State Space modelling for practitioners from the hand of SSpace

Diego J. Pedregal

INTRODUCTION

State Space systems (SS) are a very powerful and useful machinery for time series and econometrics modelling, with applications in many areas of business, industry and other fields. It has been relegated to a corner because it has remained as an area of research mainly and it has not gone very deep into the practitioner's community.
However, I am persuaded that the discussion of SS systems may be also addressed from a practitioner's point of view, not entering deep statistical theory that supports them, but addressing the practical aspects of implementation, and being guided by a powerful software. This is the case of SSpace, a MATLAB toolbox written to deal with SS systems that is freely available (Villegas, M.A., Pedregal, D.J., 2018a; https://bitbucket.org/predilab/sspace-matlab/).
Since there is a technical long paper published about SSpace, this note will focus on how easy is to implement in SSpace typical models that most practitioners actually use, in order to overcome the misgivings about SS systems. Moreover, the note will show how easy and intuitive is to build standard models in SSpace with the help of available templates (for Exponential Smoothing, ARIMA, Unobserved Components, regression, transfer functions, VARMAX, non-linear and non-gaussian, etc.). The note also shows how easy is to implement models with especial features that transcends the standard ones, like models with heteroskedastic noise; models with time varying parameters; non-linear relations among variables; imposing arbitrary constraints; etc. There other topics that are not treated here that are easily implementable in SSpace, like time aggregation; nesting models; hierarchical forecasting; etc. One main advantage is that all these models are run within one unified and friendly environment, mainly because SS is quite a powerful approach!!

FIRST, SOME THEORY

This section shows the minimum amount of theory compatible with a decent SS systems understanding. SS systems express any dynamical time series models with two equations:

1. *Transition* or *state equation*: represents the dynamic part of the model and resembles a VAR(1) model for the system state vector. The state vector is composed of stochastic intermediate variables that may or may not have interpretation and may depend on a set of known inputs.
2. *Observation equation*: specifies the relation of output time series with states, inputs and noises.

The SSpace formulation is the most general I have ever seen. Actually, it has some elements that experts would immediately identify as redundant. But they are included here for the sake of generality:

$$\textbf{Transition equation:}\quad \alpha_{t+1} = T_t\alpha_t + \Gamma_t + R_t\eta_t$$
$$\textbf{Observation equation:}\quad y_t = Z_t\alpha_t + D_t + C_t\epsilon_t$$
$$\eta_t \approx N(0, Q_t);\quad \epsilon_t \approx N(0, H_t);\quad cov(\eta_t, \epsilon_t) = S_t$$

In these equations α_t is the state vector, y_t is the output data, η_t and ϵ_t are Gaussian noises with properties showed in the last line, the rest are the so-called system matrices. This set up shows up several very general properties:

i) State equation is dynamic (state vector appears at different time tics), while the observation equation is static.
ii) y_t, the output data, may be multivariate. Therefore, all terms in the equations are either vectors or matrices.
iii) Noises in transition and observation equations may be correlated. By this recourse both noises may coincide, producing a single source of noise system as a particular case.
iv) All matrices are potentially time varying. It is especially interesting that the noise variance matrices may be time varying, opening the door to heteroskedastic models.
v) Input-output relations may be implemented by means of Γ_t and D_t matrices, with appropriate parameterisation as, for example, $D_t = f(u_t, \beta)$, being $f(\blacksquare)$ an arbitrary function, u_t a set of input variables and β some parameters.

For a model fully specified, i.e., with particular values for all the system matrices, some recursive algorithms provide the optimal estimates of the states, forecasts, fitted values, innovations, perturbations, etc. Such algorithms are the Kalman Filter, smoothing and disturbance algorithms (Harvey, 1989; Pedregal and Young, 2002; Durbin and Koopman, 2012).

Very rarely all the information about the model is available, and the usual situation is that some of the matrix values are unknown (typically the noise variances). Such unknowns may be estimated by Maximum Likelihood (ML) by repeated runs of the Kalman Filter with different sets of parameters. Usually quasi-Newton search algorithms with some line search strategy are used to find the optimal values for the unknowns.

Most of the complications of SS models rely on the different formulation of the recursive algorithms, estimation of unknowns, etc. Such complications are already inside SSpace and are automatically chosen appropriately according to each particular case. But what is really relevant and necessary for practitioners to gain the power of SS modelling is exclusively becoming acquainted with the general formulation above.

SOME EXAMPLES

Consider now the following simple, though interesting examples. It is worthy to spend several minutes on them, making sure how all the systems proposed here fit into the general SS form in the previous section:

1) Non-linear regression: all system matrices do not exist (or are 0) but $D_t = f(u_t, \beta)$, $C_t = 1$ and $H_t = H$. The unknowns are β and H, that will be estimated by ML. Linear regression would be similar with $D_t = u_t \beta$.
2) AR(1): all matrices are zero, except $T_t = \phi$, $R_t = Z_t = 1$, $Q_t = Q$. The unknowns here are ϕ and Q.
3) Random Walk: Same as previous, with $\phi = 1$.
4) Level + noise model: $T_t = R_t = Z_t = C_t = 1$, $Q_t = Q$, $H_t = H$. Unknowns are Q and H (or just the ratio of the two, i.e. Q/H).
5) Hodrick-Prescott filter:

$$T_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix};\ R_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix};\ Z_t = [1 \quad 0];\ C_t = 1;\ \lambda = \frac{H_t}{Q_t} = \frac{H}{Q}$$

All these examples are very simple, but things start to get interesting when realising that multivariate versions are straightforward by converting any scalar element of the previous examples to matrices. With some caution, because any ‘zero’ expands to a squared matrix of zeroes, any ‘one’ converts to an identity matrix and any variance parameter changes to a semidefinite positive matrix. Another interesting extension is that the non-linear (or linear) regression in 1) above may be attached to any of the other models straightaway, producing either an AR(1), a Random Walk, level+noise or HP with inputs related through any arbitrary function. What would be the multivariate or the Hodrick-Prescott filter with inputs in a non-SS formulation? Do not waste time figuring it out yourself or looking for it: SSpace will give you the answer straightaway!!

NOW IS TIME FOR PRACTICE…

… from the hand of SSpace. The modelling procedure with SSpace resembles closely the usual one in time series, and is composed of several steps:

i) Specification of the model.
ii) Coding the model in a standard MATLAB function.
iii) Setting up the model.
iv) Estimation.
v) Validation.
vi) Use of model: filtering, smoothing, forecasting, signal extraction, etc.

A great deal of the power and flexibility of SSpace lies on step ii), consisting on writing the SS system as a standard MATLAB function. The good news is that the user does not need to be an expert on SS theory to feel the power of SS systems with SSpace. Actually, one may skip the previous theoretical section and still be able to take advantage of SSpace. Obviously, SSpace allows for the direct specification of raw SS systems from scratch if required, but it also offers the possibility of modelling standard well-known methods with the aid of templates already available with the toolbox. The conversion to SS form is made internally by SSpace and the user even would not realise he is using a SS system at all. Additional good news is that steps iii)-vi) have a common syntax for any model (actually they can be grouped together in one single MATLAB function) and then just a few functions do a full time series analysis. Summing up, doing a powerful time series analysis with SSpace is rather straightforward!

ONE CASE

Let's consider an AR(1) written from scratch in SSpace according to the SS form given above (all matrices are zero or do not exist, except $T_t = \phi$, $R_t = Z_t = 1$, $Q_t = Q$, being the unknowns ϕ and Q). Step ii) in previous section comprises editing the general template for SS systems (called SampleSS.m) and filling in the values for the system matrices. The template (after removing the abundant help that explains how it should be filled in) looks like the following listing:

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

The input to this function is a vector of unknown variables that are distributed among the system matrices; the output is a MATLAB structure with the model (i.e., the system matrices with the same names as in the equations in the previous sections). The implementation of an AR(1) with this template, once it has been saved with a different name to keep the copy of the original template intact is:

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

It is clear that the first element of the parameter vector (p) plays the role of the AR coefficient (ϕ), while the second element is the noise variance. Well, to be exact, matrix Q is a transformation of p(2) to ensure that matrix Q is positive, regardless of the value of p(2). In other words, varmatrix is a function that maps any real positive or negative number into a positive value (or any vector of arbitrary values into a semidefinite positive matrix in multivariate models). In the similar way, model.T could have been defined as model.T = constrain(p(1), [-1 1]), implying that any value of p(1) is mapped into a (-1, 1) interval, forcing the model to be stationary (this is what function constrain does, also valid for vectors).
It may have been very simple or complicated to set up this model, but once it is done, it can be stored for ever for future use and applied to as many series as required. In this way, a catalogue of different useful models may be stored by the users as they are correctly implemented and tested.
Now that the model is fully written it is time to do the rest of the analysis, namely steps iii) – vi) in previous section. Assuming there is a time series in memory stored in vector y, the code to carry out these steps are more or less always the same (I hope the function names are self-explanatory):

iii)	Setting up model:	sys = SSmodel('y', y, 'model', @ar1);
iv)	Estimation:	sys = SSestim(sys);
v)	Validation:	sys = SSvalidate(sys);
vi)	Use (filtering):	sys = SSfilter(sys);

Obviously, there are many more customisable options in function SSmodel, that governs how the posterior analysis is performed. Mind that at each step the input and the output to these functions are always the same sys (though they could be different). This implies that each function fills in new elements to the system in such a way that, at the last step the object sys contains all the relevant information about the system.
The validation step with SSvalidate produces a table with a full account of the estimation procedure and diagnostic checks. The last step produces automatic interpolation and extrapolation (forecasts) of the series if vector y contains NaN values in the middle or at the end of the time series. The forecast values may be retrieved from sys.yfor. See the toolbox documentation for a full account of the fields of sys.

FURTHER EXAMPLES

Writing more complex models than the previous one in SS form is a cumbersome process with a lot of pitfalls difficult to avoid. Remembering the general SS form of any sort of model is only reserved to very privileged minds. This is the reason why building templates for popular methods is so important. They should be done in a way that the user defines the model directly in terms of the method idiosyncrasy, instead of doing it in terms of the SS form. Therefore, SSpace provides templates for several useful models (and more may be built in the future): Unobserved Components (either Dynamic Harmonic Regression or Basic Structural Models), ARIMA, Exponential Smoothing, linear and dynamic linear regression (time varying coefficients), non-Gaussian models, Stochastic Volatility models, non-linear models. There are additional templates to deal with interesting problems like time aggregation, concatenation of SS models and models with nested inputs. Remember that multivariate versions of most of previous models are also available and that in all of them any linear, non-linear, transfer function, time-varying regression or even a mixture of them may be included to model the relations between outputs and inputs. Therefore, SSpace is truly a toolbox full of useful tools!!

Let's illustrate some of these interesting features running on the fatalities in Spanish roads between January 1998 and December 2017 (see Figure 1). Two models are presented here as mere illustrations of the SSpace capabilities (for a deeper analysis of this series see Castillo et al., 2010).

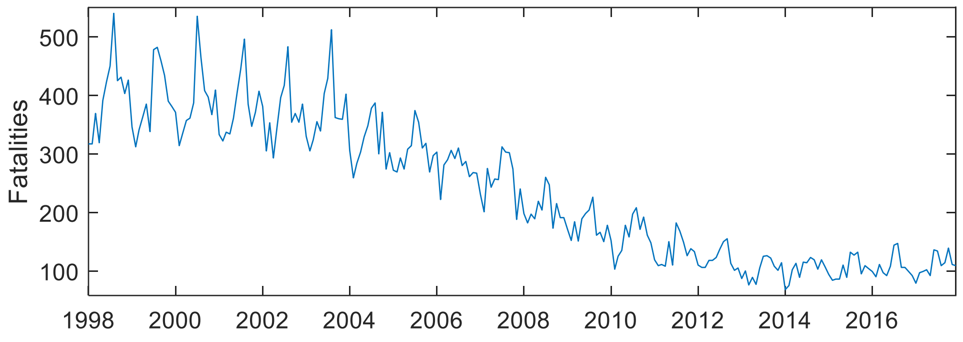

Figure 1. Traffic fatalities in Spain

Setting up an exponential smoothing model is rather simple, the most difficult part is just editing the SampleES.m template, that looks like the following listing (removing all the help around):

```
function model = SampleES(p)
ModelType = 'AAA12';
Phi = constrain(p(some_item), [0 1]);
Alpha = constrain(p(some_item), [0 2]);
Beta =  constrain(p(some_item), [0 4-2*Alpha]);
AlphaS = constrain(p(some_item), [0 2]);
D = [];
Sigma = varmatrix(p(some_item));
```

The template is prepared for general multivariate models. The user just must select the model type in string variable ModelType: three letters for level, slope and seasonal, respectively, followed by a number that specifies the seasonal period; any letter may be A for additive component, N for none, D for damped. The rest of options are the damping factor for trend (Phi), and smoothing constants for each component included (Alpha, Beta, AlphaS). Finally, influence of exogenous variables may be introduced via D matrix and the observation noise variance is included in Sigma. The model also allows for autocorrelated noise, but that part has been removed here for simplicity. Then, specifying an exponential model in SSpace entails just selecting an appropriate ModelType and introducing correlative numbers whenever a some_item is encountered.

The template for ARIMA models is still simpler, because it involves only the specification of the polynomials as vectors of parameters in the back-shift operator in column form, i.e.,

```
function model= SampleARIMA(p)
Sigma = [];
```

```
DIFFpoly= [];
ARpoly = [];
MApoly = [];
D= [];
```

This template prepared for an ‘airline’ model without inputs would be:

```
function model= airline(p)
Sigma = varmatrix(p(1));
DIFFpoly= conv([1; -1], [1; zeros(11, 1); -1]);
ARpoly = 1;
MApoly = conv([1; p(2)], [1; zeros(11, 1); p(3)]);
```

In this model DIFFpoly and MApoly are the product of regular and seasonal polynomials in the back-shift operator, carried out by the conv function. In the hypothetical case that the regular MA(1) coefficient needs to be equal to the seasonal one, the previous MApoly should be conv([1; p(2)], [1; zeros(11, 1); p(2)]), instead. In other words, imposing arbitrary constraints in any model is very easy with SSpace.
Once the models are prepared, the rest of the analysis is straightforward. Assuming the data has been loaded on a vector y, the code for setting up the models for both cases above would be very similar indeed:

```
sysES    = SSmodel('y', y, 'model', @modelES, 'p0', -3);
sysARIMA = SSmodel('y', y, 'model', @airline, 'p0', [-3; 0; 0]);
```

Here one additional detail is important, the initial conditions for the parameters to start the search. In general, good initial conditions for variance parameters inside the varmatrix function is -3 (varmatrix(-3) = 0.0025). However, for other parameters, like the moving average coefficients in the ARIMA model, a good initial condition in general is 0.
The rest of the code is already known:

```
sys# = SSestim(sys#);
sys# = SSvalidate(sys#);
sys# = SSfilter(sys#);
```

where sys# stands for either sysES or sysARIMA. Command SSvalidate on the ARIMA model produces a table like the following one with a full account of the estimation process and diagnostic checks (autocorrelation, heteroskedasticity and Gaussianity):

```
-----------------------------------------------------
          Param    S.E.   T-test  P-value |Gradient|
-----------------------------------------------------
 p(1)   -2.2607  0.0496  45.6086   0.0000   0.000000
 p(2)   -0.7816  0.0410  19.0714   0.0000   0.000000
 p(3)   -0.8219  0.0809  10.1620   0.0000   0.000000
-----------------------------------------------------
               AIC: -1.4544
               SBC:  -1.1884
               HQC:  -1.3488
    Log-likelihood: 163.6219
      Corrected R2:   0.9584
Residual Variances:   0.0114
-----------------------------------------------------
Summary Statistics:
-----------------------------------------------------
 Missing data points     0.0000
 Q( 1)                   1.4405
 Q( 4)                   3.1795
 Q( 8)                   8.6719
 Q(12)                  13.7786
 H(67)                   0.4054
 P-value                0.0003
 Bera-Jarque            0.2770
 P-value                0.8707
-----------------------------------------------------
```

If variable y is padded with NaN values at the end, the Kalman Filter (SSfilter) produces and stores forecasts in sys#.yfor.

A final example expanding on the previous ones is included here to give an idea of the flexibility of SSpace when modelling exogenous variables in a 'mischievous' way. This is so because we introduce a transfer function without appealing to the SS form of a transfer function!!

Many reforms have been enforced to reduce the traffic casualties in Spain. One that was controversial at the time was the enforcement of a Penalty Point Systems (PPS) driving license in July 2006. Though there were many other reforms, we will concentrate on just the PPS for the sake of illustration (see details in Castillo et al., 2010). One question that interested researchers and society in general is whether the effect of the PPS was permanent or transitory. Far from the controversy, if an analyst would like to implement a transitory change of this kind, a first order transfer function may be used. To do this in SSpace, we have to change any of the model functions above in two ways: i) add a dummy variable (zeros everywhere with just a 1 in July 2006) as an additional input to the function, and ii) add the code to implement the transfer function by means of the matrix D. The new function would be

```
function model = modelES1(p, PPS)
…
D = filter(p(5), [1 p(6)], PPS');
```

where dots indicate that the rest of the function does not change. Function filter filters out variable PPS with a model that consists of a numerator (p(5)) and a first order denominator ([1 p(6)]). The final conclusions on the PPS effect depend on the values of the estimated parameters: i) p(5)=0 implies no effect, ii) p(6)=0 means the effect is just on the first month, iii) p(6)=1 implies the effect is permanent, and iv) *0* <p(6)< *1* supports the transitory effect hypothesis.

The call to set up the model (with SSmodel) has to tell SSpace that it includes an exogenous variable (with the identifier 'user_inputs'), i.e.,

```
sysES = SSmodel('y', y, 'model', @modelES1, 'p0', …
                [-3; -3; -3; -3; 0], 'user_inputs', PPS);
```

In the final estimated model p(5)=-0.19, and p(6)=-0.71 (with an standard errors of 0.08 and 0.13, respectively), supporting the transitory hypothesis.

A final interesting note, maybe for advanced readers, is that any missing value in the input may be interpolated by including it in vector p. Mind that interpolating outputs is a standard operation in many model, but not the interpolation of inputs simultaneously. For example, if a given input to a model has a missing value in observations 100 and 150 they could be interpolated (with estimation of its standard error) by an appropriate definition of matrix D, e.g., D([100 150]) = p(1 : 2). There is no need to interpolate the input as a previous step to the modelling procedure, it may be done all together.

FINAL COMMENTS

State Space is a deep and wide universe of possibilities that expands the modelling possibilities of practitioners in many directions. Some are presented in this brief tutorial, but many have been left out for space reasons. Some of them already implemented in SSpace are certain types of models (Unobserved Components, Dynamic Linear Regression, non-Gaussian, non-linear, multivariate models); other recursive algorithms (smoothing, disturbing); other estimation methods (concentrated likelihood, minimization of forecast errors several steps ahead); other issues in general (time aggregation, nested models in inputs). Readers interested enough are referred to the original note, the toolbox documentation and the demos. Demos are especially illustrative, since they proceed little by little starting from very easy examples to other much more complex.

I must say that SSpace is an always on-going project, since I keep adding new features, debugging, etc. At the moment it is a base on which other more specific approaches may be built. As an 'naughty' example, I may quote a hierarchical approach to forecasting based on SS departing from the main stream, though interesting in itself because of its implications (Villegas and Pedregal, 2018b).

References

Castillo, JI, Castro, M, Pedregal, DJ, (2010), An econometric analysis of the effects of the penalty points system driver's license in Spain, Accident Analysis & Prevention 42 (4): 1310-1319.
Durbin J, Koopman SJ (2012). Time Series Analysis by State Space Methods. 38. Oxford University Press.
Harvey, A.C. (1989), Forecasting, Structural Time Series and the Kalman Filter, Cambridge University Press.
Pedregal, D.J., Young, P.C. (2002), Statistical Approaches to Modelling and Forecasting Time Series. In Clements, M. and Hendry, D. (eds.), Companion to Economic Forecasting (Blackwell Publishers), 69-104.
Taylor, C.J., Pedregal, D.J., Young, P.C., Tych, W., (2007) 'Time series analysis and forecasting with the Captain toolbox', Environmental Modelling and Software, 22, 797-814.
Villegas, M.A., Pedregal, D.J. (2018a), SSpace: A toolbox for State Space modelling, Journal of Statistical Software, 87-5,1-26.
Villegas, M.A., Pedregal, D.J. (2018b), Supply chain decision support systems based on a novel hierarchical forecasting approach, Decision Support Systems, 114, 29-36