A full help system and documentation is provided. The way the toolbox is built allows for extensions in many ways. In order to fuel such extensions and discussions an on-line forum has been launched.

Keywords: state space models, unobserved components, ARIMA, exponential smoothing, Kalman filter, MATLAB.

1. Introduction

SSpace is a MATLAB toolbox (The MathWorks Inc 2017) that provides a number of routines designed for a general analysis of State Space systems. It combines both flexibility and simplicity, and at the same time it enhances the power and versatility of State Space modelling in a friendly environment. The toolbox possess very distinct properties to other State Space pieces of software, but at the same time takes advantage of methods and algorithms from other sources, mainly Taylor, Pedregal, Young, and Tych (2007) and Durbin and Koopman (2012). The combination of all these factors give SSpace a particular flavour.

Popularity of high level programming languages like MATLAB has brought the availability of many free packages with an incredibly wide range of applications in many research areas. State Space (SS) routines are not an exception, and therefore good packages are so widespread nowadays, either as open source or paid versions, that it is impossible to quote even a portion
of them. Without any intention of being exhaustive, we offer here a list of the most popular packages within the academic community. A partial review is available in volume 41, 2011 of the Journal of Statistical Software (Commandeur, Koopman, and Ooms 2011). There are several toolboxes written in MATLAB, like toolboxes supplied with the core program, but also CAPTAIN (Taylor et al. 2007), SSM (Peng and Aston 2011), SSMMATLAB (Gómez 2015) and E4 (Casals, García-Hiernaux, Jerez, Sotoca, and Trindade 2016). One piece of software widely known is SSFpack (Koopman, Shephard, and Doornik 2008). Some others are written either in R (Petris and Petrone 2011), RATS (Doan 2011), gretl (Lucchetti 2011), etc. Also, some other, menu-driven programs that incorporate SS routines with less flexible programming capabilities are STAMP (Koopman, Harvey, Doornik, and Shephard 2009), Eviews (Bossche 2011), SAS (Selukar 2011), Stata (Drukker and Gates 2011), etc. ECOTOOL is a complementary MATLAB toolbox written by the same authors for the identification and estimation of dynamical systems (Pedregal and Trapero 2012).

In a broad sense, SSspace provides the user with the most advanced and up-to-date features available in the State Space framework, sharing some of these properties with some packages mentioned above and competing with them. However, some other features are specific of this toolbox and will not be found in any of the alternatives.

Regarding statistical issues, the main features of SSspace are:

1. Full multivariate linear and non-linear Gaussian models, and univariate non-Gaussian models are implementable. In any of these possibilities non-linear, time varying or transfer function relations with inputs are possible.

2. The framework is very general in the equations formulation and in the sense that all system matrices are potentially time-varying or state-dependent.

3. Kalman filter, fixed interval smoothing and disturbance smoothing are implemented with exact, diffuse or ad-hoc initialisation. Exact initialisation in non-linear models is also possible following Koopman and Lee (2009).

4. Steady state detection of linear invariant systems.

5. Use of exact score in Maximum Likelihood estimation, when possible. Use of numerical gradient is always possible, sometimes compulsory.

6. Other estimation procedures apart from Maximum Likelihood are implemented. At the moment the toolbox offers Concentrated Likelihood and minimization of combinations of several steps ahead forecast errors. More cost functions may be added in the future.


8. A family of models not used in any of the alternative packages is included, namely the Dynamic Harmonic Regression. To the best knowledge of the authors, this is the first time that a multivariate Seemingly Unrelated Dynamic Harmonic Regression is used and implemented. Such model is an extension of the univariate Dynamic Harmonic Regression counterpart, see Young, Pedregal, and Tych (1999) and worked example 3 below.
On the operative side, SSpace is rather flexible and user friendly because:

- The toolbox is user-oriented in the sense that a big effort has been done on the developer side with the aim of simplifying usage to the final users. As a consequence, a comprehensive time series analysis may be performed with full control over models, parameters and specifications, by using just a few number of functions that follow a simple and fixed calling-standard that is easy to remember (see section of worked examples below). One example of this simplicity is that just one function (namely SSpacefilter) is used for filtering any kind of model, regardless of whether it is linear, non-gaussian or non-linear, i.e., the toolbox detects the type of model and apply the appropriate algorithms in each case automatically without any specific intervention of the user.

- The toolbox is composed of less than thirty functions with names that have been carefully selected following nemotechnic rules, so that the user may remember them or easily look for their names. There are three groups of functions:
  - Core functions (named SS*) which set up the models and perform the basic operations of filtering, smoothing, forecasting, validating, etc.
  - Template functions (Sample*) which help the user to set up the model either in terms of the crude State Space system matrices or in terms of its specific nature.
  - Other helpful functions for easy handling of models, matrices, etc. The main ones are for constraining parameters, building semi-definite covariance matrices, differencing time series, building forecast confidence bands, etc.

- Another key point is that models are directly specified by the user in a user-coded function written in plain MATLAB syntax. This approach has at least the following advantages:
  - Once the model is specified in the user-coded function, the same syntax applies to all sort of models, regardless of linearity, gaussianity, estimation method, etc. i.e., the same functions with the same syntax are used for estimation, filtering, smoothing, etc. Internally the operation of the toolbox may be rather different in each case, but such complexity does not require any intervention of the user.
  - The user-coded function is written following a particular template, that is basically a list of empty values for all the system matrices. The toolbox offers particular templates for a wide range of standard models. These templates may be extended or substituted by users. Such functions and templates may be extended in many ways. Typically, for complex models the model function has to be extended with the aid of additional inputs to the MATLAB function in order to define the model in SS form.
  - The user has full control over his models in a fairly straightforward manner, e.g., different specific parameterizations of the same model are possible, parameter constraints of any kind may be imposed, non-standard features of the models may be added (like adding heteroskedasticity, time varying parameters, non linear eXogenous relations, ...), etc.

- Parameter estimation is carried out by the standard fminunc function from the optimization toolbox in MATLAB. By editing the script optimizer.m the user may tune the optimisation settings and/or change even the optimizer itself.
As a summary, the toolbox is highly configurable and could be extended in several ways, a reason for which users are encouraged to push their own contributions to the repository https://bitbucket.org/predilab/sspace-matlab/. Usually default options for modelling are in place in such a way that a few commands with a few options would produce sensible results, but advanced users may take advantage of the possibility to configure the toolbox at their own convenience. For example, they could select different sets of initial parameter values to start estimation in order to find the global optimum, build alternative cost functions to log-likelihood for parameter estimation based on the output of the Kalman filter, try out different optimisation algorithms to search for optimal parameters, add templates for models not yet implemented or suggest different versions to the existing ones, build canned functions for standard model estimation, etc.

The rest of the paper is organized as follows: the next section shows the models implemented in analytical form; Section 3 provides a broad vision of the functions included in the toolbox and its main usage on implementing a full time series analysis through a simple example; Section 4 illustrates the usage of SSpace with several examples for different scenarios and complexity levels; and Section 5 extracts the main conclusions. Code listings and examples are considerably compressed for space reasons, but may be consulted in extended form in the SSpace demos.

2. Models implemented in SSpace

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

\[
\begin{align*}
\alpha_{t+1} &= T_t \alpha_t + \Gamma_t \eta_t, \quad \eta_t \sim N(0, Q_t) \\
y_t &= Z_t \alpha_t + D_t \epsilon_t, \quad \epsilon_t \sim N(0, H_t) \\
\alpha_1 &\sim N(a_1, P_1) \\
\end{align*}
\]

(1)

In this equations \(\alpha_t\) is the state vector of length \(n\); \(y_t\) are the \(m \times 1\) vector of output data; \(\eta_t\) and \(\epsilon_t\) are the state and observational vectors of zero mean Gaussian noises, with dimensions \(r \times 1\) and \(h \times 1\), respectively; both noises are allowed to be correlated by a system matrix \(S_t = \text{Cov}(\eta_t, \epsilon_t)\) of dimension \(r \times h\); \(\alpha_1\) is the initial state with mean \(a_1\) and covariance \(P_1\) and independent of all disturbances and observations involved. The remaining elements in (1) are the rest of system matrices with appropriate dimensions, i.e.,

- \(T_t\): \(n \times n\);
- \(\Gamma_t\): \(n \times 1\);
- \(R_t\): \(n \times r\);
- \(Z_t\): \(m \times n\);
- \(D_t\): \(m \times 1\);
- \(C_t\): \(m \times h\).

One interesting feature (see Section 4.1.2 for an example) is that the toolbox is flexible enough to allow the terms \(\Gamma_t\) and \(D_t\) be defined in such a way that \(k\) exogenous input variables appear explicitly, i.e., \(\Gamma_t = f(\gamma_t, u_t)\) and \(D_t = g(d_t, u_t)\), with \(u_t\) of dimensions \(k \times 1\). Beware that general functions \(f(\bullet)\) and \(g(\bullet)\) include as particular cases time varying linear functions \(\Gamma_t = \gamma_t u_t\) and \(D_t = d_t u_t\), with \(\gamma_t\) and \(d_t\) of dimensions \(n \times k\) and \(m \times k\), respectively.

Apart from this, the formulation in (1) is also rather general. In particular, data sets may be multivariate, all system matrices are time varying and noises in state and observation equations may be correlated. The system is actually so general that some readers would
immediately detect some redundancies and some terms that are not strictly necessary in most applications. However, we have preferred to set up the model in the most general possible way, such that any potential user acquainted with SS methodology does not have to change his particular mindset, so that the effort to translate the system into SSpace is kept to a minimum.

The non-Gaussian SS set up is shown in Equation 2:

\[
\begin{align*}
\alpha_{t+1} &= T_t \alpha_t + \Gamma_t + R_t \eta_t, \quad \eta_t \sim N(0, Q_t) \\
y_t &\sim p(y_t \mid \theta_t) + D_t, \\
\theta_t &= Z_t \alpha_t \\
&= 1, 2, \ldots, N
\end{align*}
\]

Here \( \theta_t \) is known as the *signal*. With this representation it is possible to deal with three types of models (see Durbin and Koopman, 2012):

1. **Exponential family distribution**, where \( p(y_t \mid \theta_t) = \exp\left[ y_t' \theta_t - b_t(\theta_t) + c_t(y_t) \right], -\infty < \theta_t < \infty. \)

2. **Stochastic Volatility models**, i.e., \( y_t = \exp(\frac{1}{2} \theta_t) \epsilon_t + D_t. \)

3. **Observations 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.

Finally, the non-linear models are of the type shown in Equation 3.

\[
\begin{align*}
\alpha_{t+1} &= T_t(\alpha_t) + \Gamma_t + R_t(\alpha_t) \eta_t, \quad \eta_t \sim N(0, Q_t(\alpha_t)) \\
y_t &= Z_t(\alpha_t) + D_t + C_t(\alpha_t) \epsilon_t, \quad \epsilon_t \sim N(0, H_t(\alpha_t)) \\
&= 1, 2, \ldots, N
\end{align*}
\]

Functions \( T_t(\alpha_t) \) and \( Z_t(\alpha_t) \) with first derivatives provide non-linear transformations of the state vector into vectors of size \( n \times 1 \) and \( m \times 1 \), respectively. The rest of system matrices may also depend on the state vector and \( S_t = 0. \)

Given this general framework, (extended) Kalman filtering, state and disturbance smoothing provide the basis for optimal state estimation, parameter estimation, signal extraction, forecasting, etc. For all the algorithmic issues not explicitly commented in this paper refer to Young *et al.* (1999), Taylor *et al.* (2007) and Durbin and Koopman (2012).

Table 2 shows some of the main features of most common software packages. The top block corresponds to toolboxes written in MATLAB, the rest are developed in other environments. The table highlights several facts: i) exact initialization is present in most packages, ii) non-linear and non-gaussian models are less common than expected and iii) most packages use Maximum Likelihood estimation as the only estimation method.

Moreover, there are some unique properties of SSpace, as far as the authors are concerned, like the implementation of multivariate Dynamic Harmonic Regression models, the implementation of systems by direct specification of the system matrices in a MATLAB function, system estimation via the minimization of forecast errors several steps ahead, the possibility of systems implementing arbitrary non-linear (and possibly multivariate) relations among inputs and outputs, the possibility of multiple-input-multiple-output transfer functions, and the chance to select an arbitrary minimizer algorithm to estimate the models (fminunc as a default).
Table 1: Options available in most common state space software packages. The options are exact initialization (EI), other estimation methods apart from ML (+ML), univariate treatment of multivariate models (UTMM), non linear and non gaussian models (NLNG) and availability of state disturbance algorithms (DA).

The flexibility of SSpace is so big that, being this its most powerful feature, at the same time it may be a problem for some users. Specifying a model from scratch in SSpace takes some time and a bit of familiarization with the toolbox. To solve these problems two solutions are implemented: i) when specifying the model, SSpace performs internally a full set of coherency tests to ensure that the model is correctly specified and issue error or warning messages with the specific problems that guide the user towards the solution, and ii) templates for most common models are provided so that the user do not have to bear in mind the State Space form of any of them. This is actually a part of SSpace that will grow as more templates appear in the future.

3. SSpace overview

Table 3 shows the core functions necessary to carry out a comprehensive time series analysis. From all this the most important to understand is SSmodel. It creates a structure with the user inputs and all the outputs, that will be empty at the time of creation (for a full description of inputs and outputs type help SSmodel at the MATLAB prompt). This structure will be the input to the rest of functions that have to handle the system, like estimation, filtering, etc., and it also may be the output to such functions, in a way that it is completed little by little with each additional operation. With SSmodel the user specifies the input and output data, the model to use, additional inputs to the user-coded function necessary to implement the model, either exact or diffuse or ad-hoc initialization of recursive algorithms, initial parameters for parameter estimation, fixed parameters that would be frozen in estimation, the cost function to optimize, exact or numerical score (if possible) in Maximum Likelihood estimation, etc. To sum up, it sets up the models up to the smallest detail, controlling the posterior performance of the rest of functions.

Once the model is created, SSestim performs parameter estimation by the method previously selected in SSmodel. SSvalidate produces a table with the estimation results and diagnostics.
SSfilter produce the innovations and filtered estimates of states and covariance matrices with additional output. If smoothed output is preferred, it is produced by the SSsmooth function. Disturbance errors (and smoothed output) may be computed by SSdisturb. Finally, there are eight step-by-step demos ready, that may be run with the SSdemo function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSmodel</td>
<td>Creates SSpace model object or adds properties to an existing one</td>
</tr>
<tr>
<td>SSestim</td>
<td>Estimation of a SSpace model</td>
</tr>
<tr>
<td>SSvalidate</td>
<td>Validation of a SSpace model</td>
</tr>
<tr>
<td>SSfilter</td>
<td>Optimal kalman filtering of SSpace model</td>
</tr>
<tr>
<td>SSsmooth</td>
<td>Optimal fixed interval smoothing of SSpace model</td>
</tr>
<tr>
<td>SSdisturb</td>
<td>Optimal disturbance smoother</td>
</tr>
<tr>
<td>SSdemo</td>
<td>Run SSpace demos 1 to 8</td>
</tr>
</tbody>
</table>

Table 2: Main functions of the SSpace library.

All the functions in Table 3 run on the model previously coded by the user in MATLAB language. In order to make the communication between the user and SSpace efficient and easy, a number of templates have been created and listed in Table 3. SampleSS sets up any linear and gaussian models with or without inputs and any sort of non-standard feature. The rest of linear models are self-explanatory and are intended for the creation of well-known models. There are also some templates for non-Gaussian models and for general non-linear models (SampleNENL). Additional templates help the user to build models with time aggregation, concatenate systems o nest systems in inputs. In all cases, time varying system matrices are three dimensional, being time the third dimension.

<table>
<thead>
<tr>
<th>Linear and Gaussian models</th>
</tr>
</thead>
<tbody>
<tr>
<td>SampleSS: General SS template</td>
</tr>
<tr>
<td>SampleARIMA: ARIMA models with eXogenous variables</td>
</tr>
<tr>
<td>SampleBSM: Basic Structural Model</td>
</tr>
<tr>
<td>SampleDHR: Dynamic Harmonic Regression</td>
</tr>
<tr>
<td>SampleDLR: Dynamic Linear Regression</td>
</tr>
<tr>
<td>SampleES: Exponential Smoothing with eXogenous variables</td>
</tr>
<tr>
<td>Non-Gaussian models</td>
</tr>
<tr>
<td>SampleNONGAUSS: General non-Gaussian models</td>
</tr>
<tr>
<td>SampleEXP: Non-Gaussian exponential family models</td>
</tr>
<tr>
<td>SampleSV: Sochastic volatility models</td>
</tr>
<tr>
<td>Non-linear models</td>
</tr>
<tr>
<td>SampleNL: General non-linear models</td>
</tr>
<tr>
<td>Other templates</td>
</tr>
<tr>
<td>SampleAGG: Models with time aggregation</td>
</tr>
<tr>
<td>SampleCAT: Concatenation of State Space systems</td>
</tr>
<tr>
<td>SampleNEST: Nesting in inputs State Space systems</td>
</tr>
</tbody>
</table>

Table 3: Available templates for the SSpace toolbox.

The rest of functions in Table 3 are very useful to set up models in SS form: i) confband builds
confidence bands of filtered or smoothed outputs in a comfortable way, ii) \texttt{constrain} settles constraints among parameters in the models, iii) \texttt{varmatrix} is a function useful to constrain covariance matrices to be semi positive definite in multivariate models or just positive in scalar cases, iv) \texttt{normalize} standardizes any set of time series with a time varying covariance structure, vi) \texttt{vdiff} produces differencing of vector time series, and vii) \texttt{optimizer} is an editable script that allows tuning the optimizer tolerances and even to change the optimizer itself.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{confband}</td>
<td>Forecasts confidence intervals</td>
</tr>
<tr>
<td>\texttt{constrain}</td>
<td>Free constraints of parameters</td>
</tr>
<tr>
<td>\texttt{varmatrix}</td>
<td>Semidefinite positive covariance matrices</td>
</tr>
<tr>
<td>\texttt{normalize}</td>
<td>Variable normalization (standardization)</td>
</tr>
<tr>
<td>\texttt{vdiff}</td>
<td>Differentiation of vector time series</td>
</tr>
<tr>
<td>\texttt{optimizer}</td>
<td>Optimizer options</td>
</tr>
</tbody>
</table>

Table 4: Auxiliary functions for the \texttt{SSpace} library.

The analysis with \texttt{SSpace} consists of following the next steps, that mimics closely the steps any researcher ought to follow in any SS analysis.

1. Write the model on paper or specify the model in SS form.
2. Translate model to \texttt{MATLAB} code using the templates supplied.
3. Set up model with \texttt{SSmodel}.
4. Estimate unknown parameters with \texttt{SSestim}.
5. Check appropriateness of model with \texttt{SSvalidate}.
6. Determine optimal estimates of states, their covariance matrices, innovations, etc. Any or several of \texttt{SSfilter}, \texttt{SSsmooth} or \texttt{SSdisturb} may be used.

In the next subsections, a local level (or random walk plus noise) model is used to illustrate how to implement all these steps applied to the Nile river data used in Durbin and Koopman (2012), specifically \texttt{demo1} of \texttt{SSpace}. The data consists of the flow volume of the Nile river at Aswan from 1871 to 1970. The local level model is given in Equation 4, being $B$ the back-shift operator.

$$y_t = \frac{\theta}{(1-B)} + \epsilon_t; \quad \text{VAR}(\eta_t) = Q \quad \text{VAR}(\epsilon_t) = H$$

(4)

### 3.1. Specify model (step 1)

One SS representation of (4) is (5):

\[
\begin{align*}
\text{State Equation:} & \quad \alpha_{t+1} = \alpha_t + \eta_t \quad \eta_t \sim N(0, Q) \\
\text{Observation Equation:} & \quad y_t = \alpha_t + \epsilon_t \quad \epsilon_t \sim N(0, H)
\end{align*}
\]
It may be seen immediately that the local level is one of the simplest models that can be specified in SS form. Comparing it with the general form (1) we see that for this case all system variables are scalar and time invariant, i.e., \( T_t = R_t = Z_t = C_t = 1 \), \( Q_t = Q \), \( H_t = H \) and \( \Gamma_t \) and \( D_t \) do not exist because the model has no inputs.

### 3.2. Code the model (step 2)

The best way to deal with the previous model is to edit the SampleSS template, rename it to, say, `example1`, and fill in all the matrices values accordingly. The aspect of SampleSS is shown below, with the system matrix names easily identifiable. The template is offered in this way, nothing should be removed, but anything could be added in order to define the system matrices.

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

The following is the adaptation of such template to the local level model.

```matlab
function model = example1(p)
    model.T = 1;
    model.Gam = [];
    model.R = 1;
    model.Z = 1;
    model.D = [];
    model.C = 1;
    model.Q = 10.^p(1);
    model.H = 10.^p(2);
    model.S = 0;
end
```

Beware that the input argument `p` to both functions is a vector of parameters, in this case just the scalars \( Q \) and \( H \). By default, both state and observation noises are considered independent. Furthermore, the first element in the vector `p` has been assigned to the matrix \( Q \), while the second is assigned to \( H \). Since both must be positive values, the system matrices are defined as powers of 10. An alternative and equivalent definition is `model.Q = varmatrix(p(1))`.

### 3.3. Setting up the model (step 3)

Now the user has to communicate with SSpace and build a model to use later on. This is done with the SSmodel function. In this case the MATLAB code is simply `sys = SSmodel('y',
```
nile, 'model', @example1). It is assumed that nile is a vector variable with the Nile data in and basically with this line code the user is telling that he wants to apply the local level model written in example1 to the data in the MATLAB variable nile.

This is the most important step because at this stage is where the user defines the posterior performance of the toolbox. In essence, if the validation of the model is not correct, then the user has to come back to SSmodel and change either the model, options, etc. There are many options available to set up the model, that are passed on to SSmodel using duplets (see all possibilities in the SSpace documentation).

3.4. Estimate parameters (step 4)
Having defined the model in the previous step, the rest is straightforward. In particular, the estimation is done by sys = SSestim(sys).

No additional inputs to this functions are necessary, since everything has been set up in the previous step via the SSmodel function, in particular the estimation method that will be Exact Maximum Likelihood by default. Parameters are stored in the sys output structure. If estimation converges to well defined optimum, then estimation results, with standard errors of parameters, information criteria, etc. may be shown by means of the SSvalidate function with the syntax sys = SSvalidate(sys).

3.5. Use the model (step 5)
A final step consists of estimating the filtered and/or smoothed output together with the disturbances of the model, further validation tests. These operations constitute the basis for forecasting, signal extraction, interpolation, etc. When only the filtered output is required the call should be sys = SSfilter(sys); if smoothed estimates without disturbances is preferred then the call should be sys = SSsmooth(sys); whereas the full computation and output is produced by the call sys = SSdisturb(sys).

Results in all these brief examples are stored always in sys output structure, though at any point different structures may be used. It stores parameters with covariance matrix, optimal states and covariances, fitted output values and covariances, forecasted values and covariances, innovations, disturbances estimates with covariances. Further statistical diagnostics are advisable.

4. Worked examples

The examples shown in this section are presented as illustrations of the flexibility and power of the toolbox, with no pretension of showing any scientific result or improvement over other researcher’s analysis. Code listings are truncated in order to save space. This is especially important in the case of the templates shown, for which the extended versions are also included in the software provided with an abundant help. Identification and validation issues are not treated in the examples to keep them short.

4.1. Example 1: Local level
The next listing puts together the MATLAB code shown up to now for the local level model
applied to the Nile river data, with some additions for plotting outputs.

```matlab
% Load data and set up time variable with NaN's for interpolation and forecasting
load nile
y = [nile; nan(10, 1)];
y(61:70) = nan;
t = (1 : size(y, 1))';
% Build SSpace model
sys = SSmodel('y', y, 'model', @example1);
% Estimate model by exact ML
sys = SSestim(sys);
% Model table output etc.
sys = SSvalidate(sys);
% Smoothing
sys = SSsmooth(sys);
% Plotting fitted values with 90% confidence bands
plot(t, y, 'k', t, sys.yfit, 'r.-', t, confband(sys.yfit, sys.F, 1), 'r:');
% Plotting innovations
plot(sys.v)
% Estimating and plotting disturbances
sys = SSdisturb(sys);
plot(t, sys.eta, 'k', t, sys.eps, 'r')
```

Some missing observations have been arbitrarily added in the middle of the data and at the end to show how interpolation and forecasting are automatically done. The output of the `SSvalidate` call is:

```
Linear Gaussian model: example1.
Objective function: llik
System collapsed at observation 1 of 100.
Exact gradient used.
```

```
<table>
<thead>
<tr>
<th>Param</th>
<th>S.E.</th>
<th>T-test</th>
<th>P-value</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(1)</td>
<td>3.1404</td>
<td>0.3595</td>
<td>8.7362</td>
<td>0.0000</td>
</tr>
<tr>
<td>p(2)</td>
<td>4.2084</td>
<td>0.0855</td>
<td>49.2198</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
```

```
AIC: 12.906
BIC: 12.9938
HQC: 12.9398
Log-likelihood: -571.3177
Corrected R2: 0.2669
Residual Variances: 21919.3612
```

Summary Statistics:
This table shows abundant information about convergence of estimation, significance of parameters, information criteria, and diagnostic statistics on the innovations (autocorrelation, heteroskedasticity and gaussianity). The resulting plots are shown in Figure 1.

![Figure 1: Fit, innovations and disturbances of example1.m.](image)

Exactly the same results may be obtained if the model is estimated by Concentrated Maximum Likelihood. Two slight changes have to be made to the previous code: i) one of the variances in the user model has to be concentrated out of the likelihood by setting it to 1 (say, \texttt{model.Q = 1}; in \texttt{example1.m}); ii) the call to \texttt{SSmodel} changes to \texttt{sys = SSmodel('y', y, 'model', @example1, 'OBJ_FUNCTION_NAME', @llikc)}. Beware that only the three initial characters are necessary in the string inputs arguments (i.e., \texttt{'OBJ'}, instead of \texttt{'OBJ_FUNCTION_NAME'}). See a more detailed description by running \texttt{SSdemo(1)}.
One of the challenges for this time series, is to evaluate whether the construction of the Aswan dam in 1899 (observation 29) led to a significant decline in the river flow. This may be tested in several ways by changing the user function, either by including a $D_t$ in Equation 1 directly (Case 1 below), or by using a dummy variable as input to the SS system (Cases 2 and 3). It is worthy passing through these three cases to realize the flexibility of \texttt{SSpace} when specifying models.

\textit{Case 1}

In this case the user function for concentrated maximum likelihood would be:

```matlab
function model = example2(p)
    model.T = 1;
    model.Gam = [];
    model.R = 1;
    model.Z = 1;
    model.D = [repmat(p(2), 1, 28) repmat(p(3), 1, 82)];
    model.C = 1;
    model.Q = 1;
    model.H = 10.^p(1);
    model.S = 0;
end
```

Keep in mind that matrix $D_t$ is time varying, but it is not defined as a three dimensional matrix, as it is the general convention in \texttt{SSpace}. This is an exception that affects also $\Gamma_t$ and has been considered very convenient from the user point of view, since handling three dimensional matrices in \texttt{MATLAB} is much more cumbersome than two dimensional. Nevertheless, orthodox three dimensional matrices would work exactly in the same way. The call for estimating the model by Concentrated Maximum Likelihood is

```matlab
sys = SSmodel('y', y, 'model', @example2, 'OBJ_FUNCTION_NAME', @llikc, 'p0', [-1; 1000]);
```

An interesting point of this example is that a $D_t$ matrix is used, but not input data is supplied and there is no need to specify a $\Gamma_t$ matrix. Here, initial parameters for the numerical search are added via the duplet \{'p0', [-1; 1000]\}. The rest of the code is identical to the previous example. The results are shown in Figure 2.

It is important to note that the estimation is such that there is no additional information in the series apart from the jump in the volume due to the Aswan dam. By comparison with Figure 1, it is easy to check this, since the innovations and output are essentially the same.

\textit{Case 2}

An alternative way to do the same, which is more formal from a statistical point of view, consists on defining one dummy variable as a step, taking zeros up to observation 28 and ones afterwards, i.e., $u = [\text{zeros}(28, 1); \text{ones}(82, 1)]$. In this case, the user function is:

```matlab
function model = example3(p)
    model.T = 1;
    model.Gam = [];
    model.R = 1;
end
```
model.Z = 1;
model.D = p(2);
model.C = 1;
model.Q = 1;
model.H = 10.^p(1);
model.S = 0;

The difference with the previous option is that matrix model.D is just the second element of the parameter vector \( p \), i.e., a coefficient that will multiply the input dummy variable \( u_t \).

Now the call to SSmodel should be:

\[
\text{sys} = \text{SSmodel('y', y, 'u', u, 'model', @example3, 'OBJ', @llikc)};
\]

Here, the duplet \{‘u’, u\} tells SSpace which is the input variable to use. The estimation of the coefficient measuring the jump is negative and significant.

**Case 3**

A final case, still worth mentioning, consists of including the dummy input variable into the user function as an additional input argument, but now the SS system is considered as a system without inputs:

\[
\text{function model = example4(p, u)}
\]
\[
\text{model.T} = 1;
\]
model.Gam = [];  
model.R = 1;    
model.Z = 1;    
model.D = p(2)*u;  
model.C = 1;    
model.Q = 1;    
model.H = 10.^p(1);  
model.S = 0;    

The call now should be:

sys = SSmodel('y', y, 'model', @example4, 'OBJ', @llikc, 'user_inputs', {u});

Additional inputs to the user function may be passed on to the model definition with the help of duplets {'user_inputs', {u}}. Though it does not make sense in this case, it is worth noting that the dependence to the inputs may be modelled by a non-linear function, e.g., just by defining model.D = p(2)*(u(2,:) .^ (p(3))) or any other specification. This is the main advantages of specifying models by writing a function. More examples may be checked in demo5.

4.2. Example 2: Univariate models

In this second example the air-passenger data taken from Box, Jenkins, Reinsel, and Ljung (2015) is analysed with a number of different univariate possibilities.

Case 1: Basic Structural Model (BSM)

A BSM a la Harvey (Harvey 1989) may be used with the template SampleBSM (it is also possible to use SampleSS and test the SS-form from scratch). In such template, separate definitions are in place for the trend and the harmonics and input variables, if any. Have a look on the function demo_airpasbsm.m that is one of the cases implemented in demo2.

The part of the template related to the trends is:

\[
\begin{align*}
\text{m} &= \quad ;
\text{I} &= \text{eye}(\text{m}); \\
\text{O} &= \text{zeros}(\text{m}); \\
\text{TT} &= [\text{I \ I}; \text{O \ I}]; \\
\text{ZT} &= [\text{I \ O}]; \\
\text{RT} &= [\text{I \ O}; \text{O \ I}]; \\
\text{QT} &= [];
\end{align*}
\]

where the variable \( m \) is the number of output variables (1 for univariate) and matrices \( \mathbf{I} \) and \( \mathbf{O} \) are pre-defined as a unity matrix and a block of zeros. The model is specified directly by the State Space form of the Local Linear Trend type (LLT), i.e.,

\[
\begin{pmatrix}
\alpha_{1,t+1} \\
\alpha_{2,t+1}
\end{pmatrix} = 
\begin{pmatrix}
1 & 1 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
\alpha_{1,t} \\
\alpha_{2,t}
\end{pmatrix} + 
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
\eta_{1,t} \\
\eta_{2,t+1}
\end{pmatrix}
\]
A LLT is fully specified by filling in the missing variables in the previous listing, i.e., \( m = 1 \), and \( QT = [10.^p(1) \ 0; \ 0 \ 10.^p(2)] \). Other possibilities are implemented by small variations to this option, one interesting case is an Integrated Random Walk (IRW) or smooth trend with one single noise by setting \( RT = [0; \ I] \); \( QT = 10.^p(1) \); (Young et al. 1999).

The periodic components (either seasonal or cyclical) in SampleBSM may be coded as either trigonometric or dummy seasonality. For the trigonometric case the empty code in the template is:

\[
\begin{align*}
\text{Period} &= []; \\
\text{Rho} &= []; \\
\text{Qs} &= \text{repmat}(, , );
\end{align*}
\]

This portion of the code filled in for a monthly time series converts to:

\[
\begin{align*}
\text{Period} &= [12 \ 6 \ 4 \ 3 \ 2.4 \ 2]; \\
\text{Rho} &= [1 \ 1 \ 1 \ 1 \ 1 \ 1]; \\
\text{Qs} &= \text{repmat(varmatrix(p(3)), 1, 6)};
\end{align*}
\]

The argument \( \text{Period} \) is used to provide the periods for the seasonal and/or cyclical, \( \text{Rho} \) is the damping factor of each harmonic sinusoid, i.e., values between zero and one and values of unity recommended for seasonal harmonics. The variances for each harmonic are introduced via the \( \text{Qs} \) matrix (all variances equal for all the harmonics in the example). If all variances have to be different the code would be \( \text{Qs} = \text{varmatrix}(p(3:8))' \). The function \text{varmatrix} provides an alternative way of building semi positive definite covariance matrices.

The rest of the template is:

\[
\begin{align*}
\text{H} &= \text{varmatrix}(p(4)); \\
\text{D} &= [];
\end{align*}
\]

where \( \text{H} \) is the variance of the observed noise and matrix \( \text{D} \) is included to deal with input variables (none in this case).

Once the BSM model is set up, the code to run is as follows:

```matlab
load airpas;
y = log(airpas);
sys = SSmodel('y', y, 'model', @demo_airpasbsm);
sys = SSestim(sys);
sys = SSsmooth(sys);
T = sys.a(1, :);
I = y-sys.yfit';
S = y-I-T;
```

Here the trend, Seasonal and Irregular components are estimated by combinations of the estimated states and stored in \( T \), \( S \) and \( I \) matrices, respectively.
Case 2: ARIMA

The ARIMA model is easily implemented by using the SampleARIMA template used as an example in demo3, check demo_airpasarima. Assuming that an $ARIMA(0,1,1) \times (0,1,1)_{12}$ is to be estimated, the relevant part of this template is

\[
\text{Sigma} = \text{varmatrix}(p(3));
\text{Diffy} = \text{conv}([1 -1], [1 \text{ zeros(1, 11) -1}])';
\text{ARpoly} = 1;
\text{MApoly} = \text{conv}([1 p(1)], [1 \text{ zeros(1, 11) p(2) }]);
\]

where Diffy is the differencing polynomial in the backshift operator $B$ such that $B^j y_t = y_{t-j}$, ARpoly is the AR polynomial and MApoly is the MA polynomial. Check that the differencing order is the convolution (i.e., multiplication) of a regular and seasonal difference operators, i.e., $\Delta \Delta^{12} = (1-B)(1-B^{12})$. The MA polynomial of the model is $(1 + \theta_1 B)(1 + \Theta_1 B^{12})$, with $p(1) = \theta_1$ and $p(2) = \Theta_1$. All polynomials are coded as column vectors of the corresponding coefficients in ascending order of powers of the backshift operator, as it is common in other MATLAB toolboxes. Though it does not make sense in this example, constraints among parameters would be very simple to impose, e.g., if the constraint $\theta_1 = \Theta_1$ is needed, the MA polynomial may be coded as MApoly = conv([1 p(1)], [1 zeros(1, 11) p(1)]).

Case 3: Exponential Smoothing

A final illustration for this data is an Exponential Smoothing model a la Hyndman, Koehler, Ord, and Snyder (2008), which template is SampleES, check demo_airpasES used in demo3. The template filled in for this example is:

\[
\text{ModelType} = 'AAA12';
\text{Phi} = [];
\text{Alpha} = \text{constrain}(p(1), [0 2]);
\text{Beta} = \text{constrain}(p(2), [0 4-2*Alpha]);
\text{AlphaS} = \text{constrain}(p(3), [0 2]);
\text{D} = [];
\text{Sigma} = \text{varmatrix}(p(4));
\]

The argument ModelType deals with the type of model, the first letter stands for the type of level (‘N’ for none, ‘A’ for additive and ‘D’ for damped with damping factor Phi), the second letter is the slope type (‘N’ for none or ‘A’ for additive), and the third letter stands for the type of seasonal (again either ‘N’ or ‘A’), and the numbers after those letters is the period of the seasonal component. The rest of the code sets up the $\alpha$, $\beta$ and $\gamma$ parameters affecting the level, slope and seasonal components, that are suppose to be estimated within certain values, see Hyndman et al. (2008). Sigma stands for the variance of the unique noise present in the model. Input variables may be included by introducing a D matrix as a function of the p vector of parameters.

4.3. Example 3: Multivariate dynamic harmonic regression (DHR)

The models presented so far are univariate and may be easily extended to multivariate versions. In this example we present a new model that has not yet been used in its multivariate
version, namely the multivariate Dynamic Harmonic Regression. It is built in the spirit of seemingly unrelated equations models, i.e., the relation among the endogenous variables is not explicit in the equations, but are modelled exclusively via non-diagonal covariance matrices of the different noises. The specific formulation is given in Equation 6.

\[
y_t = T_t + C_t + S_t + f(u_t) + \epsilon_t
\]

\[
C_t + S_t = \sum_{i=1}^{k} [A_{it}\cos(\omega_{it}) + B_{it}\sin(\omega_{it})]
\]

Obviously \( y_t \) is multivariate in this case; \( T_t, C_t, S_t \) are a set of trends, cycles and seasonals, respectively; \( f(u_t) \) models linear or non-linear relations to inputs; \( \epsilon_t \) is a white noise variable with full covariance matrix \( H_t \); \( \omega_i (i = 1, 2, \ldots, k) \) are the frequencies for the periodic components that typically include the seasonal fundamental frequency and all the harmonics down to \( \pi \) but also may include cyclical frequencies; and \( A_{it} \) and \( B_{it} \) are diagonal matrices of time varying parameters. Usually Random Walk parameters may suffice, but other options may be preferred in specific applications. In essence the DHR model is effectively a multivariate Fourier analysis in particular frequencies with time varying parameters. The univariate version of these models estimated in the frequency domain where explored by Young et al. (1999) and Taylor et al. (2007).

These type of models may be easily implemented in SSpace with the help of the template SampleDHR. This template resembles SampleBSM of the previous example, with a user function that needs an additional input argument that have to be taken into account in the SSmodel call, i.e., the size in time of the sample. In the case of a trivariate model of quarterly data is implemented in demo_energydhr used in demo4. A compacted version of such template is

```matlab
function model = demo_energydhr(p, N)
% Trend
m = 3;
I = eye(m); O = zeros(m);
TT = [I I; O I];
ZT = [I O];
RT = [I O; O I];
QT = blkdiag(varmatrix(p(1:6)), varmatrix(p(7:12)));
% Seasonal/cyclical DHR components
Periods = repmat([4 2], 3, 1);
Rho = ones(3, 2);
Qs = repmat(varmatrix(p(13:18)), 1, 2);
% Covariance matrix of irregular component (observed noise)
H = varmatrix(p(19:24));
```

There are a total of 24 unknown parameters, all of them located at covariances matrices. It is important to note here the use of the varmatrix function, used to convert any set of arbitrary parameters into a semi-positive covariance matrix. Beware that, since covariance matrices are symmetrical, a \( 3 \times 3 \) matrix have 9 elements, but only 6 of them are different. This function allows also to impose rank and other constraints to build homogeneous models, etc. (see the SSpace documentation). It is also easy to check that there are only two harmonics and both are estimated with the same covariance matrices, check \( Qs \) matrix.

Such model may be run on the energy data of Harvey (1989), with the following code:
load energy
y = log(energy);
p0 = repmat([-2 -2 -2 0 0 0]', 4, 1);
sys = SSmodel('y', y, 'model', @demo_energydhr, 'p0', p0, ...
    'user_inputs', length(y));
sys = SSestim(sys);
sys = SSsmooth(sys);
Trend = sys.a(1:3, :);
Irregular = y-sys.yfit';
Seasonal = y-Trend-Irregular;

It is important to initialise the searching algorithm with appropriate diagonal covariance matrices, this is achieved by the setting of p0 above. The call to SSmodel is done with the required initial parameters and the number of time samples.

Figure 3 shows some output of this DHR model.

Figure 3: Unobserved components of DHR example.

4.4. Example 4: Non-gaussian

The number of van drivers casualties in the UK, see Durbin and Koopman (2012) and Figure 4, is a case where a Poisson distribution is justified, because numbers are small and the units are integers. Neither the Gaussian assumption is strictly correct nor the logarithmic transformation manage to produce sensible results. The model used is a BSM with a trend, dummy seasonal, irregular and an exogenous effect due to the enforcement of the seat belt law, but the distribution of the observations is a Poisson, see Equation 7.
In order to set up this model in \texttt{SSSpace} we have to create two user functions, one with the linear model, i.e., the BSM with a dummy seasonal (beware that there is no noise in the observation equation) by using the \texttt{SampleBSM} template and a second one to change the observation equation into a Poisson model with the help of \texttt{SampleEXP} template (other distributions available are Binary, Binomial(n), Negative Binomial and Exponential). Both functions may be checked in \texttt{demo\_van\_poisson}, used in \texttt{demo7}. The linear model is (function \texttt{demo\_van} in file \texttt{demo\_vanl.m}):

\begin{verbatim}
function model = demo_van(p, H)
  TT = 1;
  ZT = 1;
  RT = 1;
  QT = varmatrix(p(1));
  Period = 12;
  Qs = 0;
  D = p(2);
end
\end{verbatim}

As it may be seen, the model is just a Random Walk trend with a dummy seasonal component and a dummy effect dealing with the step change due to the seat belt law enforcement.

The second user model is rather simple, it consists of a call to the linear model above and just telling \texttt{SSSpace} that the observations follow a Poisson distribution. This function has the peculiarity that necessarily has to have two input arguments, while the user may add as many as the model require for its definition:

\begin{verbatim}
function model = demo_van_poisson(p, H)
  model = demo_van(p, H);
  model.dist = Poisson;
end
\end{verbatim}

Then, the following code would produce the analysis for the data with some artificial missing values in the middle and some at the end to produce the forecasts:

\begin{verbatim}
load Van
y = Van(:, 1);
u = Van(:, 2);
y(50:55) = nan;
y = [y; nan(20, 1)];
u = [u; ones(20, 1)];
sys = SSmodel('y', y, 'u', u, 'model', @demo_van_poisson);
\end{verbatim}
sys = SSestim(sys);
sys = SSvalidate(sys);
sys = SSsmooth(sys);

Figure 4 shows the trend with the jump due to the seatbelt law effect and twice the standard error confidence bands.

```
function model = demo_uk(p)
    TT = [1 1; 0 1];
    ZT = [1 0];
    RT = [0; 1];
    QT = varmatrix(p(1));
    % Trigonometric seasonal model
```

4.5. Example 5: Non-linear

This example illustrates the use of the Extended Kalman Filter with exact initialisation applied to the monthly visits abroad by UK residents from January 1980 to December 2006 following Koopman and Lee (2009) and Durbin and Koopman (2012). This example was used to test for the convenience of the log transform so widely use in Econometrics. As a matter of fact, it is shown that the log transform does not fix the heteroscedasticity problem, and, consequently a model with an interaction between the trend and the seasonal component is proposed instead with the rest of hypothesis applying, see Equation 8.

\[
y_t = \text{Trend}_t + \text{Cycle}_t + \exp\{b\text{Trend}_t\}\text{Seasonal}_t + \epsilon_t
\]  

Comparing this equation with the general non-linear system (3) we see that there is only one non-linear term, namely \( T_t(\alpha_t) \) that is the Equation 3 without the noise. Setting up the model now is much more difficult because the partial derivatives of \( T_t(\alpha_t) \) with respect to the vector state \( \alpha_t \) ought to be calculated explicitly. For convenience a linear BSM model is implemented in a separate model using the `SampleBSM` template (check `demo_uk` in `demo8`):

\[
y_t = \text{Trend}_t + \text{Cycle}_t + \exp\{b\text{Trend}_t\}\text{Seasonal}_t + \epsilon_t
\]  

```
Period = [constrain(p(3), [18 800]) 12 6 4 3 2.4 2];
Rho = [constrain(p(4), [0.5 1]) 1 1 1 1 1 1];
Qs = [varmatrix(p(5)) repmat(varmatrix(p(6)), 1, 6)];
% Observed noise variance
H = varmatrix(p(2));

This model has some singularities with respect to previous listing in this paper. Firstly, an
Integrated Random Walk or smooth trend is chosen that depends on just one parameter
(p(1)). Secondly, a cycle is introduced into the model by adding one element to all the
arguments in the function having to do with the seasonal component. Thirdly, one interesting
point is that the period of such cycle is estimated as a constrained value between 18 and 800
months (one an a half year and 67 years, see the use of constrain in Period). Fourthly, the
cycle is modulated by a damping factor estimated as a value between 0.5 and 1 (again with
the help of the constrain function). Finally, separate variance values for the cycle and the
seasonal components are estimated.

The non-linear model is now built with the template SampleNL (check model_uknle):

function model = demo_uknle(p, at, ctrl)
model1 = demo_uk(p(1:6));
% Defining linear matrices
if ctrl< 2
    model.T = model1.T;
    model.Gam = [];
    model.R = model1.R;
    model.Z = [];
    model.D = [];
    model.C = 1;
    model.Q = model1.Q;
    model.H = model1.H;
    model.p = p;
    model.S = [];
end
% Defining nonlinear matrices in State Equation
if any(ctrl== [2 0])
    % Code defining derivative of matrix T(a(t)) (ns x Nsigma)
    model.dTa = [];
    % Code defining matrix T(a(t)) (ns x Nsigma)
    model.Ta = [];
    % Code defining matrix R(a(t)) (ns x neps x (1 or n))
    model.Ra = [];
    % Code defining matrix Q(a(t)) (neps x neps x (1 or n))
    model.Qa = [];
end
% Defining nonlinear matrices in Observation Equation
if any(ctrl== [3 0])
    b = p(7);
    expfun = exp(b*at(1));
This template has three compulsory input arguments, i.e., the parameter vector \( p \), the current state vector \( \mathbf{a}(t) \), and a variable that controls the execution (\( \mathbf{ctrl} \)). As a first step, the function calls the linear model \( \text{demo.uk} \) in order to set up all the matrices in the state equation. Then the system matrices are redefined in three blocks, i) linear matrices, ii) non-linear or state-dependent matrices in state equation, iii) non-linear or state-dependent matrices in observation equation. Only one definition for each matrix in any of the blocks are permitted, in order to avoid errors. Beware that both \( \mathbf{Z}_t(\alpha_t) \) (in \( \text{model.Za} \)) and \( \frac{\partial \mathbf{Z}_t(\alpha_t)}{\partial \alpha_t} \) (\( \text{model.dZa} \)) have to be correctly specified.

Figure 4.5 shows the estimated components.

![Series and Trend](image)

![Cycle](image)

![Seasonal](image)

**Figure 5:** Series and trend, cycle and raw seasonal and scaled seasonal (\( \exp\{b \text{Trend}_t\}\) Seasonal\(_t\)) of Non-linear example.

### 4.6. Further features

More templates and more demos are available in \textbf{SSpace}, but are left out of this paper for the sake of clarity and space. They may be checked by running carefully the 8 demos included in
the toolbox. Some of the most relevant are linear, non-linear and time varying regressions (see SampleDLR and demo5), concatenation of State Space systems with the SampleCAT template (a typical case would be a BSM model with time varying parameters), estimating parameters of any model by minimizing functions of squared of several-steps-ahead forecast errors (see e.g., demo2), time aggregation problems (see SampleAGG and demo6), and nesting in inputs models (see SampleNEST and demo5).

5. Concluding remarks

This paper has presented SSpace, a new toolbox for the exploitation of State Space models. It is intended for a wide audience, including professional practitioners, researchers and students, indeed anyone involved in the analysis of time series, forecasting or signal processing.

The library incorporates most of modern algorithms and advances in the field of State Space modelling, following mainly Taylor et al. (2007) and Durbin and Koopman (2012). The system is very general because it is possible to implement linear, non-gaussian and non-linear systems, all system matrices may vary over time and may be multivariate, several estimation methods are implemented, inputs to the system may be introduced explicitly, etc.

Other advantages of the library are that a few functions are necessary to carry out a comprehensive analysis of time series. Such functions are used systematically following a fixed pattern that simplifies the usage of the toolbox. However, one of the main feature that makes SSpace really flexible, powerful and transparent is that the user implements models directly as a function written in MATLAB. This approach makes some cumbersome tasks truly simple and transparent, this is the case, say of trying different parameterizations of the same model or imposing parameter constraints.

The toolbox is supplied with templates for building general SS models from scratch in a completely free way, but is also accompanied by a number of templates useful for the implementation of a variety of common models.

In this paper, the capabilities of the toolbox have been demonstrated in action on several worked examples. These properties should make the toolbox particularly interesting for those in need of non-standard models, for which even many commercial alternatives may not provide the required flexibility.

Acknowledgments

We acknowledge the interesting comments of J.R. Trapero from UCLM and C.J. Taylor from Lancaster University, that substantially improved the manuscript.

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].

References


**Affiliation:**
Marco A. Villegas & Diego J. Pedregal  
ETSI Industriales  
Institute of Applied Mathematics in Science and Engineering (IMACI)  
Universidad de Castilla-La Mancha  
13071 Ciudad Real  
Telephone: +34/(9)26/295430  
E-mail: Marco.Villegas@uclm.es & Diego.Pedregal@uclm.es  
URL: http://www.uclm.es/profesorado/diego/