Journal of Statistical Software

May 2011, Volume 41, Issue 6. *http://www.jstatsoft.org/*

The State Space Models Toolbox for MATLAB

Jyh-Ying Peng
Academia Sinica

John A. D. Aston
University of Warwick

Abstract

State Space Models (**SSM**) is a MATLAB toolbox for time series analysis by state space methods. The software features fully interactive construction and combination of models, with support for univariate and multivariate models, complex time-varying (dynamic) models, non-Gaussian models, and various standard models such as ARIMA and structural time-series models. The software includes standard functions for Kalman filtering and smoothing, simulation smoothing, likelihood evaluation, parameter estimation, signal extraction and forecasting, with incorporation of exact initialization for filters and smoothers, and support for missing observations and multiple time series input with common analysis structure. The software also includes implementations of TRAMO model selection and Hillmer-Tiao decomposition for ARIMA models. The software will provide a general toolbox for time series analysis on the MATLAB platform, allowing users to take advantage of its readily available graph plotting and general matrix computation capabilities.

Keywords: ARMA model, Kalman filter, state space methods, unobserved components, software tools, TRAMO/SEATS.

1. Introduction

State Space Models (**SSM**) is a MATLAB (The MathWorks, Inc. 2007) toolbox for time series analysis using general state space models and the Kalman filter (Durbin and Koopman 2001). The goal of this software package is to provide users with an intuitive, convenient and efficient way to do general time series modeling within the state space framework. Specifically, it seeks to provide users with easy construction and combination of arbitrary models without having to explicitly define every component of the model, and to maximize transparency in their data analysis usage so no special consideration is needed for any individual model. This is achieved through the unification of all state space models and their extension to non-Gaussian and nonlinear special cases (those which can be linearized). The user creation of custom models is also implemented to be as general, flexible and efficient as possible. Thus, there are often multiple ways of defining a single model and choices as to the parametrization versus initialization and to how the model update functions are implemented. Stock model components are also provided to ease user extension to existing predefined models. Functions that implement standard algorithms such as the Kalman filter and state smoother, log likelihood calculation and parameter estimation will work across all models, including any user defined custom models.

The state space model manipulation procedures are implemented through object-oriented programming primitives provided by MATLAB and classes in the toolbox are defined to conform to MATLAB conventions whenever possible. The standard Kalman filter-based state space algorithms are implemented in C. Thus the toolbox combines efficient state space algorithms (by allowing to run C functions in MATLAB) with full integration into the MATLAB computing environment. Model building is intuitive and model objects behave like standard MATLAB objects, whereas application of state space algorithms to these objects is as fast as comparable programs in C code. Data exploration and custom model building using **SSM** is also greatly facilitated by MATLAB's interactive environment.

The result is an integrated toolbox with support for general state space models and standard state space algorithms, complemented by the built-in matrix computation and graphic plotting capabilities of MATLAB.

1.1. The state space model equations

This section presents a summary of the basic definition of models supported by **SSM**. Currently **SSM** implements the Kalman filter and related algorithms for model and state estimation, hence non-Gaussian or nonlinear models need to be approximated by linear Gaussian models prior to or during estimation. However the approximation is done automatically and seamlessly by the respective routines, even for user-defined non-Gaussian or nonlinear models.

The following notation for various sequences will be used throughout the paper:

y_t	$p \times 1$	Observation sequence
ε_t	$p \times 1$	Observation disturbance (unobserved)
α_t	$m \times 1$	State sequence (unobserved)
η_t	$r \times 1$	State disturbance (unobserved)

Linear Gaussian models

SSM supports linear Gaussian state space models in the form

$$\begin{aligned} y_t &= Z_t\alpha_t + \varepsilon_t, & \varepsilon_t &\sim \mathcal{N}(0, H_t), \\ \alpha_{t+1} &= c_t + T_t\alpha_t + R_t\eta_t, & \eta_t &\sim \mathcal{N}(0, Q_t), \\ \alpha_1 &\sim \mathcal{N}(a_1, P_1), & t &= 1, \dots, n. \end{aligned} \tag{1}$$

Thus the matrices $Z_t, c_t, T_t, R_t, H_t, Q_t, a_1, P_1$ are required to define a linear Gaussian state space model. For details of these matrices refer to Commandeur *et al.* (2011). The matrices and their dimensions are listed here for reference:

Z_t	$p \times m$	State to observation transform matrix
c_t	$m \times 1$	State update constant
T_t	$m \times m$	State update transform matrix
R_t	$m \times r$	State disturbance transform matrix
H_t	$p \times p$	Observation disturbance variance
Q_t	$r \times r$	State disturbance variance
a_1	$m \times 1$	Initial state mean
P_1	$m \times m$	Initial state variance

Non-Gaussian models

SSM supports non-Gaussian state space models in the form

$$\begin{array}{rcll} y_t & \sim & p(y_t|\theta_t), & \theta_t = Z_t\alpha_t, \\ \alpha_{t+1} & = & c_t + T_t\alpha_t + R_t\eta_t, & \eta_t \sim Q_t = p(\eta_t), \\ \alpha_1 & \sim & \mathcal{N}(a_1, P_1), & t = 1, \ldots, n. \end{array} \tag{2}$$

The sequence θ_t is the signal and Q_t is a non-Gaussian distribution (e.g., heavy-tailed distribution). The non-Gaussian observation disturbance can take two forms: an exponential family distribution

$$H_t = p(y_t|\theta_t) = \exp\left[y_t^\top\theta_t - b_t(\theta_t) + c_t(y_t)\right],\ -\infty < \theta_t < \infty, \tag{3}$$

or a non-Gaussian additive noise

$$y_t = \theta_t + \varepsilon_t,\ \varepsilon_t \sim H_t = p(\varepsilon_t). \tag{4}$$

With model combination it is also possible for H_t and Q_t to be a combination of Gaussian distributions (represented by variance matrices) and various non-Gaussian distributions.

Nonlinear models

SSM supports nonlinear state space models in the form

$$\begin{array}{rcll} y_t & = & Z_t(\alpha_t) + \varepsilon_t, & \varepsilon_t \sim \mathcal{N}(0, H_t), \\ \alpha_{t+1} & = & c_t + T_t(\alpha_t) + R_t\eta_t, & \eta_t \sim \mathcal{N}(0, Q_t), \\ \alpha_1 & \sim & \mathcal{N}(a_1, P_1), & t = 1, \ldots, n. \end{array} \tag{5}$$

Z_t and T_t are functions that map $m \times 1$ vectors to $p \times 1$ and $m \times 1$ vectors respectively, and both functions should possess first derivatives. With model combination it is also possible for Z_t and T_t to be a combination of linear functions (matrices) and nonlinear functions.

1.2. Getting started

The easiest and most frequent way to start using **SSM** is by constructing predefined models, as opposed to creating a model from scratch. This section presents some examples of simple time series analysis using predefined models; the complete list of available predefined models can be found in Appendix A.

Model construction

To create an instance of a predefined model, call the functions `ssm_*` where the wildcard is the short name for the model, with arguments as necessary. For example:

- `model = ssm_llm` creates a local level model.
- `model = ssm_arma(p, q)` creates an ARMA(p, q) model.

The resulting variable `model` is a `SSMODEL` object and can be displayed just like any other MATLAB variables. To set or change the model parameters, use `model.param`, which is a row vector that behaves like a MATLAB matrix except its size cannot be changed. The initial conditions usually defaults to exact diffuse initialization, where `model.a1` is zero, and `model.P1` is ∞ on the diagonals, but can likewise be changed. Models can be combined by horizontal concatenation, where only the observation disturbance model of the first one will be retained. For example, N models

$$\texttt{m}_1: \quad y_t^{(1)} = Z_t^{(1)}\alpha_t^{(1)} + \varepsilon_t^{(1)}, \tag{6}$$

$$\texttt{m}_2: \quad y_t^{(2)} = Z_t^{(2)}\alpha_t^{(2)} + \varepsilon_t^{(2)}, \tag{7}$$

$$\cdots$$

$$\texttt{m}_N: \quad y_t^{(N)} = Z_t^{(N)}\alpha_t^{(N)} + \varepsilon_t^{(N)}, \tag{8}$$

can be combined by horizontal concatenation $\texttt{M} = [\texttt{m}_1\ \texttt{m}_2\ \cdots\ \texttt{m}_N]$ into

$$\texttt{M}: \quad y_t = Z_t^{(1)}\alpha_t^{(1)} + Z_t^{(2)}\alpha_t^{(2)} + \cdots + Z_t^{(N)}\alpha_t^{(N)} + \varepsilon_t^{(1)}. \tag{9}$$

More details on the class `SSMODEL` can be found in Peng and Aston (2007).

Model and state estimation

With the model created, estimation can be performed. **SSM** expects the data `y` to be a matrix of dimensions $p \times n$, where n is the data size (or time duration). The model parameters are estimated by maximum likelihood, the `SSMODEL` class method `estimate` performs the estimation. For example:

- `model1 = estimate(y, model0)` estimates the model and stores the result in `model1`, where the parameter values of `model0` is used as initial value.
- `[model1 logL] = estimate(y, model0, psi0, [], optname1, optvalue1, optname2, optvalue2, ...)` estimates the model with `psi0` as the initial parameters using option settings specified with option value pairs, and returns the resulting model and loglikelihood.

After the model is estimated, state estimation can be performed, this is done by the `SSMODEL` class method `kalman` and `statesmo`, which is the Kalman filter and state smoother respectively.

- `[a P] = kalman(y, model)` applies the Kalman filter on `y` and returns the one-step-ahead state prediction and variance.
- `[alphahat V] = statesmo(y, model)` applies the state smoother on `y` and returns the expected state mean and variance.

The filtered and smoothed state sequences `a` and `alphahat` are `m` × `n+1` and `m` × `n` matrices respectively, and the filtered and smoothed state variance sequences `P` and `V` are `m` × `m` × `n+1` and `m` × `m` × `n` matrices respectively, except if `m` = 1, in which case they are squeezed and transposed. The complete list of data analysis functions can be found in Appendix B.

1.3. Toolbox structure and implementation

SSM is composed of two parts, the core C library of state space algorithms (built on top of **LAPACK** Anderson *et al.* 1999) and the MATLAB model object library. This section provides a brief overview of the toolbox, for complete details as well as usage particulars refer to the **State Space Models** manual (Peng and Aston 2007).

C state space algorithms library

The core C library has functions for Kalman filtering and smoothing, with support for exact diffuse initialization, missing values and dynamic observation vector length (p_t varies through time), and unconditional sampling from state space models. A set of MATLAB C mex functions are built on top of the C library implementing the Kalman filter, state smoother, disturbance smoother, fast state smoother, simulation smoother, loglikelihood and gradient calculation, unconditional sampling, signal extraction and filtering and smoothing weights calculation (support for dynamic p_t is suppressed). The C source files are stored in the `csrc` subdirectory, and can all be automatically compiled by running the script `mexall.m`.

MATLAB object library

The MATLAB object library consists of five classes `SSMAT`, `SSDIST`, `SSFUNC`, `SSPARAM` and `SSMODEL`.

The class `SSMAT` represents a state space matrix, with elements marked as variable (dependent on model parameters) and/or dynamic (dependent on time). `SSMAT` objects are designed to mimic MATLAB matrices and, in particular, concatenation and addition are supported.

The classes `SSDIST` and `SSFUNC` are derived from `SSMAT` and represent non-Gaussian distributions and nonlinear functions, respectively. Because **SSM** currently handles both non-Gaussian and nonlinear model elements by linear approximation of the loglikelihood function, derivation of these classes from state space matrices is natural. User definition of custom `SSDIST` objects can be a bit involved, as functions that calculate the approximating (heteroscedastic) Gaussian variances from observation data and the smoothed state, and the (non-Gaussian) loglikelihood from estimation innovation, are required. But once these are defined, the model can then be used in all **SSM** functions without further coding. User definition of `SSFUNC` requires the nonlinear function itself and its first derivative.

The class `SSPARAM` manages the model parameters, including storing and transformation of the values.

The class `SSMODEL` represents a state space model and embeds all the previous classes. `SSMODEL` objects can be horizontally concatenated to form a new model with the sum of the observation vectors from the previous models as its observation. Block diagonal concatenation leads to a model with the concatenation of observation vectors from the previous models as the observation. This new model can then be further horizontally concatenated, or dependence between observation elements can be introduced through changing the observation or state disturbance.

The class methods of `SSMODEL` provides the glue that joins the state space algorithms (ssa) `mex` functions with the object library. Overhead is minimized by only passing the resolved state space matrices H_t, Z_t, etc. to the `mex` functions (in other words, the `mex` functions have no knowledge of model parameters). Parameter estimation is achieved through a set of update functions that transform parameters to model matrices, which are all stored in the `SSMODEL` object level for efficiency and flexibility, since often multiple model elements are dependent on parameter values through the same mechanism, storing the update functions in individual `SSMAT`s leads to the need for either messy global variables or duplicated computation. The disadvantage for this solution is increased complexity in defining custom models. Reducing this complexity will be the objective of future work. The parameter estimation class method itself is implemented in MATLAB and makes use of existing optimization routines (e.g., `fminsearch`), and calls the `mex` function `kalman_int` as part of its operation.

A set of functions for the construction of predefined objects are provided (see Appendix A for a list of predefined models). The functions `ssm_*` returns predefined `SSMODEL` objects for various models (see the appendix for the complete list of predefined models). Auxiliary functions `mat_*`, `fun_*` and `dist_*` constructs predefined individual MATLAB matrices, update functions and SSDIST objects respectively, and can be variously combined for more convenient and flexible construction of custom models.

1.4. Paper overview

The rest of the article proceeds as follows. In Section 2, as in all the papers in this special volume, the local level model will be examined and used to analyze the Nile river data. In the remaining sections, many data analysis examples are conducted, based on the data in the book by Durbin and Koopman (2001), to facilitate easy comparison and understanding of the methods used here. In the interests of brevity, most of the model descriptions are either omitted or only cursorily introduced, as further details can be found either in Durbin and Koopman (2001) or in the introductory paper of this volume (Commandeur *et al.* 2011). In addition, all the examples (plus others as well) are included as demos in the software package. Specifically, Section 3 contains a univariate analysis of the road accident data from (Harvey and Durbin 1986), while Section 4 contains a bivariate analysis. Section 5 contains an analysis using ARMA models, while Section 6 contains an analysis using cubic spline methods. Section 7 reviews an application of the software to seasonal adjustment using ARIMA models. Section 8 shows how the software extends to non-Gaussian models.

2. Local level models – The Nile river data

In this example the Nile data (as seen in the other papers of this volume) will be analyzed using the local level model as described by Harvey (1989). Briefly, in line with the description given in Commandeur *et al.* (2011), the equations for the local level model are

$$\begin{aligned} y_t &= \mu_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2_\varepsilon), \\ \mu_{t+1} &= \mu_t + \xi_t, \quad \xi_t \sim \mathcal{N}(0, \sigma^2_\xi), \end{aligned} \tag{10}$$

Code for the model construction and estimation for the data y are as follows:

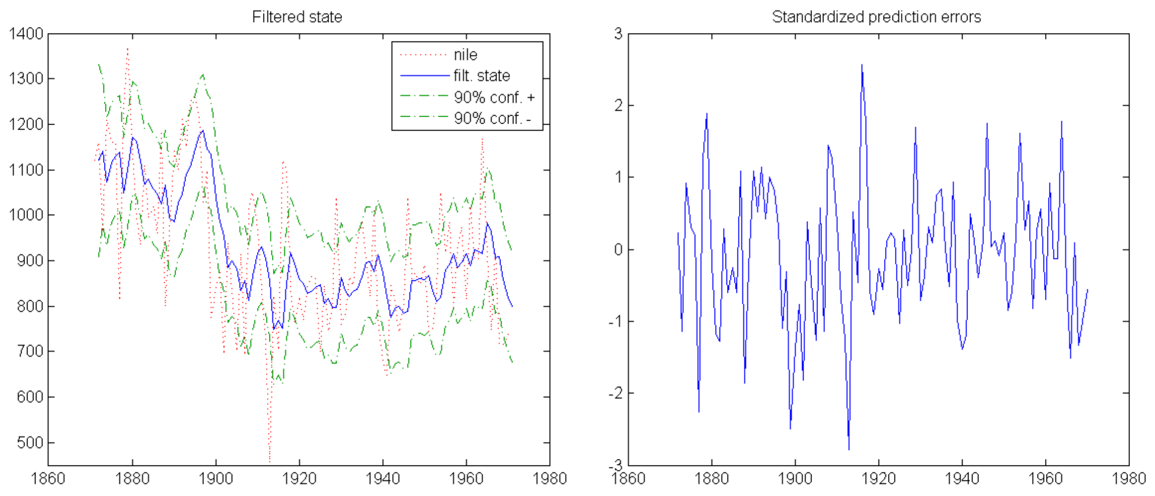

Figure 1: Nile data filtered state sequence and standardized prediction errors.

```
llm          = estimate(y, ssm_llm, [10000 5000]);
[logL fvar] = loglik(y, llm);
```

When the model is parameterized such that $\psi = \log([\sigma_\varepsilon \; \sigma_\xi])$, the maximum likelihood parameter estimates are $\psi = [4.8112 \; 3.6462]$ and asymptotic s.e. $[0.1041 \; 0.4354]$ respectively, calculated through numerical derivatives of the likelihood. This yields parameter estimates of $\sigma_\varepsilon^2 = 15098.4, \sigma_\xi^2 = 1469.1$, very close to those seen in Durbin and Koopman (2001). The following code obtains the filtered and smoothed state sequences and its 90% confidence interval, which are shown in Figures 1–2:

```
[a P v F] = kalman(y, llm);
aconf90 = [a+1.645*sqrt(P); a-1.645*sqrt(P)];
[alphahat V r N] = statesmo(y, llm);
alphaconf90 = [alphahat+1.645*sqrt(V); alphahat-1.645*sqrt(V)];
```

Figure 2 also shows the forecast and confidence interval obtained by the Kalman filter using the following code:

```
yforc = [y repmat(NaN, 1, 9)];
[aforc Pforc vforc Fforc] = kalman(yforc, llm);
```

where `repmat()` is a standard MATLAB function. The auxiliary residuals (Figure 3) can be calculated directly from the smoothed disturbances, which can be found using

```
[epshat etahat epsvarhat etavarhat] = disturbsmo(y, llm);
```

3. Univariate analysis

In this and the following example, data on road accidents in Great Britain (Harvey and Durbin 1986) is analyzed using structural time series models following Durbin and Koopman

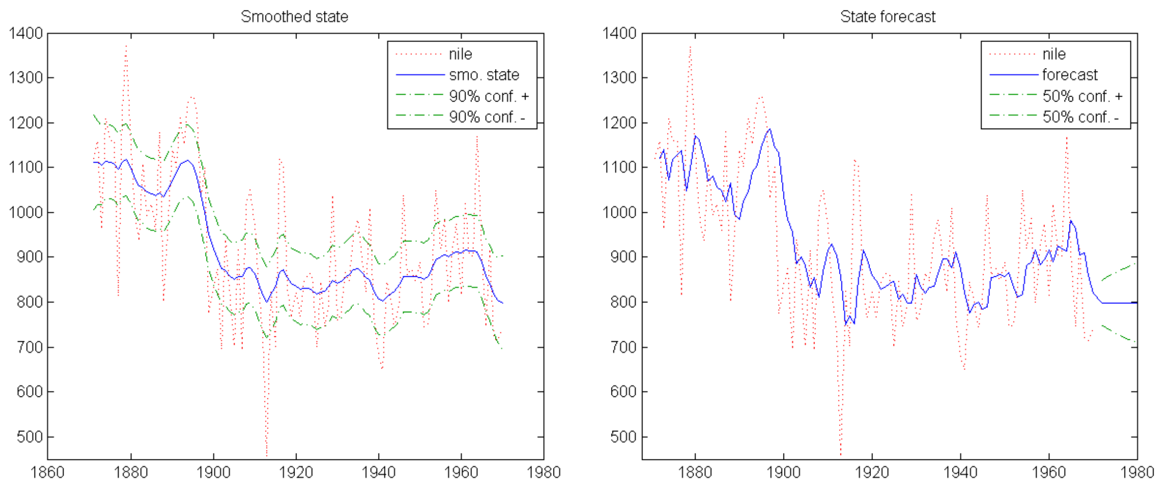

Figure 2: Nile data smoothed state sequence and Kalman filter forecast.

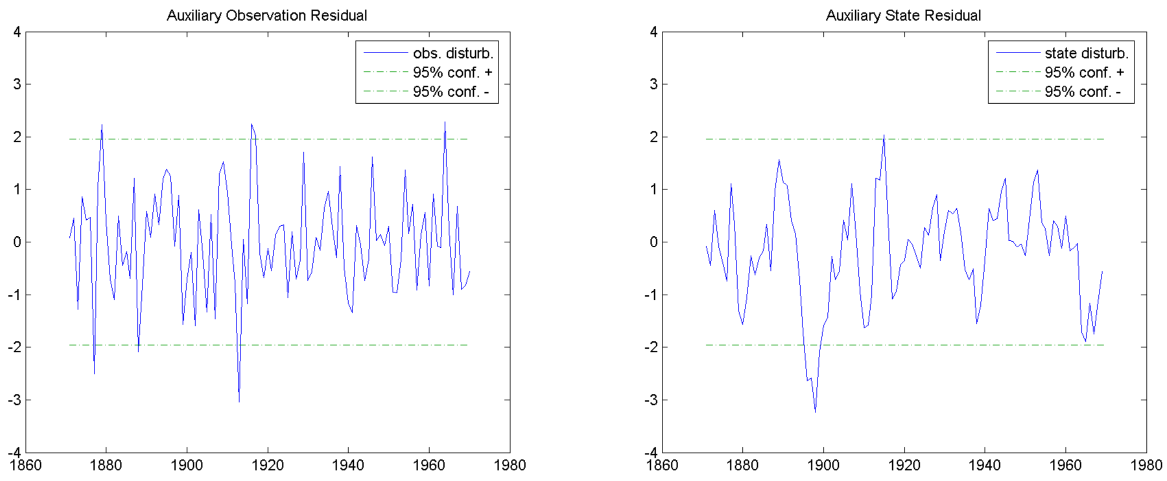

Figure 3: Nile data standardized auxiliary residuals.

(2001, Section 9.2). The purpose of the analysis is to assess the effect of seat belt laws on road accident casualties, with individual monthly figures for drivers, front seat passengers and rear seat passengers. The monthly price of petrol and average number of kilometers traveled will be used as regression variables. The data is from January 1969 to December 1984.

The drivers series will be analyzed with a univariate structural time series model, which consists of local level component μ_t, trigonometric seasonal component γ_t, regression component (on price of petrol) and intervention component (introduction of seat belt law) βx_t. The model equation is

$$y_t = \mu_t + \gamma_t + \beta x_t + \varepsilon_t, \tag{11}$$

where ε_t is the observation noise. The following code example constructs this model:

```
lvl = ssm_llm;
seas = ssm_seasonal('trig1', 12);
```

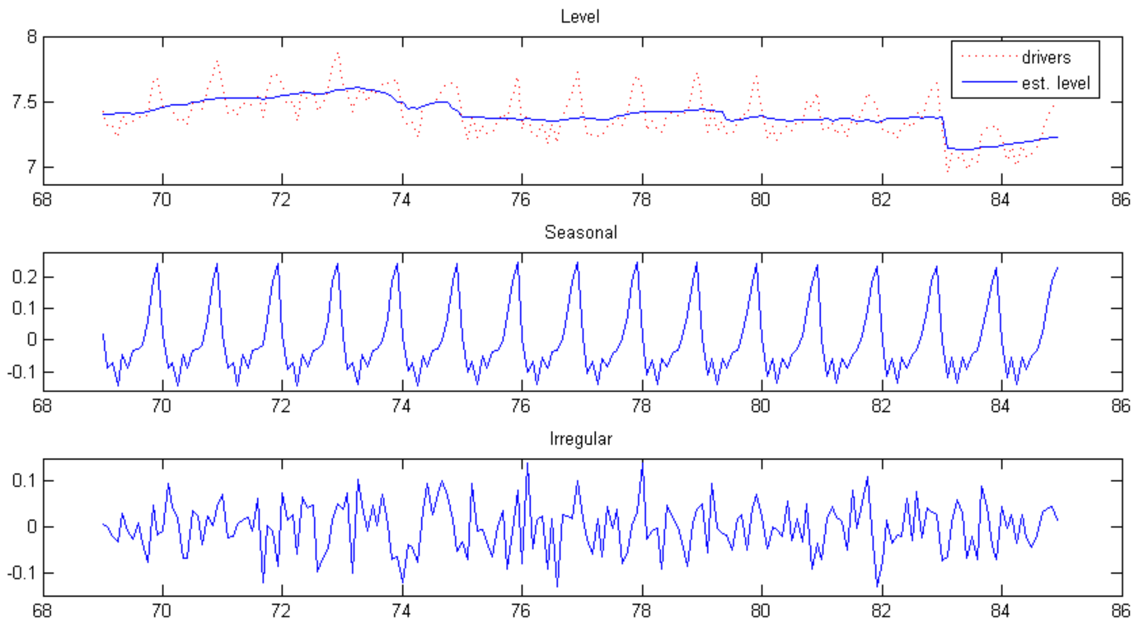

Figure 4: Driver casualties estimated by basic structural time series model.

```
intv = ssm_intv(n, 'step', 170);
reg = ssm_reg(petrol, 'price of petrol');
bstsm = [lvl seas intv reg];
bstsm.name = 'Basic structural time series model';
bstsm.param = [10 0.1 0.001];
```

With the model constructed, estimation can proceed with the code:

```
[bstsm logL] = estimate(y, bstsm);
[alphahat V] = statesmo(y, bstsm);
irr = disturbsmo(y, bstsm);
ycom = signal(alphahat, bstsm);
ylvl = sum(ycom([1 3 4], :), 1);
yseas = ycom(2, :);
```

The estimated model parameters are `[0.0037862 0.00026768 1.162e-006]`, which can be obtained by displaying `bstsm.param`, the loglikelihood is `175.7790`. State and disturbance smoothing is performed with the estimated model, and the smoothed state is transformed into signal components by `signal`. Because $\mathtt{p} = 1$, the output `ycom` is $\mathtt{M} \times \mathtt{n}$, where `M` is the number of signal components. The level, intervention and regression are summed as the total data level, separating seasonal influence. Using MATLAB graphic functions, the individual signal components and data are shown in Figure 4. The coefficients for intervention and regression are defined as part of the state vector in this model, so they can be obtained from the last two elements of the smoothed state vector (due to the order of component concatenation) at the last time point. The coefficient for the intervention is `alphahat(13, n) = -0.23773`, and the coefficient for the regression (price of petrol) is `alphahat(14, n) = -0.2914`. In this way diagnostics of these coefficient estimates can also be obtained by the smoothed state variance `V`.

4. Bivariate analysis

The front seat passenger and rear seat passenger series (`y2`) will be analyzed together using bivariate structural time series model following Durbin and Koopman (2001, Section 9.3), with components as before. Specifically, separate level and seasonal components are defined for both series, but the disturbances are assumed to be correlated. To reduce the number of parameters estimated the seasonal component is assumed to be fixed, so that the total number of parameters is six. We also include regression on the price of petrol (`petrol`) and kilometers traveled (`km`), and intervention for only the first series, since the seat belt law only effects the front seat passengers. The following is the model construction code:

```
bilvl = ssm_mvllm(2);
biseas = ssm_mvseasonal(2, [], 'trig fixed', 12);
biintv = ssm_mvintv(2, n, {'step' 'null'}, 170);
bireg = ssm_mvreg(2, [petrol; km]);
bistsm = [bilvl biseas biintv bireg];
bistsm.name = 'Bivariate structural time series model';
```

The model is then estimated with carefully chosen initial values, and state smoothing and signal extraction proceeds as before:

```
[bistsm logL] = estimate(y2, bistsm,
                [0.0054 0.0086 0.0045 0.00027 0.00024 0.00023]);
[alphahat V] = statesmo(y2, bistsm);
y2com = signal(alphahat, bistsm);
y2lvl = sum(y2com(:,:, [1 3 4]), 3);
y2seas = y2com(:,:, 2);
```

When `p` >1 the output from `signal` is of dimension `p` $\times$ `n` $\times$ `M`, where `M` is the number of components. The level, regression and intervention are treated as one component of data level as before, separated from the seasonal component. The components estimated for the two series is shown in Figure 5. The intervention coefficient for the front passenger series is obtained by `alphahat(25, n) = -0.30025`, the next four elements are the coefficients of the regression of the two series on the price of petrol and kilometers traveled.

5. ARMA models

In this example the difference of the number of users logged on an internet server (Makridakis *et al.* 1998) is analyzed by ARMA models (Commandeur *et al.* 2011, Section 4) following Durbin and Koopman (2001, Section 9.4), and model selection via BIC and missing data analysis are demonstrated. To select an appropriate ARMA(p, q) model for the data various values for p and q are tried, and the BIC of the estimated model for each is recorded, the model with the lowest BIC value is chosen.

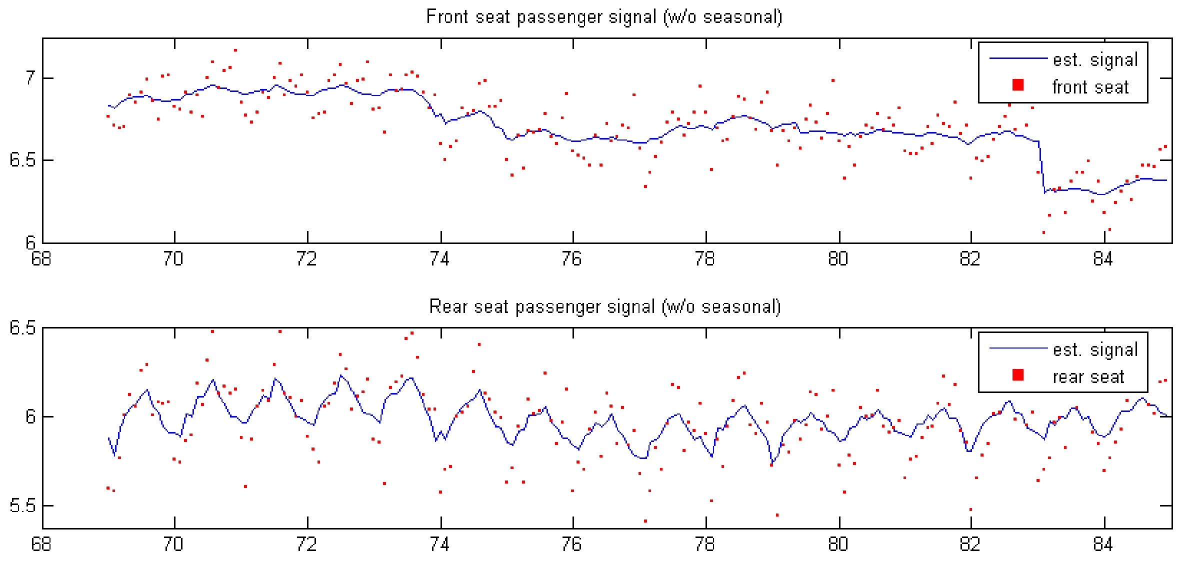

Figure 5: Passenger casualties estimated by bivariate structural time series model.

	$q = 0$	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$
$p = 0$	6.3999	5.6060	5.3299	5.3601	5.4189	5.3984
$p = 1$	5.3983	5.2736	5.3195	5.3288	5.3603	5.3985
$p = 2$	5.3532	5.3199	5.3629	5.3675	5.3970	5.4436
$p = 3$	5.2765	5.3224	5.3714	5.4166	5.4525	5.4909
$p = 4$	5.3223	5.3692	5.4142	5.4539	5.4805	5.4915
$p = 5$	5.3689	5.4124	5.4617	5.5288	5.5364	5.5871

Table 1: BIC values of ARMA(p, q) models for users logged on an internet server.

```
for p = 0 : 5, for q = 0 : 5
    [arma logL output] = estimate(y, ssm_arma(p, q), 0.1);
    BIC(p+1, q+1) = output.BIC;
end, end
[m i] = min(BIC(:));
arma = estimate(y, ssm_arma(mod(i-1, 6), floor((i-1)/6)), 0.1);
```

The BIC values obtained for each model are shown in Table 1. The model with the lowest BIC is ARMA$(1, 1)$, second lowest is ARMA$(3, 0)$, and the former model is therefore chosen for subsequent analysis.

Next missing data is simulated by setting some time points to `NaN`, model and state estimation can still proceed normally with missing data present.

```
y([6 16 26 36 46 56 66 72 73 74 75 76 86 96]) = NaN;
arma = estimate(y, arma, 0.1);
yf = signal(kalman(y, arma), arma);
```

Forecasting is equivalent to treating future data as missing, thus the data set `y` is appended

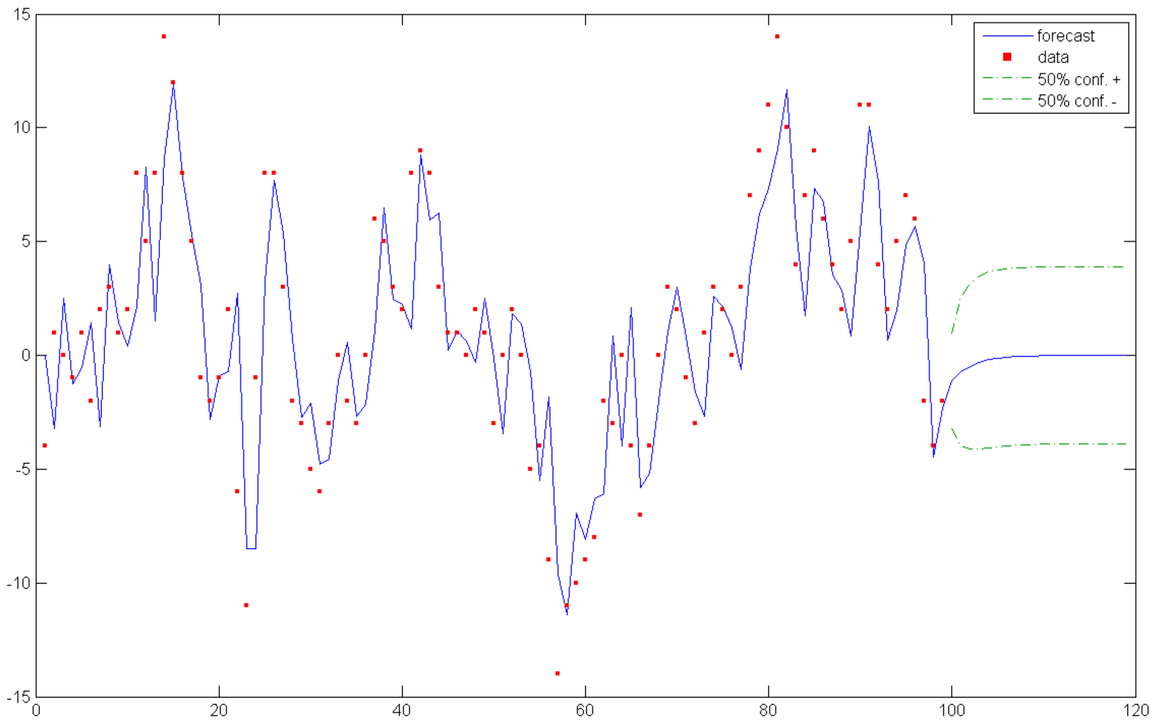

Figure 6: Forecast using ARMA(1, 1) model with 50% confidence interval.

with as many `NaN` values as the steps ahead to forecast. Using the previous estimated ARMA(1, 1) model the Kalman filter will then effectively predict future data points.

```
[a P v F] = kalman([y repmat(NaN, 1, 20)], arma);
yf = signal(a(:, 1:end-1), arma);
conf50 = 0.675*realsqrt(F); conf50(1:n) = NaN;
```

The plot of the forecast and its confidence interval is shown in Figure 6. Note that if the Kalman filter is replaced with the state smoother, the forecasted values will still be the same.

6. Cubic spline smoothing

The general state space formulation can also be used to do cubic spline smoothing (Durbin and Koopman 2001, Sections 3.11 and 9.5), by putting the cubic spline into an equivalent state space form, and accounting for the continuous nature of such smoothing procedures. The discrete state space representation of the cubic spline model is given by

$$\begin{aligned} y_t &= \alpha_t + \varepsilon_t, & \varepsilon_t &\sim \mathcal{N}(0, \sigma^2), \\ \alpha_{t+1} &= 2\alpha_t - \alpha_{t-1} + \zeta_t, & \zeta_t &\sim \mathcal{N}(0, \sigma^2/\lambda), \lambda > 0 \end{aligned} \tag{12}$$

Here the continuous acceleration data of a simulated motorcycle accident (Silverman 1985) is smoothed by the cubic spline model,

$$\begin{aligned} y(t_i) &= \mu(t_i) + \varepsilon_i, & \varepsilon_t &\sim \mathcal{N}(0, \sigma_\varepsilon^2), \quad i = 1, \ldots, n \\ \mu_t &= \mu(0) + v(0)t + \sigma_\zeta \int_0^t w(s)ds, & 0 &\le t \le T \end{aligned} \tag{13}$$

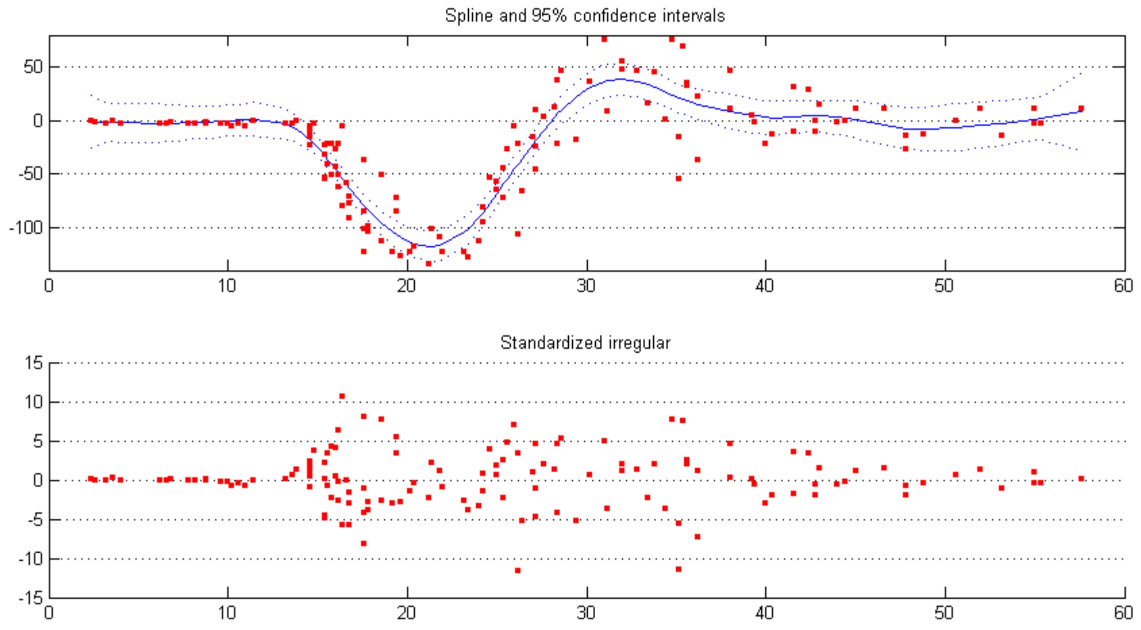

Figure 7: Cubic spline smoothing of motorcycle acceleration data.

and with $\lambda = \frac{\sigma_\varepsilon^2}{\sigma_\zeta^2}$. The continuous cubic spline model is predefined in the software:

```
spline = estimate(y, ssm_spline(delta), [1 0.1]);
[alphahat V] = statesmo(y, spline);
conf95 = squeeze(1.96*realsqrt(V(1, 1, :)))';
[eps eta epsvar] = disturbsmo(y, spline);
```

where `delta` is the distance between the observations y, not necessarily regularly spaced. The smoothed data and standardized irregular are plotted and shown in Figure 7.

It is seen from Figure 7 that the irregular may be heteroscedastic, an easy *ad hoc* solution is to model the changing variance of the irregular by a continuous version of the local level model. Assume the irregular variance is $\sigma_\varepsilon^2 h_t^2$ at time point t and $h_1 = 1$, then we model the absolute value of the smoothed irregular `abs(eps)` with h_t as its level. As defined in Harvey and Koopman (2000), the continuous local level model with level h_t needs to be constructed from scratch.[1]

```
contllm = ssmodel('', 'continuous local level', ...
  0, 1, 1, 1, ssmat(0, [], true, zeros(1, n), true), ...
  'Qd', {@(X) exp(2*X)*delta}, {[]}, ssparam({'zeta var'}, '1/2 log'));
contllm = [ssm_gaussian contllm];
alphahat = statesmo(abs(eps), estimate(abs(eps), contllm, [1 0.1]));
h2 = (alphahat/alphahat(1)).^2;
```

[1]The following code calls more advanced forms of the **SSM** class constructors and MATLAB anonymous functions, the interested reader can refer to the **SSM** Manual (Peng and Aston 2007) or the MATLAB documentation for more details.

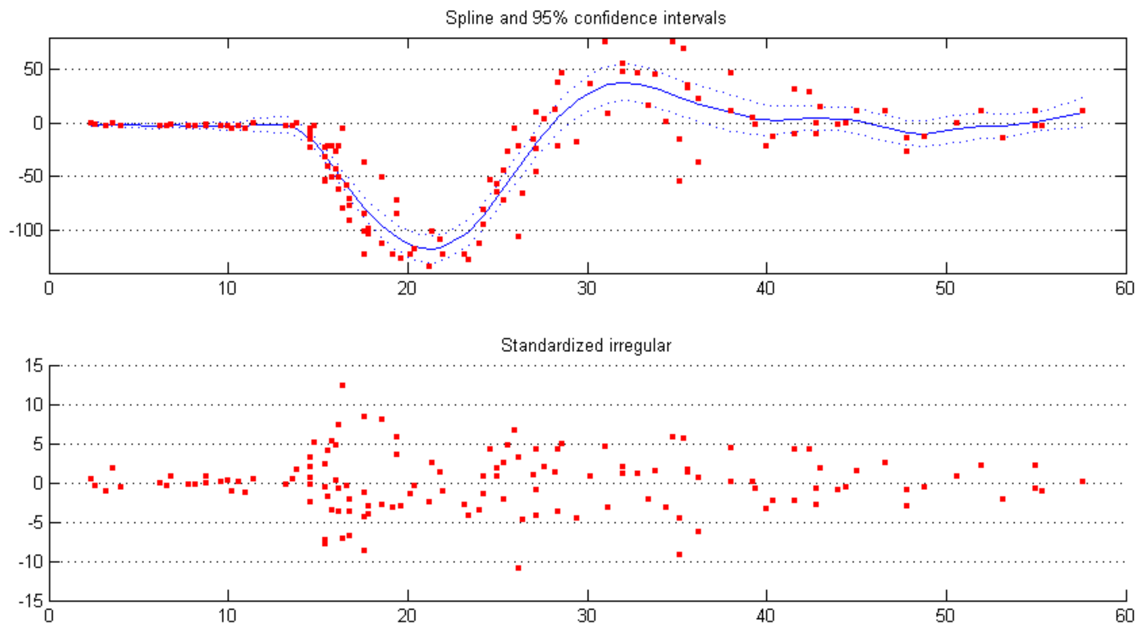

Figure 8: Cubic spline smoothing of motorcycle acceleration data with heteroscedastic noise.

`h2` is then the relative magnitude of the noise variance h_t^2 at each time point, which can be used to construct a custom dynamic observation noise model as follows:

```
hetnoise = ssmodel('', 'Heteroscedastic noise', ...
  ssmat(0, [], true, zeros(1, n), true), zeros(1, 0), [], [], [], ...
  'Hd', {@(X) exp(2*X)*h2}, {[]}, ssparam({'epsilon var'}, '1/2 log'));
hetspline = estimate(y, [hetnoise spline], [1 0.1]);
[alphahat V] = statesmo(y, hetspline);
conf95 = squeeze(1.96*realsqrt(V(1, 1, :)))';
[eps eta epsvar] = disturbsmo(y, hetspline);
```

The smoothed data and standardized irregular with heteroscedastic assumption is shown in Figure 8, where it is seen that the confidence interval shrank, especially at the start and end of the series, and the irregular is slightly more uniform.

In summary the motorcycle series is first modeled by a cubic spline model with Gaussian irregular assumption, then the smoothed irregular magnitude itself is modelled with a local level model. Using the irregular level estimated at each time point as the relative scale of irregular variance, a new heteroscedastic irregular continuous time model is constructed with the estimated level built-in, and plugged into the cubic spline model to obtain new estimates for the motorcycle series.

7. Hillmer-Tiao decomposition

As an example of extensions to the **SSM** object library, TRAMO model selection (Gómez and Maravall 2001a) and Hillmer-Tiao decomposition for ARIMA type models as used in SEATS

(Gómez and Maravall 2001b) are implemented. These ideas are widely used as part of the TRAMO/SEATS seasonal adjustment framework. The functions implemented here either take SSMODEL objects as arguments, or return SSMODEL objects.

In this example seasonal adjustment is performed by the Hillmer-Tiao decomposition (Hillmer and Tiao 1982) of airline models, a particular type of regComponent model (see Bell (2011) in this volume for a more in depth discussion of RegComponent models). Briefly, The ARIMA Model Based (AMB) decomposition for the airline model with seasonal period s, can be expressed in the following way using the decomposition of Hillmer and Tiao (1982)

$$\begin{aligned} U(B)S_t &= \theta_S(B)\omega_t \\ (1-B)^2T_t &= \theta_T(B)\eta_t \\ I_t &= \varepsilon_t \end{aligned}$$

where $U(B) = (1 + B + \ldots + B^{s-1})$ and $\omega_t, \eta_t, \epsilon_t$ are independent white noise processes and

$$y_t = S_t + T_t + I_t.$$

Here $By_t = y_{t-1}$, with S_t being the seasonal component, T_t the trend and I_t the irregular component, and $\theta_S(B)$ and $\theta_T B$ are seasonal and trend MA components, respectively. In order to define a unique solution (which may or may not exist) for $\theta_S(B)$ and $\theta_T(B)$, the ARIMA component parameters, in terms of the parameters of the original airline model,

$$(1-B)(1-B^s)y_t = (1-\theta B)(1-\Theta B^s)\varepsilon_t, \quad \varepsilon_t \sim N(0,\sigma^2) \tag{14}$$

with MA parameter θ, seasonal MA parameter Θ, and variance σ^2, the restriction is taken that the pseudo-spectral densities of the seasonal and trend components have a minimum of zero (in line with the admissible decompositions of Hillmer and Tiao 1982). When this condition cannot be met without resulting in a negative variance for I_t, the decomposition is said to be inadmissible.

The data (manufacturing and reproducing magnetic and optical media, US Census Bureau) is fitted with the airline model, then the estimated model is Hillmer-Tiao decomposed into an ARIMA components model with trend and seasonal components. The same is done for the Frequency Specific ARIMA model (Aston *et al.* 2007), also known as the generalized airline model, and the seasonal adjustment results are compared. The following are the code to perform seasonal adjustment with the airline model:

```
air = estimate(y, ssm_airline, 0.1);
aircom = ssmhtd(air);
ycom = signal(statesmo(y, aircom), aircom);
airseas = ycom(2, :);
```

`aircom` is the decomposed ARIMA components model corresponding to the estimated airline model, and `airseas` is the seasonal component, which will be subtracted out of the data `y` to obtain the seasonal adjusted series. `ssmhtd` automatically decompose ARIMA type models into trend, seasonal and irregular components, plus any extra MA components as permissible, which typically result from sARIMA models other than the airline model.

The same seasonal adjustment procedure is then done with the generalized airline model, using parameter estimates from the airline model as initial parameters:

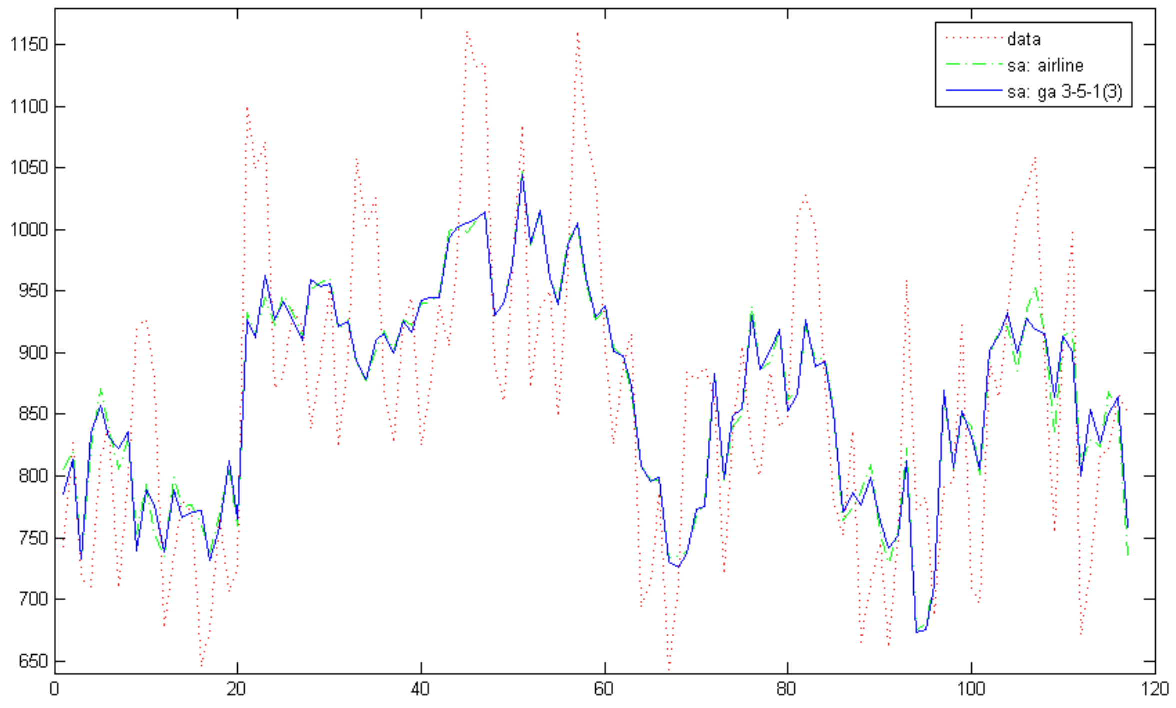

Figure 9: Seasonal adjustment with airline and generalized airline models.

```
param0 = air.param([1 2 2 3]);
param0(1:3) = -param0(1:3); param0(2:3) = param0(2:3).^(1/12);
ga = estimate(y, ssm_genair(3, 5, 3), param0);
gacom = ssmhtd(ga);
ycom = signal(statesmo(y, gacom), gacom);
gaseas = ycom(2, :);
```

The code creates a generalized airline (3-5-1) model, Hillmer-Tiao decomposition produces the same components as for the airline model since the two models have the same order. From the code it can be seen that the various functions work transparently across different ARIMA type models. Figure 9 shows the comparison between the two seasonal adjustment results.

8. Non-Gaussian linear models

8.1. Poisson distribution error models

The road accident casualties and seat belt law data analyzed in Sections 3 and 4 also contains a monthly van driver casualties series. Due to the smaller numbers of van driver casualties the Gaussian assumption is not justified in this case, previous methods cannot be applied. Here a poisson distribution is assumed for the data (Durbin and Koopman 2001, Section 14.2), the mean is $\exp(\theta_t)$ and the log density of the observation y_t is

$$\log p(y_t|\theta_t) = \theta_t^\top y_t - \exp(\theta_t) - \log y_t!$$

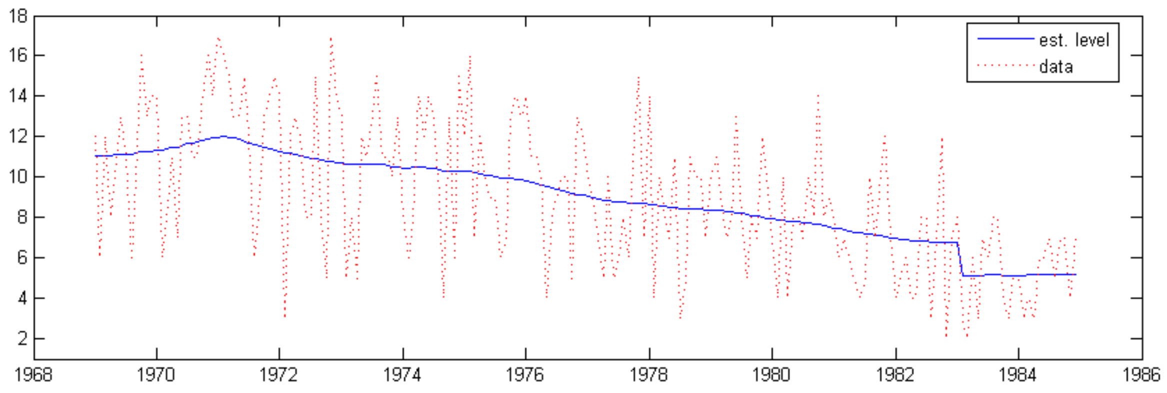

Figure 10: Van driver casualties and estimated level.

The signal θ_t, the log of the conditional mean, is modeled with a local level model. The total model is then constructed by concatenating a poisson distribution model with a standard structural time series model, the former model will replace the default Gaussian noise model.

```
pbstsm = [ssm_poisson ssm_llm ...
  ssm_seasonal('dummy fixed', 12) ssm_intv(n, 'step', 170)];
pbstsm.name = 'Poisson basic STSM';
```

Model estimation using the function `estimate` will automatically calculate the Gaussian approximation to the poisson model. Since this is an exponential family distribution the data y_t also need to be transformed to $\tilde{y}_t$, a time-varying transformation based on an appropriate approximating model (see Durbin and Koopman 2001, Table 11.1), which is stored in the output argument `output`, and used in place of y_t for all functions implementing linear Gaussian (Kalman filter related) algorithms. The following is the code for model and state estimation:

```
[pbstsm logL output] = estimate(y, pbstsm, 0.0006, [], 'fmin', 'bfgs',
  'disp', 'iter');
[alphahat V] = statesmo(output.ytilde, pbstsm);
thetacom = signal(alphahat, pbstsm);
```

Note that the original data `y` is input to the model estimation routine, which also calculates the transform `ytilde`. The model estimated then has its Gaussian approximation built-in, and will be treated by the state smoother as a linear Gaussian model, hence the transformed data `ytilde` needs to be used as input. The loglikelihood `logL` here is based on importance sampling using Gaussian approximation of the model, with 1000 samples (default setting). The state estimates `alphahat` are calculated from both the approximation and original models. The signal components `thetacom` obtained from the smoothed state `alphahat` is the components of θ_t, the mean of `y` can then be estimated by `exp(sum(thetacom, 1))`.

The exponentiated level $\exp(\theta_t)$ with the seasonal component eliminated is compared to the original data in Figure 10. The effect of the seat belt law can be clearly seen.

8.2. t distribution models

In this example another kind of non-Gaussian models, t distribution models is used to analyze

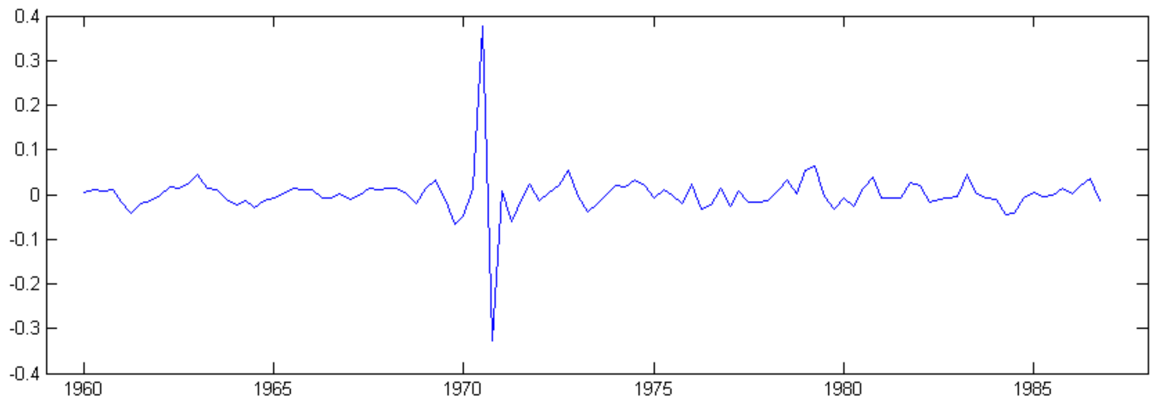

Figure 11: t distribution irregular.

quarterly demand for gas in the UK (data from the standard datasets in Koopman *et al.* 2000) following Durbin and Koopman (2001, Section 14.3). A structural time series model with a t distribution heavy-tailed observation noise is constructed, similar to the last section, and model estimation and state smoothing is performed.

```
tstsm = [ssm_t ssm_llt ssm_seasonal('dummy', 4)];
tstsm = estimate(y, tstsm, [0.0018 4 7.7e-10 7.9e-6 0.0033], [],
  'fmin', 'bfgs');
[alpha irr] = fastsmo(y, tstsm);
```

Since t distribution is not an exponential family distribution, data transformation is not necessary, and `y` is used throughout. A plot of the irregular in Figure 11 shows that potential outliers with respect to the Gaussian assumption have been detected by the use of heavy-tailed distribution.

9. Concluding remarks

SSM is an open source MATLAB toolbox for data analysis using state space methods. It combines the usability and flexibility of interactive model construction with the efficiency of C-implemented state space algorithms. In addition to the large number of predefined models available for immediate use, any models expressible in state space form, with appropriate approximations if they are nonlinear or non-Gaussian, can be defined and used within **SSM**, without having to write additional code for optimization, estimation or combination with other models. While extensive support for MCMC or other sampling based approximation techniques is not currently implemented beyond that of simulation smoothing, these could be combined using other MATLAB implementations of such methods.

By adapting open source licenses, it is hoped that the software package can attain wide-spread use (as of September 2010 well over 2000 downloads of the software have been made since **SSM** went online on October 2007), with user contribution of new models and even new state space algorithms. Possible future features under consideration include incorporation of particle filters to analyze more non-Gaussian or heteroscedastic model types, and incorporation of Markov switching for general regime or change-point analysis. On the software side, usability will continue to be improved, while it is also important to consider porting to a completely open source environment such as R or Python.

Software availability

The software is freely available from `http://sourceforge.net/projects/ssmodels/` along with detailed documentation and numerous demo files, including full code and instructions for performing the analysis of the examples given in the paper. The toolbox was written for MATLAB 7.0 R14 and later, and any C compiler compatible with MATLAB can be used.

10. Acknowledgments

Most of this work was done while the authors were employed by the Institute of Statistical Science, Academia Sinica. The authors would like to thank David Findley, Brian Monsell, Siem Jan Koopman, participants at the Joint Statistical Meeting 2006 in Seattle and many of the users of the software for very helpful comments and suggestions. This work was partially supported by the National Science Council (Taiwan), grant nos. 94-2118-M-001-014 and 95-2118-M-001-003, EPSRC grant EP/H016856/1 and the EPSRC /HEFCE CRiSM grant.

References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999). ***LAPACK** Users' Guide*. 3rd edition. Society for Industrial and Applied Mathematics, Philadelphia, PA. ISBN 0-89871-447-8.

Aston JAD, Findley DF, McElroy TS, Wills KC, Martin DEK (2007). "New ARIMA Models for Seasonal Time Series and their Application to Seasonal Adjustment and Forecasting." *Research Report Series, US Census Bureau.*

Bell WR (2011). "**REGCMPNT** – A Fortran Program for Regression Models with ARIMA Component Errors." *Journal of Statistical Software*, **41**(7), 1–23. URL `http://www.jstatsoft.org/v41/i07/`.

Commandeur JJF, Koopman SJ, Ooms M (2011). "Statistical Software for State Space Methods." *Journal of Statistical Software*, **41**(1), 1–18. URL `http://www.jstatsoft.org/v41/i01/`.

Durbin J, Koopman SJ (2001). *Time Series Analysis by State Space Methods*. Oxford University Press, Oxford.

Gómez V, Maravall A (2001a). "Automatic Modeling Methods for Univariate Series." In D Peña, GC Tiao, RS Tsay (eds.), *A Course in Time Series*, chapter 7. John Wiley & Sons, New York.

Gómez V, Maravall A (2001b). "Seasonal Adjustment and Signal Extraction in Economic Time Series." In D Peña, GC Tiao, RS Tsay (eds.), *A Course in Time Series*, chapter 8. John Wiley & Sons, New York.

Harvey AC (1989). *Forecasting, Structural Time Series Models and the Kalman Filter.* Cambridge University Press, Cambridge.

Harvey AC, Durbin J (1986). "The Effects of Seat Belt Legislation on British Road Casualties: A Case Study in Structural Time Series Modelling." *Journal of the Royal Statistical Society A*, **149**, 187–227.

Harvey AC, Koopman SJ (2000). "Signal Extraction and the Formulation of Unobserved Components Models." *Econometrics Journal*, **3**, 84–107.

Hillmer SC, Tiao GC (1982). "An ARIMA-Model-Based Approach to Seasonal Adjustment." *Journal of the American Statistical Association*, **77**, 63–70.

Koopman SJ, Harvey AC, Doornik JA, Shephard N (2000). ***STAMP** 6.0: Structural Time Series Analyser, Modeller and Predictor.* Timberlake Consultants, London.

Makridakis S, Wheelwright SC, Hyndman RJ (1998). *Forecasting: Methods and Applications.* 3rd edition. John Wiley & Sons, New York.

Peng JY, Aston JAD (2007). ***State Space Models** Manual.* URL http://sourceforge.net/projects/ssmodels/.

Silverman BW (1985). "Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting." *Journal of the Royal Statistical Society B*, **47**, 1–52.

The MathWorks, Inc (2007). *MATLAB – The Language of Technical Computing, Version 7.5.* The MathWorks, Inc., Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

A. Predefined model reference

The predefined models can be organized into two categories, observation disturbance models and normal models. The former contains only specification of the observation disturbance, and is used primarily for model combination; the latter contains all other models, and can in turn be partitioned into structural time series models, ARIMA type models, and other models.

The following is a list of each model and function name.

- *Observation disturbance models*
  - *Gaussian noise:* `ssm_gaussian([p, cov])` or `ssm_normal([p, cov])`
    `p` is the number of variables, default `1`.
    `cov` is true if they are correlated, default `true`.
  - *Null noise:* `ssm_null([p])`
    `p` is the number of variables, default `1`.
  - *Poisson error:* `ssm_poisson`
  - *Binary error:* `ssm_binary`
  - *Binomial error:* `ssm_binomial(k)`
    `k` is the number of trials, can be a scalar or row vector.
  - *Negative binomial error:* `ssm_negbinomial(k)`
    `k` is the number of trials, can be a scalar or row vector.
  - *Exponential error:* `ssm_exp`
  - *Multinomial error:* `ssm_multinomial(h, k)`
    `h` is the number of cells.
    `k` is the number of trials, can be a scalar or row vector.
  - *Exponential family error:* `ssm_expfamily(b, d2b, id2bdb, c)`

    $$p(y|\theta) = \exp\left(y^\top\theta - b(\theta) + c(y)\right)$$

    `b` is the function $b(\theta)$.
    `d2b` is $\ddot{b}(\theta)$, the second derivative of $b(\theta)$.
    `id2bdb` is $\ddot{b}(\theta)^{-1}\dot{b}(\theta)$.
    `c` is the function $c(y)$.
  - *t-distribution noise:* `ssm_t([nu])`
    `nu` is the degree of freedom, will be estimated as model parameter if not specified.
  - *Gaussian mixture noise:* `ssm_mix`
  - *General error noise:* `ssm_err`
- *Structural time series models*
  - *Integrated random walk:* `ssm_irw(d)`
    `d` is the order of integration.
  - *Local polynomial trend:* `ssm_lpt(d)`
    `d` is the order of the polynomial.

- *Local level model:* `ssm_llm`
- *Local level trend:* `ssm_llt`
- *Seasonal components:* `ssm_seasonal(type, s)`
  `type` can be `'dummy'`, `'dummy fixed'`, `'h&s'`, `'trig1'`, `'trig2'` or `'trig fixed'`.
  `s` is the seasonal period.
- *Cycle component:* `ssm_cycle`
- *Regression components:* `ssm_reg(x[, varname])`
- *Dynamic regression components:* `ssm_dynreg(x[, varname])`
  `x` is a $m \times n$ matrix, m is the number of regression variables.
  `varname` is the name of the variables.
- *Intervention components:* `ssm_intv(n, type, tau)`
  `n` is the total time duration.
  `type` can be `'step'`, `'pulse'`, `'slope'` or `'null'`.
  `tau` is the onset time.
- *Constant components:* `ssm_const`
- *Trading day variables:* `ssm_td(y, m, N, td6)`
  `'td6'` is set to true to use six variables.
- *Length-of-month variables:* `ssm_lom(y, m, N)`
- *Leap-year variables:* `ssm_ly(y, m, N)`
- *Easter effect variables:* `ssm_ee(y, m, N, d)`
  `y` is the starting year.
  `m` is the starting month.
  `N` is the total number of months.
  `d` is the number of days before Easter.
- *Structural time series models:* `ssm_stsm(lvl, seas, s[, cycle, x])`
  `lvl` is `'level'` or `'trend'`.
  `seas` is the seasonal type (see seasonal components).
  `s` is the seasonal period.
  `cycle` is `true` if there's a cycle component in the model, default `false`.
  `x` is explanatory (regression) variables (see regression components).
- *Common levels models:* `ssm_commonlvls(p, A_ast, a_ast)`

$$\begin{aligned} y_t &= \begin{bmatrix} 0 \\ a^* \end{bmatrix} + \begin{bmatrix} I_r \\ A^* \end{bmatrix} \mu_t^* + \varepsilon_t \\ \mu_{t+1}^* &= \mu_t^* + \eta_t^* \end{aligned}$$

  `p` is the number of variables (length of y_t).
  `A_ast` is A^*, a $(p-r) \times r$ matrix.
  `a_ast` is a^*, a $(p-r) \times 1$ vector.
- *Multivariate local level models:* `ssm_mvllm(p[, cov])`
- *Multivariate local level trend:* `ssm_mvllt(p[, cov])`
- *Multivariate seasonal components:* `ssm_mvseasonal(p, cov, type, s)`

  - *Multivariate cycle component:* `ssm_mvcycle(p[, cov])`
    `p` is the number of variables.
    `cov` is a logical vector that is true for each correlated disturbances, default all `true`.
    Other arguments are the same as the univariate versions.
  - *Multivariate regression components:* `ssm_mvreg(p, x[, dep])`
    `dep` is a $p \times m$ logical matrix that specifies dependence of data variables on regression variables.
  - *Multivariate intervention components:* `ssm_mvintv(p, n, type, tau)`
  - *Multivariate structural time series models:* `ssm_mvstsm(p, cov, lvl, seas, s[, cycle, x, dep])`
    `cov` is a logical vector that specifies the covariance structure of each component in turn.

- *ARIMA type models*
  - *ARMA models:* `ssm_arma(p, q, mean)`
  - *ARIMA models:* `ssm_arima(p, d, q, mean)`
  - *Multiplicative seasonal ARIMA models:* `ssm_sarima(p, d, q, P, D, Q, s, mean)`
  - *Seasonal sum ARMA models:* `ssm_sumarma(p, q, D, s, mean)`
  - *SARIMA with Hillmer-Tiao decomposition:* `ssm_sarimahtd(p, d, q, P, D, Q, s, gauss)`
    `p` is the order of AR.
    `d` is the order of I.
    `q` is the order of MA.
    `P` is the order of seasonal AR.
    `D` is the order of seasonal I.
    `Q` is the order of seasonal MA.
    `s` is the seasonal period.
    `mean` is true if the model has mean.
    `gauss` is true if the irregular component is Gaussian.
  - *Airline models:* `ssm_airline([s])`
    `s` is the period, default 12.
  - *Frequency specific SARIMA models:* `ssm_freqspec(p, d, q, P, D, nparam, nfreq, freq)`
  - *Frequency specific airline models:* `ssm_genair(nparam, nfreq, freq)`
    `nparam` is the number of parameters, 3 or 4.
    `nfreq` is the size of the largest subset of frequencies sharing the same parameter.
    `freq` is an array containing the members of the smallest subset.
  - *ARIMA component models:* `ssm_arimacom(d, D, s, phi, theta, ksivar)`
    The arguments match those of the function `htd`, see its description for details.

- *Other models*
  - *Cubic spline smoothing (continuous time):* `ssm_spline(delta)`
    `delta` is the time duration of each data point.

– *1/f noise models (approximated by AR):* `ssm_oneoverf(m)`
  `m` is the order of the approximating AR process.

B. Data analysis functions reference

Most functions in this section accepts analysis settings options, specified as option name and option value pairs (e.g., `('disp', 'off')`). These groups of arguments are specified at the end of each function that accepts them, and are represented by `opt` in this section.

- `batchsmo`
  `[alphahat epshat etahat] = BATCHSMO(y, model[, opt])` performs batch smoothing of multiple data sets. `y` is the data of dimension $\texttt{p} \times \texttt{n} \times \texttt{N}$, where `n` is the data length and `N` is the number of data sets, there must be no missing values. `model` is a `SSMODEL`. The output is respectively the smoothed state, smoothed observation disturbance and smoothed state disturbance, each of dimensions $\texttt{m} \times \texttt{n} \times \texttt{N}$, $\texttt{p} \times \texttt{n} \times \texttt{N}$ and $\texttt{r} \times \texttt{n} \times \texttt{N}$. This is equivalent to doing `fastsmo` on each data set.

- `disturbsmo`
  `[epshat etahat epsvarhat etavarhat] = DISTURBSMO(y, model[, opt])` performs disturbance smoothing. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. The output is respectively the smoothed observation disturbance ($\texttt{p} \times \texttt{n}$), smoothed state disturbance ($\texttt{r} \times \texttt{n}$), smoothed observation disturbance variance ($\texttt{p} \times \texttt{p} \times \texttt{n}$ or $1 \times \texttt{n}$ if $\texttt{p} = 1$) and smoothed state disturbance variance ($\texttt{r} \times \texttt{r} \times \texttt{n}$ or $1 \times \texttt{n}$ if $\texttt{r} = 1$).

- `estimate`
  `[model logL output] = ESTIMATE(y, model[, param0, alpha0, opt])` estimates the parameters of `model` starting from the initial parameter value `param0`. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. `param0` can be empty if the current parameter values of `model` is used as initial value, and a scalar `param0` sets all parameters to the same value. Alternatively `param0` can be a logical row vector specifying which parameters to estimate, or a $2 \times \texttt{w}$ matrix with the first row as initial value and second row as estimated parameter mask. The initial state sequence estimate `alpha0` is needed only when `model` is non-Gaussian or nonlinear. `output` is a structure that contains optimization routine information, approximated observation sequence $\tilde{y}$ if non-Gaussian or nonlinear, and the AIC and BIC of the output `model`.

- `fastsmo`
  `[alphahat epshat etahat] = fastsmo(y, model[, opt])` performs fast smoothing. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. The output is respectively the smoothed state ($\texttt{m} \times \texttt{n}$), smoothed observation disturbance ($\texttt{p} \times \texttt{n}$), and smoothed state disturbance ($\texttt{r} \times \texttt{n}$).

- `gauss`
  `[model ytilde] = GAUSS(y, model[, alpha0, opt])` calculates the Gaussian approximation. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. `alpha0` is the initial state sequence estimate and can be empty or omitted.

- `kalman`
  `[a P v F] = KALMAN(y, model[, opt])` performs Kalman filtering. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. The output is respectively the filtered state ($\texttt{m} \times \texttt{n+1}$), filtered state variance ($\texttt{m} \times \texttt{m} \times \texttt{n+1}$, or $1 \times \texttt{n+1}$ if $\texttt{m} = 1$), one-step prediction error (innovation) ($\texttt{p} \times \texttt{n}$), one-step prediction variance ($\texttt{p} \times \texttt{p} \times \texttt{n}$, or $1 \times \texttt{n}$ if $\texttt{p} = 1$).

- `linear`
  `[model ytilde] = LINEAR(y, model[, alpha0, opt])` calculates the linear approximation. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. `alpha0` is the initial state sequence estimate and can be empty or omitted.

- `loglik`
  `LOGLIK(y, model[, ytilde, opt])` returns the log likelihood of `model` given `y`. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. `ytilde` is the approximating observation $\tilde{y}$ and is needed for some non-Gaussian or nonlinear models.

- `sample`
  `[y alpha eps eta] = SAMPLE(model, n[, N])` generates observation samples from `model`. `model` is a `SSMODEL`, `n` specifies the sampling data length, and `N` specifies how many sets of data to generate. `y` is the sampled data of dimension $\texttt{p} \times \texttt{n} \times \texttt{N}$, `alpha`, `eps`, `eta` are respectively the corresponding sampled state ($\texttt{m} \times \texttt{n} \times \texttt{N}$), observation disturbance ($\texttt{p} \times \texttt{n} \times \texttt{N}$), and state disturbance ($\texttt{r} \times \texttt{n} \times \texttt{N}$).

- `signal`
  `SIGNAL(alpha, model)` generates the signal for each component according to the state sequence and model specification. `alpha` is the state of dimension $\texttt{m} \times \texttt{n}$, and `model` is a `SSMODEL`. The output is a cell array of data each with dimension $\texttt{p} \times \texttt{n}$, or a $\texttt{M} \times \texttt{n}$ matrix where `M` is the number of components if $\texttt{p} = 1$.

- `simsmo`
  `[alphatilde epstilde etatilde] = SIMSMO(y, model, N[, antithetic, opt])` generates observation samples from `model` conditional on data `y`. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. `antithetic` should be set to `1` if antithetic variables are used. The output is respectively the sampled state sequence ($\texttt{m} \times \texttt{n} \times \texttt{N}$), sampled observation disturbance ($\texttt{p} \times \texttt{n} \times \texttt{N}$), and sampled state disturbance ($\texttt{r} \times \texttt{n} \times \texttt{N}$).

- `statesmo`
  `[alphahat V] = STATESMO(y, model[, opt])` performs state smoothing. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, and `model` is a `SSMODEL`. The output is respectively the smoothed state ($\texttt{m} \times \texttt{n}$), smoothed state variance ($\texttt{m} \times \texttt{m} \times \texttt{n}$, or $1 \times \texttt{n}$ if $\texttt{p} = 1$). If only the first output argument is specified, fast state smoothing is automatically performed instead.

- `oosforecast`
  `[yf err SS] = OOSFORECAST(y, model, n1, h)` performs out-of-sample forecast. `y` is the data of dimension $\texttt{p} \times \texttt{n}$, `model` is a `SSMODEL`, `n1` is the number of time points to exclude at the end, and `h` is the number of steps ahead to forecast, which can be an array. The output `yf` is the forecast obtained, `err` is the forecast error, and `SS` is the forecast error cumulative sum of squares.

Affiliation:

John A. D. Aston
Centre for Research in Statistical Methodology
Department of Statistics
University of Warwick
Coventry, CV4 7AL, United Kingdom
E-mail: `J.A.D.Aston@warwick.ac.uk`

Journal of Statistical Software `http://www.jstatsoft.org/`
published by the American Statistical Association `http://www.amstat.org/`

Volume 41, Issue 6 *Submitted:* 2010-01-18
May 2011 *Accepted:* 2010-11-17