Illustrations of state space modelling using SsfPack

Bouk van Geloven and Siem Jan Koopman

Department of Econometrics, VU University
1081 HV Amsterdam, The Netherlands

SsfPack: `http://www.ssfpack.com`
Ox: `http://www.doornik.com/ox`

May 2009

1 Introduction

State space modelling provides a unified methodology for treating a wide range of problems in time series analysis. This paper demonstrates, via applications, how to put linear models into state space form, the relationship between the state space model and the ARIMA model, the Kalman filter, various smoothing methods and maximum likelihood estimation of state space models, by using *SsfPack* of Koopman et al. (1999). The examples are taken from Tsay (2005, Chapter 11) and are implemented in *Ox* (version 2.0 or later) matrix programming language of Doornik (1998). This paper can be seen as a supplement to Koopman et al. (1999), as it contains more examples of the *SsfPack* commands used to perform state space analysis. We discuss the programs extensively, however for further details of the commands used, see Durbin and Koopman (2001), or for a discussion about *SsfPack* see Koopman et al. (1999).

All examples presented here are in the form of *Ox* code and originate from Tsay (2005, Chapter 11). As we only discuss the examples and not the models we refer to Tsay (2005) for details about the models or Durbin and Koopman (2001) for a wider coverage of the literature of state space modelling in time series.

We begin by introducing the state space form and the *SsfPack* notation in Section 2. Section 3 discusses the state space formulation of selected linear models. Applications of the recursive algorithms associated with the Kalman filter are given in Section 4. We conclude in Section 5.

2 State space form

The state space form provides a unified representation of a wide range of linear Gaussian time series models including ARMA models, time-varying regression models, dynamic linear models and unobserved components time series models, see, for example, Durbin and Koopman (2001). This framework also encapsulates different specifications for non-parametric and spline regressions. The Gaussian state space form consists of a transition equation and a measurement equation; we formulate it as

$$\begin{aligned} \alpha_{t+1} &= d_t + T_t\alpha_t + H_t\varepsilon_t, \quad \alpha_1 \overset{d}{=} \mathrm{N}(a, P), \quad t = 1, ..., n & (1)\\ \theta_t &= c_t + Z_t\alpha_t, & (2)\\ y_t &= \theta_t + G_t\varepsilon_t, \quad \varepsilon_t \overset{d}{=} \mathrm{NID}(0, I), & (3) \end{aligned}$$

where $\mathrm{NID}(\mu, \Psi)$ indicates an independent sequence of normally distributed random vectors with mean μ and variance matrix Ψ, and, similarly, $\mathrm{N}(\cdot, \cdot)$ a normally distributed variable. The N observations at time t are placed in the vector y_t and the $N \times n$ data matrix is given by $(y_1, ..., y_n)$. The $m \times 1$ state vector α_t contains unobserved stochastic processes and unknown fixed effects. The initial state vector is assumed to be random with mean a and variance matrix P. The deterministic matrices T_t, Z_t, H_t and G_t are referred to as system matrices and they are usually sparse selection matrices. The vectors d_t and

c_t are fixed, and can be useful to incorporate known effects or known patterns into the model, otherwise they are zero. When the system matrices are constant over time, we drop the time indices to obtain the matrices T, Z, H and G. The resulting state space form is referred to as time invariant. For a more complete discussion about the Gaussian state space form see Durbin and Koopman (2001, Chapter 4).

2.1 The state space representation in *SsfPack*

The state space form in *SsfPack* is represented by

$$\begin{aligned}
&\begin{pmatrix}\alpha_{t+1}\\ y_t\end{pmatrix} = \delta_t + \Phi_t\alpha_t + u_t, \quad u_t \overset{d}{=} \mathrm{NID}(0,\Omega_t), \quad t = 1,...,n \\
&\delta_t = \begin{pmatrix}d_t\\ c_t\end{pmatrix}, \quad \Phi_t = \begin{pmatrix}T_t\\ Z_t\end{pmatrix}, \quad u_t = \begin{pmatrix}H_t\\ G_t\end{pmatrix}\varepsilon_t, \quad \Omega_t = \begin{pmatrix}H_tH_t' & H_tG_t'\\ G_tH_t' & G_tG_t'\end{pmatrix}, \\
&\alpha_1 \overset{d}{=} \mathrm{N}(a,P).
\end{aligned} \tag{4}$$

The vector δ_t is $(m+N)\times m$ and Ω_t =$(m+N)\times(m+N)$. For an overview of the appropriate dimension see Table 1.

α_{t+1}, d_t, a	$m \times 1$	T_t, P	$m \times m$
y_t, θ_t, c_t	$N \times 1$	H_t	$m \times r$
δ	$(m+N) \times 1$	Z_t	$N \times m$
ε_t	$r \times 1$	G_t	$N \times r$
Φ	$(m+N) \times m$	Σ	$(m+1) \times m$
Ω	$(m+N) \times (m+N)$		
m	dimension of state vector		
N	number of variables		
n	number of observations		
r	dimension of disturbance vector		

Table 1. Dimensions of state space matrices.

Specifying a model in state space form within *SsfPack* can be done in different ways depending on its complexity. At the most elementary level, the state space form is time invariant with $\delta = 0,\ a = 0$ and $P = \kappa I$ where κ is some pre-set constant. For this elementary case only two matrices are required, that is

$$\Phi = \begin{pmatrix}T\\ Z\end{pmatrix}, \quad \Omega = \begin{pmatrix}HH' & HG'\\ GH' & GG'\end{pmatrix}.$$

To illustrate, consider the local level model

$$\mu_{t+1} = \mu_t + \eta_t, \quad \eta_t \overset{d}{=} \mathrm{NID}(0,\sigma_\eta^2), \tag{5}$$

$$y_t = \mu_t + e_t, \quad e_t \overset{d}{=} \mathrm{NID}(0,\sigma_e^2), \tag{6}$$

with $\mu_1 \overset{d}{=}\mathrm{N}(0,\kappa)$ where κ is large. The state vector only contains the trend component μ_t and can easily be extended. The matrices Φ and Ω for model (5) and (6) are given by

$$\Phi = \begin{pmatrix}1\\ 1\end{pmatrix}, \quad \Omega = \begin{pmatrix}\sigma_\eta^2 & 0\\ 0 & \sigma_e^2\end{pmatrix}.$$

In *Ox* code, when $\sigma_\eta^2 = 1.1$ and $\sigma_e^2 = 0.5$, these matrices can be created as follows

```
mPhi = <1;1>;
mOmega = diag(<1.1,0.5>;
```

2.2 Data sets used in the illustrations

Ssfpack expects all data variables to be in *row* vectors. Various data formats can be loaded easily in *Ox*, such as Excel and PcGive files. Many examples start with a statement such as

```
s_mY=loadmat("aa-3rv.xls")[][1]';
```

which creates `s_mY` as an $1 \times n$ matrix, where the second column of the file *aa-3rv.xls* is selected, which contains the daily realized volatility series of Alcoa stock 10-minute returns. This series was originally considered by Tsay (2005).

The data sets used in this paper can be found in Table 2.

aa-3rv.xls	Daily realized volatility series of Alcoa stock from January 2, 2003 to May 7, 2004
m-fac9003.xls	Monthly returns of GM stock and S&P 500 index from January 1990 to December 2003
q-jnj.xls	Quarterly earnings of Johnson and Johnson from January 1960 to December 1980
m-pfesp-ex9003.xls	Monthly returns of Pfizer stock and S&P 500 index from January 1990 to December 2003
m-ppiaco.xls	Monthly producer price index from January 1921 to August 2008

Table 2. Data sets used in the illustrations.

The series are available on `www.ssfpack.com` and on the website of the book from Tsay (2005). For a complete discussion about the datasets we refer to this book.

2.3 Initial conditions

To specify the initial state condition in *Ssfpack* explicitly, the matrix

$$\Sigma = \begin{pmatrix} P \\ a' \end{pmatrix},$$

is required. The block matrix P in Σ is equal to matrix P in (4) except when a diagonal element of P is equal to -1, indicating that the corresponding initial state vector element is diffuse. When the initial state conditions are not explicitly defined, it will be assumed that the state vector is fully diffuse. In certain circumstances the user may wish to specify P freely, in *Ox* code, this matrix can be created as follows

```
mOmega = diag(<1.1,0;0,2.3;0,0>;
```

2.4 Time-varying state space form

When some elements of the system matrices are not constant but change over time, additional administration is required in *SsfPack.* Basically, it requires two additional variables, first a data matrix X which contains the time-varying values and second an index matrix for Φ, Ω or δ that identifies the time-varying variables from X. The elements of the index matrices are all set to -1 except the elements for which the corresponding elements in Φ, Ω or δ are time varying[1]. The non-negative index value indicates the row of X which contains the time-varying values. The notation and name used in *SsfPack/Ox* for time-varying state space models are the following.

Index / data matrix	*SsfPack/Ox*
J_Φ	`mJPhi`
J_Ω	`mJOmega`
J_δ	`mJDelta`
X	`mXt`

Table 3. Notation in SsfPack for time varying state space models

[1] Note: The index of the first row or column, for *Ox* programming language, is *zero*, unlike other programming languages which start at one.

2.5 Formulating the state space in *Ssfpack*

A time-invariant state space form can be inputted in one of the following three formats, depending on the model at hand

```
mPhi, mOmega
mPhi, mOmega, mSigma
mPhi, mOmega, mSigma, mDelta
```

A state space form with time-varying system elements requires the index matrices J_Φ, J_Ω and J_δ, together with a data matrix X to which the indices refer. Therefore the fourth possible formulation is

```
mPhi, mOmega, mSigma, mDelta, mJPhi, mJOmega, mJDelta, mXt
```

where `mXt` is the data matrix with n columns as discussed in Section 2.4. For a complete discussion about the *Ssfpack* representation of state space models see Koopman et al. (1999).

3 Putting Linear Models in state-space form

Many dynamic time series models in economics and finance can be represented in state space form. It would be tedious if we had to construct the system matrices of the state space form (4) manually for every model. Therefore, *Ssfpack* provides functions to create these matrices for several commonly used models. This section documents, via applications, those functions. However, the system matrices may still be constructed or modified manually, even after using the provided routines.

In many applications, the system matrices are time-invariant. However, these matrices can be time-varying, making the state space model flexible. In the following we consider the examples of Tsay (2005) in Section 11.3.1 to 11.3.5.

3.1 CAPM with Time-Varying Coefficients

We start off with considering the capital asset pricing model (CAPM) with time-varying intercept and slope, see Tsay (2005, equation (11.27)). In this model Φ_t contains $r_{M,t}$, which is time-varying. Some special input is required to specify such a model, see Section 2.4. The *SsfPack* routine `GetSsfReg` provides the time-varying state space structure for univariate regression models. The function call is

```
GetSsfReg(mXt, &mPhi, &mOmega, &mSigma, &mJPhi);
```

where `mXt` is a data matrix containing the explanatory variables. This matrix is only used to identify the number of regressors to be included in the model. The remaining four arguments are used to receive the system matrices Φ, Ω, Σ and J_Φ. The `&` is used to pass a reference to the variable, which is changed on return.

To illustrate we consider the monthly simple excess returns of General Moters stock from January 1990 to December 2003, see *m-fac9003.xls*. The monthly simple excess return of the S&P 500 composite index is used as the market return. The specification of a time-varying CAPM requires values of the variances σ_η^2, σ_ε^2 and σ_e^2. We suppose $(\sigma_\eta,\ \sigma_\varepsilon,\ \sigma_e) = (0.02,\ 0.04,\ 0.1)$. The state space specification for the CAPM is given in Listing 1.

```
#include <oxstd.h>
#include <packages/ssfpack/ssfpack.h>

main()
{
    decl mXt, mPhi, mOmega, mSigma, mJPhi;
    mXt = 1|loadmat("m-fac9003.xls")[][13]'/100;   //S&P500 as benchmark in CAPM model
                                                   //Data in percentages
    GetSsfReg(mXt,&mPhi, &mOmega, &mSigma, &mJPhi);//Get state space form
    mOmega=diag(<0.02, 0.04, 0.1>).^2;             //Values for sigma_eta, sigma_eps and sigma_e
```

```
    print("Phi", mPhi,"Omega", mOmega,"Sigma", mSigma, "JPhi", mJPhi);
    print("mXt: numerical matrix: 168 rows, 2 columns");
    print("%13g","%c", {"Constant","S&P"},"%r", {"row 1 "} , mXt[][0]');
    print("%13g","...");
    print("%13g","%r", {"row 168"} , mXt[][columns(mXt)-1]');
}
```

$$\Phi = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad J_\Phi = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0.0004 & 0 & 0 \\ 0 & 0.0016 & 0 \\ 0 & 0 & 0.01 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}$$

```
mXt: numerical matrix: 168 rows, 2 columns
                Constant         S&P
row 1                  1     -0.0752
...
row 168                1        0.05
```

Listing 1: CAPM.ox with output.

As this is the first complete program, we discuss it in some detail. The first line includes the standard *Ox* library. The second line includes the *SsfPack* header file, required to use the package (this assumes that *SsfPack* is installed in `ox/packages/ssfpack`). Every *Ox* program must have a `main()` function, which is where program execution commences. Variables are declared using the `decl` statement (variables must always be declared). The expression inside <> is a matrix constant. Such a constant may not contain variables; if that is required use horizontal '~' and vertical '|' concatenation to construct the matrix, for example: `var = phi1 ~phi2 ~phi3;`. The `print()` statements simple prints the desired result. In most examples below we only list the salient contents of `main()`. Then the `include` statements, `main()`, variable declarations and the `print()` statements must be added to create an *Ox* program which can be run.

3.2 ARMA Models

The autoregressive moving average model of order p and q, denoted by ARMA(p, q), is for example given in Tsay (2005, equation (11.28)). There are many ways to transform such an ARMA model into a state space form. Tsay (2005, section 11.3.2) discusses three methods available in the literature, Akaike's, Harvey's and Aoki's approach. The *SsfPack* routine `GetSsfArma` provides the appropriate system matrices for any univariate ARMA model. Harvey's approach is used. The routine requires two vectors containing the autoregressive parameters $\phi_1, ..., \phi_p$ and the moving average parameters $\theta_1, ..., \theta_q$ which must be chosen in such a way that the implied ARMA model is stationary and invertible, *SsfPack* does not verify this. The function call

```
GetSsfArma(vAr, vMa, dStDev, &mPhi, &mOmega, &mSigma);
```

places the ARMA coefficients within the appropriate state elements and it solves the set of linear equations for the variance matrix of the initial state vector. The arguments `vAr` and `vMa`, should be either row vectors or column vectors. The scalar value `dStDev` represents the standard deviation of the ARMA process. The remaining three arguments are used to receive the system matrices Φ, Ω and Σ.

In addition to this *SsfPack* routine, which uses Harvey's approach, both Akaike's and Aoki's approach are implemented in our *Ox* example ARMA.ox. Both approaches have the following function call

```
GetSsfArmaAkaike(vAr, vMa, dStDev, &mPhi, &mOmega, &mSigma);
GetSsfArmaAoki(vAr, vMa, dStDev, &mPhi, &mOmega, &mSigma);
```

which receives the system matrices Φ, Ω and Σ. Note however that the arguments `vAr` and `vMa`, should be row vectors and not column vectors. Since we do not want to get into detail about the algorithm of these approaches, which are stated in Tsay (2005, section 11.3.2), we only list the function call in our example. For programming details about the implementation of both approaches we refer to the program which can be found on the *SsfPack* website.

To illustrate, we consider the ARMA(2,1) model

$$y_t = 1.2y_{t-1} - 0.35y_{t-2} + \eta_t - 0.25\eta_{t-1}, \quad \eta_t \overset{d}{=} N(0, 1.1^2),$$

the state space form for all three approaches is given in Listing 2.

```
main()
{
    vAr = <1.2;-0.35>;
    vMa = <-0.25>;
    dStDev = 1.1;
                                            //Harvey's (1993, Section 4.4) approach ssf ARMA model
    GetSsfArma(vAr, vMa, dStDev, &mPhi, &mOmega, &mSigma);
    println("Harvey's (1993, Section 4.4) approach state-space form:");
    print("Phi =", mPhi, "Omega =", mOmega, "Sigma =", mSigma);
                                            //Akaike's (1975) approach ssf ARMA model
    GetSsfArmaAkaike(vAr, vMa, dStDev, &mPhi, &mOmega, &mSigma);
    println("Akaike's (1975) approach state-space form ARMA model:");
    print("Phi =", mPhi, "Omega =", mOmega, "Sigma =", mSigma);
                                            //Aoki's (1987, Chapter 4) approach ssf ARMA model
    GetSsfArmaAoki(vAr, vMa, dStDev, &mPhi, &mOmega, &mSigma);
    println("Aoki's (1987, Chapter 4) approach state-space form ARMA model:");
    print("Phi =", mPhi, "Omega =", mOmega, "Sigma =", mSigma);
}
```

```
Harvey's (1993, Section 4.4) approach state-space form:
```

$$\Phi = \begin{pmatrix} 1.2 & 1 \\ -0.35 & 1 \\ 1 & 0 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 1.21 & -0.3025 & 0 \\ -0.3025 & 0.075625 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 4.0607 & -1.4874 \\ -1.4874 & 0.57306 \\ 0 & 0 \end{pmatrix}$$

```
Akaike's (1975) approach state-space form ARMA model:
```

$$\Phi = \begin{pmatrix} 0 & 1 \\ -0.35 & 1.2 \\ 1 & 0 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 1.21 & 1.7545 & 0 \\ 1.7545 & 2.544 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 9.9008 & 9.0248 \\ 9.0248 & 8.6908 \\ 0 & 0 \end{pmatrix}$$

```
Aoki's (1987, Chapter 4) approach state-space form ARMA model:
```

$$\Phi = \begin{pmatrix} 0 & 1 \\ -0.35 & 1.2 \\ 0.25 & 1 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1.21 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 6.5701 & 5.8401 \\ 5.8401 & 6.5701 \\ 0 & 0 \end{pmatrix}$$

Listing 2: Part of ARMA.ox with corresponding output.

3.3 Linear Regression model

As stated in Section 3.1, univariate linear regression models can be represented in state space form, see Tsay (2005, section 11.3.3) how this can be done. The *SsfPack* routine `GetSsfReg` provides the time-varying state space structure for a univariate regression model. The function returns the composite matrices Φ, Ω and Σ, as well as the index matrix J_Φ.

To illustrate, we consider the simple market model

$$r_t = \beta_0 + \beta_1 r_{M,t} + e_t, \quad t = 1, ..., 168,$$

where r_t is the return of an asset and $r_{M,t}$ is the market return, in our example the S&P 500 composite index return. The state space specification for the regression model is given in Listing 3.

```
    GetSsfReg(mXt,&mPhi, &mOmega, &mSigma, &mJPhi);//State space regression model
```

$$\Phi = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad J_\Phi = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}$$

```
mX: numerical matrix: 168 rows, 2 columns
                Constant        S&P
row 1                1     -0.0752
...
row 168              1       0.05
```

Listing 3: Part of Regr.ox with corresponding output.

3.4 Linear Regression Models with ARMA errors

Consider a regression model with ARMA(p, q) errors, see for example Tsay (2005, equation (11.45)). Such a model can also be represented in state space form, here we slightly deviate from Tsay (2005, Section 11.3.4). Consider we have a time-invariant Gaussian state space form as given in equations (1)-(3). Now *SsfPack* provides the function `AddSsfReg` to include regressors to a time-invariant model. The function call is

```
AddSsfReg(mXt, &mPhi, &mOmega, &mSigma, &mJPhi);
```

where `mXt` is the $k \times n$ matrix of regressors, it is only used to identify the number of regressors to be included in the model. The returned matrices Φ, Ω and Σ are adjusted such that

$$\Phi = \begin{pmatrix} I_k & 0 \\ 0 & T \\ 0 & Z \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0 & HH' & HG' \\ 0 & GH' & GG' \\ 0 & 0 & 0 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -I_k & 0 \\ 0 & P \\ 0 & a' \end{pmatrix}$$

where k is the number of rows in the data matrix `mXt`. The matrices T, Z, H, G, a and P are obtained from the inputted matrices `mPhi, mOmega` and `mSigma`. The returned index matrix `mJPhi` is

$$J_\Phi = \begin{pmatrix} -I_k & -I \\ -I & -I \\ i & -I \end{pmatrix}$$

where i is a $1 \times k$ vector $(0, 1, ..., k-1)$.

To illustrate, we consider the model

$$\begin{aligned} r_t &= \beta_0 + \beta_1 r_{M,t} + z_t, \quad t = 1, ..., 168, \\ z_t &= 1.2 z_{t-1} - 0.35 z_{t-2} + \eta_t - 0.25 \eta_{t-1}, \quad \eta_t \stackrel{d}{=} N(0, \sigma_\eta^2). \end{aligned}$$

where r_t is the return of an asset and $r_{M,t}$ is the market return, in our example the S&P 500 composite index return. We use the following notation to denote the $n \times 2$ matrix of regressors $(1, r_{M,t})$. First the *SsfPack* routine `GetSsfArma` provides the appropriate system matrices for the univariate ARMA(2,1) model. Harvey's approach is used. To include regressors to the state space specification the routine described above is used, the example in Listing 4 outputs the relevant state space matrices.

```
vAr = <1.2;-0.35>;                          //ARMA specification
vMa = <-0.25>;
dStDev = 1;
                                            //Harvey's ssf ARMA model including regressors
GetSsfArma(vAr, vMa, dStDev, &mPhi, &mOmega, &mSigma);
AddSsfReg(mXt,&mPhi, &mOmega, &mSigma, &mJPhi);
```

$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1.2 & 1 \\ 0 & 0 & -0.35 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad J_\Phi = \begin{pmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 0 & 1 & -1 & -1 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -0.25 & 0 \\ 0 & 0 & -0.25 & 0.0625 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\Sigma = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 3.3560 & -1.2293 \\ 0 & 0 & -1.2293 & 0.47360 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
mX: numerical matrix: 168 rows, 2 columns
                Constant        S&P
row 1                1     -0.0752
...
row 168              1       0.05
```

Listing 4: Part of RegrARMA.ox with corresponding output.

3.5 Scalar Unobserved Component Model

The state space model also deals directly with unobserved components time series models used in structural time series and dynamic linear models. Ideally, such component models should be constructed from subject matter considerations, tailored to the particular problem at hand. However, in practice there are a group of commonly used components which are used extensively. The basic univariate unobserved component model, or the structural time series model (STSM), assumes four components in the specification, an irregular, trend, seasonal and cycle. See Tsay (2005, Section 11.3.5) or Koopman et al. (1999, section 3.2) for the model specification.

The *SsfPack* routine `GetSsfStsm` provides the relevant system matrices for any univariate structural time series model. The function call is

```
GetSsfStsm(mStsm, &mPhi, &mOmega, &mSigma);
```

where the input matrix `mStsm` contains the model information as provided in Table 4.

	Arguments			
	predefined constant	STSM parameters		
`mStsm = <`	`CMP_LEVEL,`	σ_η,	0	0;
	`CMP_SLOPE,`	σ_ζ,	0	0;
	`CMP_SEAS_DUMMY,`	σ_ω	s	0;
	`CMP_SEAS_TRIG,`	σ_ω	s	0;
	`CMP_SEAS_HS,`	σ_ω	s	0;
	`CMP_CYC_0,`	σ_ψ	λ_c	ρ;
	⋮	⋮	⋮	⋮
	`CMP_CYC_9,`	σ_ψ	λ_c	ρ;
	`CMP_IRREG,`	σ_ξ	0	0 >;

Table 4. Arguments in SsfPack for the command `GetSsfStsm`.

The input matrix may contain fewer rows than the above setup and the rows may have a different sequential order. However, the resulting state vector is organized in the sequence level, slope, seasonal, cycle and irregular. The first column of `mStsm` uses predefined constants, and the remaining columns contain real values. The function `GetSsfStsm` returns the three system matrices Φ, Ω and Σ in a similar fashion to `GetSsfArma` (Section 3.2).

To illustrate, consider the local level model in equations (5) and (6) with $\sigma_e = 0.4$ and $\sigma_\eta = 0.2$. The code in Listings 5 outputs the relevant state space matrices for the local level model.

```
GetSsfStsm(<CMP_IRREG, 0.4, 0, 0;   //State space structural time series model
           CMP_LEVEL, 0.2, 0, 0;>, &mPhi, &mOmega, &mSigma);
```

$$\Phi = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0.04 & 0 \\ 0 & 0.16 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

Listing 5: Part of STSM.ox with corresponding output.

4 Illustrations of the use of the linear Gaussian state space model

In this Section we illustrate some applications of the state space model in finance and business. Our objectives are to highlight the applicability of the model and to demonstrate the practical implementation of the analysis in *Ox* with *SsfPack*. To illustrate the ideas of the spate space model and Kalman filter we start with considering the intradaily realized volatility of Alcoa. Next we will consider the CAPM and the time-varying CAPM for the monthly simple excess returns of General Moters (GM). In the third example we analyze the series of quarterly earnings per share of Johnson and Johnson from 1960 to 1980 using the unobserved component model, see Tsay (2005) for details of the data. We finish with some illustrations using the same methods, but different datasets. All examples originate from Tsay (2005, example 11.1 - 11.3 and exercise 11.2, 11.3 and 11.5).

4.1 Illustration state space model and Kalman filter

We consider the intradaily realized volatility of Alcoa stock from January 2, 2003 to May 7, 2004 for 340 observations, see *aa-3rv.xls*. The daily realized volatility used is the sum of squares of intraday 10-minute log returns measured in percentage. No overnight returns or the first 10-minute intraday returns are used. The series used in the demonstration is the logarithm of the daily realized volatility. A time plot of the logarithms of the realized volatility of Alcoa stock from January 2, 2003 to May 7, 2004 can be found in the output of Listings 8, upper left figure. This example consist of three parts, the maximum likelihood estimation of an ARIMA(0,1,1), estimation of local level model and the Kalman filter, prediction error and state smoothing, respectively.

4.1.1 Maximum likelihood estimation of ARIMA(0,1,1)

We start our analyses with the obtained ARIMA(0,1,1) model

$$(1 - B)y_t = (1 - 0.858B)a_t, \quad \widehat{\sigma}_a = 0.5177. \tag{7}$$

where y_t is the log realized volatility and the standard error of $\widehat{\theta}$ is 0.0396, compare with equation (11.4) in Tsay (2005, Chapter 11). The residuals show $Q(12) = 12.4$ with p-value 0.41, indicating that there is no significant serial correlation in the residuals. Similarly, the squared residuals give $Q(12) = 8.2$ with p-value 0.77, suggesting no ARCH effects in the series. For a definition of this test statistic see Tsay (2005, page 27). These results agree with the results of Tsay (2005, page 492 and 493).

The following *Ox* code (Listing 6) applies the ARIMA(0,1,1) maximum likelihood estimation and model checking. Here we make use of the *Arfima* package of Doornik and Ooms (2006a) to estimate the ARIMA(0,1,1) model.

The first line imports the *Arfima* package, required to use the package (this assumes that *Arfima* package is installed in `ox/packages/arfima`). The second line imports a programs which performs the Ljung and Box (1978) test for serial correlation, see Tsay (2005, page 27) for a definition of this test statistic. The function call `ACtest(vResid, iLag)` returns the test statistic and p-value. For details how to import programs in *Ox* see Doornik and Ooms (2006b, Section 5.8).

This example does not have a `main()` function, but is a function. The call for this function, from the `main()` function, is `ArmaEstimation(iAR, iMA)`. For details how to create functions in *Ox* see Doornik and Ooms (2006a). As mentioned before we estimate the ARIMA(0,1,1) model with the *Arfima* package,

as we do not want to get into details about this package we refer to Doornik and Ooms (2006a) for details. In this example the appended variable `s_mdY` is the log differenced intradaily realized volatility of Alcoa stock which is loaded in the `main()` function. In the program `s_mdY` is a static variable, that is why it does not has to be an argument in the function `ArmaEstimation(iAR, iMA)`. After appending this variable to the *Arfima* database, we perform estimation and model checking. The results of the estimation are printed automatically.

```
#import <packages/arfima/arfima>
#import "ACtest"

ArmaEstimation(const iAR, const iMA)
{
    decl arfima, vResid;

    arfima = new Arfima();                  // create an object of class Arfima
    arfima.Append(s_mdY, "logdAlcoa", 0);   // store in database (first differences)
    arfima.Select(Y_VAR,{"logdAlcoa",0,0}); // from lag 0 to lag 0 (i.e. current only)
    arfima.ARMA(iAR,iMA);                   // specify an ARMA(iAR,d,iMA) model
    arfima.SetMethod(M_MAXLIK);             // estimate by exact ML
    arfima.FixD(0);                         // Fix d at 0
    arfima.Estimate();                      // estimate, automatically prints the results
                                            // Model checking
    vResid = arfima.GetResiduals();         // get residuals
    print("Test for Autocorrelation: Ljung and Box(1978) on residuals and squared residuals
        with lag 12:", "%cf", {"%12.5g", "(%7.5f)"}, ACtest(vResid, 12) | ACtest(vResid.^2, 12)
        );

    delete arfima;                          // done with arfima: delete the object
}
```

```
---- Maximum likelihood estimation of ARFIMA(0,0,1) model ----
The estimation sample is: 1 - 340
The dependent variable is: logdAlcoa
The dataset is:

                Coefficient Std.Error t-value t-prob
MA-1              -0.858364   0.03960   -21.7  0.000

log-likelihood  -259.238677
no. of observations    340   no. of parameters        2
AIC.T            522.477354  AIC               1.5366981
mean(logdAlcoa) 3.61763e-005 var(logdAlcoa)    0.429874
sigma              0.517661  sigma^2            0.267973

BFGS using numerical derivatives (eps1=0.0001; eps2=0.005):
Strong convergence
Used starting values:
     -0.66433

Test for Autocorrelation: Ljung and Box(1978) on residuals and squared residuals with lag 12:
      12.448(0.41037)
      8.1336(0.77461)
```

Listing 6: Part of Example11.1.ox with corresponding output.

4.1.2 Maximum likelihood estimation of local level model

We can, as described in Tsay (2005, Section 11.1), transform the ARIMA(0,1,1) model into a local level model of equations (5) and (6). Details of estimation are discussed in Tsay (2005, Section 11.1.7) or Koopman et al. (1999, Section 5.1). The example in Listing 7 performs the estimation of the local level model. The estimation is performed by calling `MaxLik();` from the `main()` function. The likelihood,

see Tsay (2005, equation (11.23)), can be maximized numerically using the `MaxBFGS` routine from *Ox*, see Doornik (1998, page 243). There are two parameters to estimate

$$\varphi = \{\sigma_\eta, \sigma_e\}'.$$

`MaxBFGS` works with φ, so we need to map this into the state space formulation. In Listing 7 this is done in the function `Likelihood`. This function updates the state space definition, puts the structural time series model in state space and finally returns the log likelihood function. `MaxBFGS` accepts a function as its first argument, but requires it to be in a specific format, which is called `Likelihood` here. We prefer to maximize $\frac{l}{n}$ rather then l, to avoid dependency on n in the convergence criteria. A starting value for $\log(\sigma_\eta)$ is chosen as follows

1. first evaluate the likelihood with $\sigma_\eta = 1$,
2. next, use `s_dVar` as returned by `SsfLik` for the initial value of σ_η^2.

Upon convergence, the coefficient standard errors are computed using numerical second derivatives. The maximum likelihood estimates of the two parameters are $\sigma_\eta^2 = 0.0735$ and $\sigma_\varepsilon^2 = 0.4803$. The measurement errors have a larger variance than the state innovations, confirming that intraday high-frequency returns are subject to measurement errors.

```
Likelihood(const vP, const pdLik, const pvSco, const pmHes)
{                                       // arguments dictated by MaxBFGS()
    s_mSsf[0:1][1] = exp(vP);           // update ssf definition
    GetSsfStsm(s_mSsf, &s_mPhi, &s_mOmega, // puts structural time series model in state space
    &s_mSigma);

    SsfLik(pdLik, &s_dVar, s_mY, s_mPhi,    // returns log-likelihood function
    s_mOmega, s_mSigma);
    pdLik[0] /= s_cT;                   // log-likelihood scaled by n
    return 1;                           // 1 = success, 0 failure
}
MaxLik()
{
    decl vp, ir, dlik;
    s_mSsf = <CMP_LEVEL, 1, 0, 0;
              CMP_IRREG, 1, 0, 0>;      // set state space definition matrix
    vp = log(<1;1>);                    // starting values log(sigma)
    Likelihood(vp, &dlik, 0, 0);        // evaluate lik at start val
    vp += 0.5 * log(s_dVar);            // scale starting values
    //MaxControl(10, 5, 1);
    ir = MaxBFGS(Likelihood,&vp,&dlik,0,1); // max lik estimation with analytical scores

    print("---- Maximum likelihood estimation of two parameters of LL model ----");
    print("\n", MaxConvergenceMsg(ir),      // printing results state space estimation
          " using analytical derivatives",
          "\nLog-likelihood = ", dlik * s_cT,
          "; n = ", s_cT);
    print("%13g","%c",{"Sigma eta", "Sigma e"}
          , "%r", {"Estimates"}, exp(vp)');
    print("Phi", s_mPhi,"Omega", s_mOmega,"Sigma", s_mSigma);
}
```

```
---- Maximum likelihood estimation of two parameters of LL model ----
```

$$\begin{matrix}\widehat{\sigma}_\eta = 0.0735 \\ \widehat{\sigma}_e = 0.4803\end{matrix}, \quad \Phi = \begin{pmatrix}1\\1\end{pmatrix}, \quad \Omega = \begin{pmatrix}0.0054036 & 0\\ 0 & 0.23064\end{pmatrix}, \quad \Sigma = \begin{pmatrix}-1\\0\end{pmatrix}$$

```
Strong convergence using analytical derivatives
Log-likelihood = -258.975; n = 340
```

Listing 7: Part of Example11.1.ox with corresponding output.

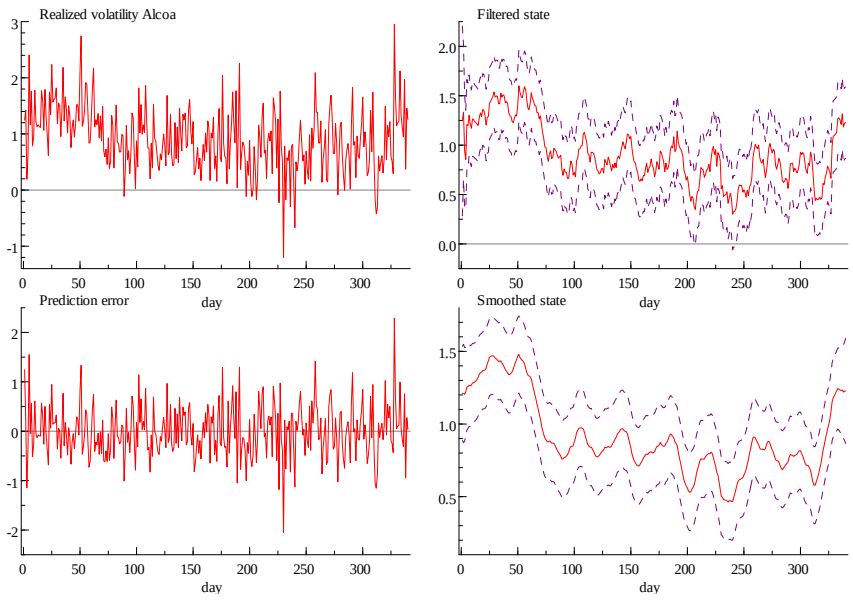
4.1.3 Kalman filter, prediction error and state smoothing

To illustrate application of the Kalman filter and state-smoothing, we use the fitted state space model for daily realized volatility of Alcoa stock return and apply the Kalman filter and state-smoothing algorithms to the data, see Tsay (2005, equation (11.12), (11.18) and (11.21)) for the algorithms. We perform our analysis with diffuse initialization, see Section 2.3 for a discussion about initialization of *SsfPack*. Our example produces the filtered state, the smoothed state both with its 95% pointwise confidence interval and the one-step ahead forecast errors. The example is reproduced in Listings 8.

The *SsfPack* function `KalmanFil` and `SsfMomentEst` are used to obtain the prediction errors, filtered state smoothing state, respectively. For more details about the *SsfPack* implementation of the functions see Koopman et al. (1999).

```
mKF = KalmanFil(s_mY, s_mPhi, s_mOmega, // returns output of Kalman filter
s_mSigma);
SsfMomentEst(ST_FIL,&vMu,s_mY,s_mPhi,   // Creating filtered state variable
s_mOmega,s_mSigma);
mstate = SsfMomentEst(ST_SMO,&mks,s_mY, // Smoothed state vector
s_mPhi,s_mOmega,s_mSigma);

DrawTMatrix(0,s_mY,1,1,1,1,0,2);        // Plot data
DrawTMatrix(1, vMu[0][],1,1,1,1,0,2);
DrawZ(sqrt(vMu[2][]), "", ZMODE_BAND, 2.0, 8);
DrawTMatrix(2, mKF[0][],1,1,1,1,0,2);
DrawTMatrix(3, mks[0][],{""}, 1, 1, 1,0,2);
DrawZ(sqrt(mks[2][]), "", ZMODE_BAND, 2.0, 8);
ShowDrawWindow();
```



Listing 8: Part of Example11.1.ox with corresponding output.

The filtered states are smoother compared with the time plot of the data. The forecast errors appear to be stable and center around zero. These forecast errors are out-of-sample 1-step ahead prediction errors. As expected, the smoothed state variables are smoother than the filtered state variables. The confidence intervals for the smoothed state variables are also narrower than those of the filtered state variables. Note that the width of the 95% confidence interval of the first observation depends on the initial value. All what is note yet covered, is the model checking of the fitted local level model. The implementation speaks for itself and we refer to the complete program and program output for details, which can be found on the *SsfPack* website.

4.2 CAPM vs time-varying CAPM

Next we consider the CAPM and the time-varying CAPM for the monthly simple excess returns of General Motors (GM) stock from January 1990 to December 2003. We use the simple excess returns of the S&P500 composite index as the market returns. In the following we will consider the estimation of the CAPM by the ordinary least squares (OLS) method and by using *SsfPack*. Furthermore, we estimate the time-varying CAPM model of Section 3.1.

4.2.1 CAPM estimation by OLS

We start our illustration with a simple market model

$$r_t = \alpha + \beta r_{M,t} + e_t, \quad e_t \stackrel{d}{=} \mathrm{N}(0, \sigma_e^2) \tag{8}$$

for $t = 1, ..., 168$. This is a fixed-coefficient model and can easily be estimated by the ordinary least squares (OLS) method. The implementation of the model is straightforward and not presented here, for details we refer to the program which can be found on the *SsfPack* website.

```
---- Summary OLS estimation of CAPM model ----
Coefficients:
                   Value  Std. Error    t-value     P value
Intercept      0.0019820   0.0063021    0.31450     0.75353
S&P              1.0457     0.14531     7.1962 1.9625e-011
Regression Diagnostics:
R-Squared            0.23778
Adjusted R-Squared   0.23319
Durbin-Watson Stat    2.0290
Residual Diagonostics:
               Statistic    P-Value
Jarque-Bera       2.5348    0.28156
Ljung-Box(25)     26.651    0.37355

Residual standard error: 0.0808158
```

Listing 9: Part of output of Example11.2.ox.

Thus, the fitted model is

$$r_t = 0.02 + 1.0457 r_{M,t} + e_t, \quad \widehat{\sigma}_e = 0.0808.$$

Based on the residual diagnostics, the model appears to be adequate for the GM stock returns with adjusted $R^2 = 23.3\%$.

4.2.2 Maximum likelihood estimation CAPM and tv-CAPM

As shown in Tsay (2005, Section 11.3), model (8) is a special case of the state space model. This model can be extended to the time-varying CAPM of Tsay (2005, Section 11.3.1). We estimate both models using *SsfPack*. The example in Listing 10 performs the maximum likelihood estimation of the CAPM and time-varying CAPM model. The estimation is performed in a similar fashion as Example11.1.ox (Section 4.1.2). The only difference, besides the different models entertained, is that the static variable `s_sModel` determines if the CAPM model or the time-varying CAPM model is estimated. As expected, the result for the CAPM is in total agreement with that of the OLS method (also see output Listing 11). Note that for the time-varying case the estimates of σ_η and σ_ε are 2.38×10^{-9} and 1.22×10^{-2}, respectively. These estimates are close to zero, indicating that α_t and β_t of the the time-varying market model are essentially constant for the GM stock returns. This is in agreement with the fact that the fixed-coefficient market model fits the data well.

```
Likelihood(const vP, const pdLik, const pvSco, const pmHes)
{                                                   // arguments dictated by MaxBFGS()
    decl ret_val;

    GetSsfReg(s_mX',&s_mPhi, &s_mOmega, &s_mSigma, // Get state space model
    &s_mJPhi);
    switch (s_sModel)
    {                                               // Update Omega
        case 0:
            s_mOmega[2][2] = exp(vP);
            break;
        case 1:
            s_mOmega = setdiagonal(s_mOmega,exp(vP));
            break;
        default:
            println("Such model not programmed");
            break;
    }
    ret_val = SsfLik(pdLik, &s_dVar, s_vY',s_mPhi, // Get loglike
    s_mOmega, s_mSigma, <>, s_mJPhi, <>, <>, s_mX');
    pdLik[0] /= s_cT;                               // log-likelihood scaled by n
    return ret_val;                                 // 1 = success, 0 failure
}
MaxLik()
{
    decl vp, ir, dlik;
    switch (s_sModel)
    {                                               // starting value
        case 0:
            vp = <0.1>;
            break;
        case 1:
            vp = <0;0;0>;
            break;
        default:
            println("Such model not programmed");
            break;
    }
    Likelihood(vp, &dlik, 0, 0);                    // evaluate lik at start val
    vp += 0.5 * log(s_dVar);                        // scale starting values
    ir = MaxBFGS(Likelihood,&vp,&dlik, 0 , 1);      // max lik estimation with analytical scores
    print("---- Maximum likelihood estimation of state space CAPM ----");
    print("\n", MaxConvergenceMsg(ir),              // printing results state space estimation
          " using analytical derivatives",
          "\nLog-likelihood = ", dlik * s_cT,
          "; n = ", s_cT);
    print("%13g","%c",{"Sigma e", "Sigma eps",
          "Sigma eta"}, "%r", {"Estimates"} ,
          reverser(sqrt(exp(vp))'));
    s_dVar = exp(vp);
}
```

```
Maximum likelihood estimation, strong convergence using analytical derivatives
          Log-likelihood    n       σ_e          σ_ε          σ_η
CAPM      179.068          168   0.0813008
tv-CAPM   179.074          168   0.0812533   0.01219   2.3821e-009
```

Listing 10: Part of Example11.2.ox with corresponding output.

4.2.3 State smoothing CAPM and tv-CAPM

We now perform smoothing algorithms to the CAPM and time-varying CAPM model and present some time plots for the time-varying CAPM fit. The *SsfPack* routine `SsfMomentEst` as well as `SsfCondDens` are used to compute the smoothed estimates of the state vector, see Koopman et al. (1999) for a detailed description of these routines. The output in Listing 11 confirms that the result of the CAPM is in total agreement with that of the OLS method. Furthermore some time plots for the time-varying CAPM fit are presented. Part (a) is the monthly simple excess return of GM stock from January 1990 to December 2003. Part (b) is the expected returns of GM stock, part (c) and (d) are the time plots of the estimates of α_t and β_t. Given the tightness in the vertical scale, these two time plots confirm the assertion that a fixed-coefficient market model is adequate for the monthly GM stock return.

4.3 Unobserved components time series model

In the following illustration we reanalyze the series of quarterly earnings per share of Johnson and Johnson from 1960 to 1980 using the unobserved component model. For model specification see Tsay (2005, page 535 and 536). We split our analysis up in two parts. We start with the maximum likelihood estimation, after which Kalman filter and smoother are applied to the data.

4.3.1 Maximum likelihood estimation STSM

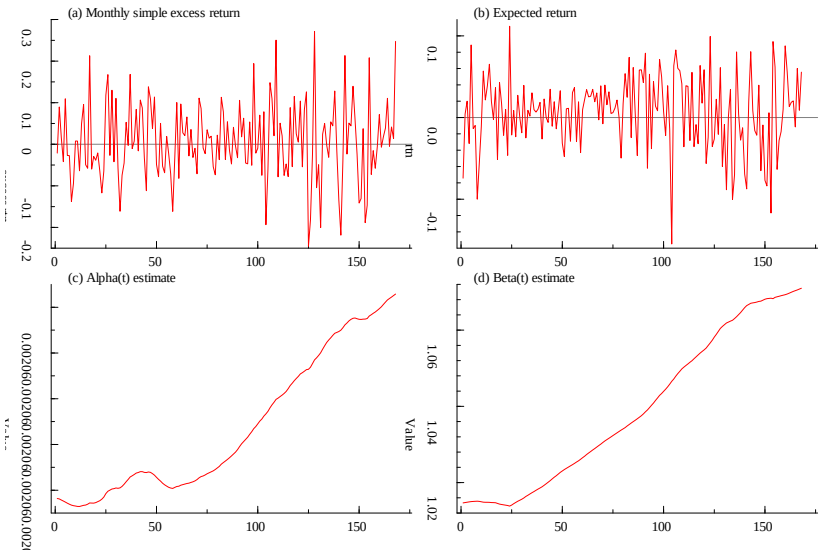
The model under consideration is a special case of the structural time series in *SsfPack* and can easily be specified using the `GetSsfStsm` routine. The example in Listing 12 performs the maximum likelihood estimation of the STSM model. Again the estimation is performed in a similar fashion as before by using the functions `MaxLik` and `Likelihood`. Performing the maximum likelihood estimation, we obtain $(\widehat{\sigma}_e, \widehat{\sigma}_\eta, \widehat{\sigma}_\omega) = (0.00418, 0.2696, 0.1712)$.

4.3.2 Kalman filter and smoother STSM

We now perform the Kalman filter and smoother, by using the *SsfPack* routine `SsfMomentEst, KalmanFil` and `KalmanSmo`, see Listing 13 for the program and output. Part (a) and (b) shows the smoothed estimates of the trend and seasonal components. Of particular interest is that the seasonal pattern seems to evolve over time. Also shown are 95% pointwise confidence regions of the unobserved components. Part (c) and (d) shows the residual plots, where (c) gives the 1-step ahead forecast errors computed by Kalman filter and (d) is the smoothed response residuals of the fitted model. Thus, state space modeling provides an alternative approach for analyzing seasonal time series.

```
Smoothing()
{
    decl mSmo, mks, mD;
    GetSsfReg(s_mX',&s_mPhi, &s_mOmega, &s_mSigma, // Get estimated state space model
    &s_mJPhi);
    switch (s_sModel)
    {                                              // starting value
        case 0:
            s_mOmega[2][2] = s_dVar;               // Estimated Omega
            mSmo = SsfMomentEst(ST_SMO, &mks, s_vY',// Perform smoothing
                   s_mPhi, s_mOmega, s_mSigma, <>,
                   s_mJPhi,   <>, <>, s_mX');      // Obtain estimate and standard errors
            print("%13g","Coefficients:","%c",{"Value",
                  "Std. Err."},"%r",{"Intercept","S&P"},
                  mks[0:1][10]~sqrt(mks[3:4][10])); // Use 10th row to avoid impact of starting value
            break;
        case 1:
            s_mOmega = diag(s_dVar);               // Values for sigma_eta, sigma_eps and sigma_e
            mD = SsfCondDens(ST_SMO, s_vY',s_mPhi, // Perform smoothing
                s_mOmega, s_mSigma, <>, s_mJPhi, <>,
                <>, s_mX');
            DrawTMatrix(0,s_vY',1,1,1,1,0,2);      // Draw results
            DrawTMatrix(1,mD[2][],1,1,1,1,0,2);
            DrawTMatrix(2,mD[0][],1,1,1,1,0,2);
            DrawTMatrix(3,mD[1][],1,1,1,1,0,2);
            ShowDrawWindow();
            break;
        default:
            println("Such model not programmed");
            break;
    }
}
```

```
---- Smoothing estimate CAPM model----
Intercept   0.0019820   (0.0063021)
S&P         1.0457      (0.14531)
```



Listing 11: Part of Example11.2.ox with corresponding output.

```
Likelihood(const vP, const pdLik, const pvSco, const pmHes)
{                                          // arguments dictated by MaxBFGS()
    decl ret_val;
    s_mSsf[0:2][1] = exp(vP);              // update ssf definition
    GetSsfStsm(s_mSsf, &s_mPhi, &s_mOmega, // get state space model
    &s_mSigma);

    ret_val = SsfLik(pdLik, &s_dVar, s_mY, // returns log-likelihood function
    s_mPhi, s_mOmega, s_mSigma);
    pdLik[0] /= s_cT;                      // log-likelihood scaled by n
    return ret_val;                        // 1 = success, 0 failure
}
MaxLik()
{
    decl vp, ir, dlik;
    s_mSsf = <CMP_IRREG, 1, 0, 0;
              CMP_LEVEL, 1, 0, 0;
              CMP_SEAS_DUMMY, 1, 4, 0>;    // set state space definition matrix
    vp = <0;0;0>;                          // starting values sigma's
    Likelihood(vp, &dlik, 0, 0);           // evaluate lik at start val
    vp += 0.5 * log(s_dVar);               // scale starting values
    //MaxControl(10, 5, 1);
    ir = MaxBFGS(Likelihood,&vp,&dlik,0,1); // max lik estimation with analytical scores
    s_dVar = exp(vp);
    s_mOmega = setdiagonal(s_mOmega, s_dVar[1:2]
              | 0 | 0 | s_dVar[0]);        // Specify model with estimates

    print("---- Maximum likelihood estimation of unobserved component model ----");
    print("\n", MaxConvergenceMsg(ir),     // printing results state space estimation
          " using analytical derivatives",
          "\nLog-likelihood = ", dlik * s_cT,
          "; n = ", s_cT);
    print("%13g","%c",{"Sigma e", "Sigma eps",
          "Sigma omega"}, "%r", {"Estimates"},
          sqrt(exp(vp))');
    print("Phi", s_mPhi,"Omega", s_mOmega,"Sigma", s_mSigma);
}
```

```
---- Maximum likelihood estimation of unobserved component model ----
Weak convergence (no improvement in line search) using analytical derivatives
```

	$\widehat{\sigma}_e$	$\widehat{\sigma}_\eta$	$\widehat{\sigma}_\omega$
Estimates	0.00418491	0.269612	0.171245

Log-likelihood = 63.7541; n = 84

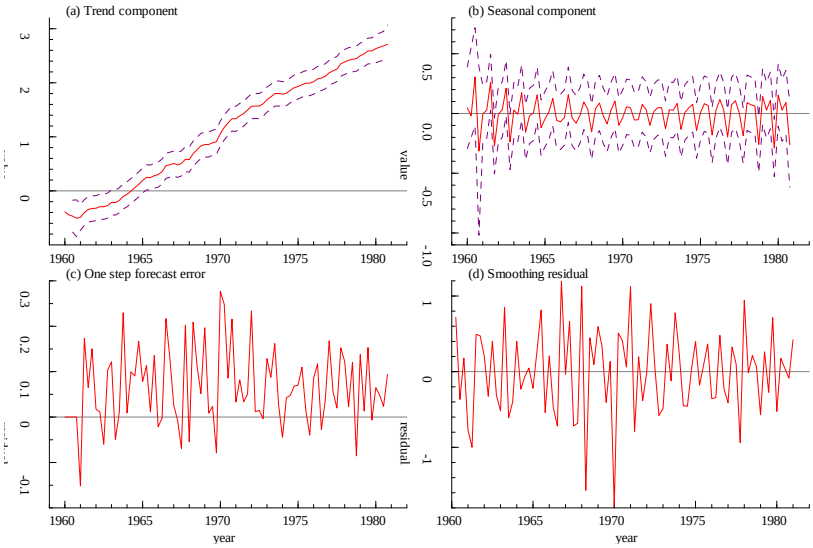
$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0.7269 & 0 & 0 & 0 & 0 \\ 0 & 0.29325 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.7513e-005 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Listing 12: Part of Example11.3.ox with corresponding output.

```
FilterSmoother()
{
    decl mkf, mks, mSmo, mKS;

    mSmo= SsfMomentEst(ST_SMO, &mKS, s_mY, // perform smoothing
          s_mPhi, s_mOmega, s_mSigma);
    mkf = KalmanFil(s_mY,s_mPhi, s_mOmega, // kalman filter
          s_mSigma);
    mks = KalmanSmo(mkf, s_mPhi, s_mOmega, // kalman smoother
          s_mSigma);
    mkf[0][0:3]=0;

    DrawTMatrix(0, mKS[0][],{""}, 1960, 1, 4, 0, 2);
    DrawZ(sqrt(mKS[5][]), "", ZMODE_BAND, 2.0, 8);
    DrawTMatrix(1, mKS[1][],{""}, 1960, 1, 4, 0, 2);
    DrawZ(sqrt(mKS[6][]), "", ZMODE_BAND, 2.0, 8);
    DrawTMatrix(2, mkf[0][],{""}, 1960, 1, 4, 0, 2);
    DrawTMatrix(3, mks[4][1:],{""}, 1960, 2, 4, 0, 2);
    ShowDrawWindow();
}
```



Listing 13: Part of Example11.3.ox with corresponding output.

4.4 Further illustrations

In the following we will consider exercise 11.2, 11.3 and 11.5 from Tsay (2005, Chapter 11). These illustrations use the same methods as the illustrations in Section 4.1, 4.2 and 4.3, respectively. Since the *Ox* code is essentially the same as before we do not discuss it here. The *Ox* code for these problems is available on the website `www.ssfpack.com`, under program name *Exercise11.2ox, Exercise11.3ox* and *Exercise11.5.ox*. In the following we will discuss the results from our analysis.

4.4.1 Tsay (2005) exercise 11.2

We start our analyses with the obtained ARIMA(0,1,1) model

$$(1-B)y_t = (1-0.875B)a_t, \quad \widehat{\sigma}_a = 0.6017.$$

where y_t is the log realized volatility of Alcoa stock returns and the standard error of $\widehat{\theta}$ is 0.04058. The volatility series is constructed using 20-minute intradaily log returns. The *Ox* code in Listing 6, essentially, applies the ARIMA(0,1,1) maximum likelihood estimation. Only a different dataset is used and we did not performed any model checking

Next we estimate the local level model see equations (5) and (6) for the log volatility series. Details of estimation are discussed in Section 4.1.2 (Listing 7). The obtained estimates are $\widehat{\sigma}_\eta = 0.0754$ and $\widehat{\sigma}_e = 0.5637$. Again, the measurement errors have a larger variance than the state innovations, confirming that intraday high-frequency returns are subject to measurement errors. The obtained time plots for the filtered and smoothed state variables with pointwise 95% confidence intervals can be found in Figure 1. As expected, the smoothed state variables are smoother than the filtered state variables. The confidence intervals for the smoothed state variables are also narrower than those of the filtered state variables.

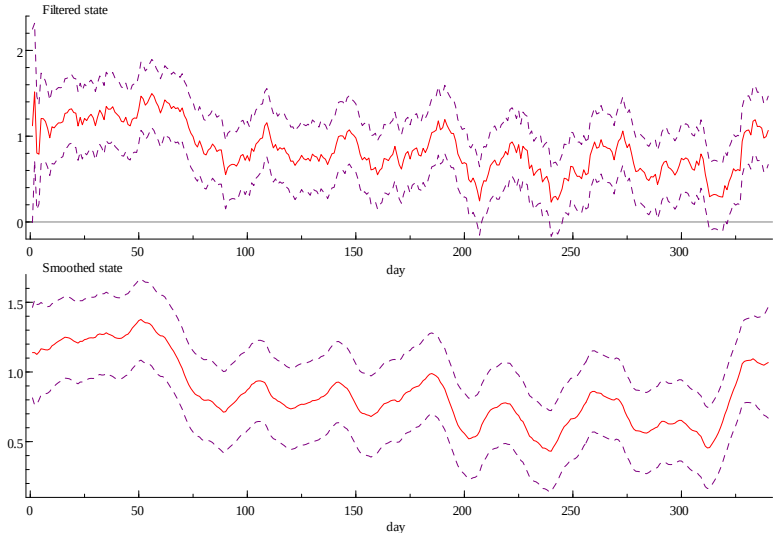


Figure 1: Filtered and smoothed state variable and its 95% pointwise confidence interval for the intraday 20-minute log realized volatility of Alcoa stock returns based on the fitted local level state space model.

4.4.2 Tsay (2005) exercise 11.3

We consider the monthly simple excess returns of Pfizer stock and the S&P 500 composite index from January 1990 to December 2003. The obtained fixed-coefficient market model to the Pfizer stock return is (standard errors in parenthesis)

$$r_t = \underset{(0.0051)}{0.012} + \underset{(0.1188)}{0.851}\, r_{M,t} + e_t, \qquad \widehat{\sigma}_e = 0.0661.$$

Next a time-varying CAPM to the Pfizer stock return is fitted, see Tsay (2005, equation (11.27)). The obtained estimates of the standard errors of the measurement and innovation equation are

$$\sigma_e = 0.064, \qquad \sigma_\eta = 6.89 \times 10^{-5}, \qquad \sigma_\varepsilon = 0.068.$$

Clearly, the innovations of the α_t series do not seem to be time varying ($\sigma_\eta \simeq 0$), while the innovation series to the β_t series do seem to be time-varying. The time plots of the smoothed estimates of α_t and β_t suggest the same conclusions, see Figure 2. Given the tightness in the vertical scale of α_t, this plot confirms the assertion that α_t is not time-varying. However the slope of the time-varying CAPM does seem to be time-varying.

4.4.3 Tsay (2005) exercise 11.5

Finally we consider monthly producer price index (PPI) from January 1921 to August 2008. The index is for all commodities and not seasonally adjusted. Let $z_t = \ln(Z_t) - \ln(Z_{t-1})$, where Z_t is the observed

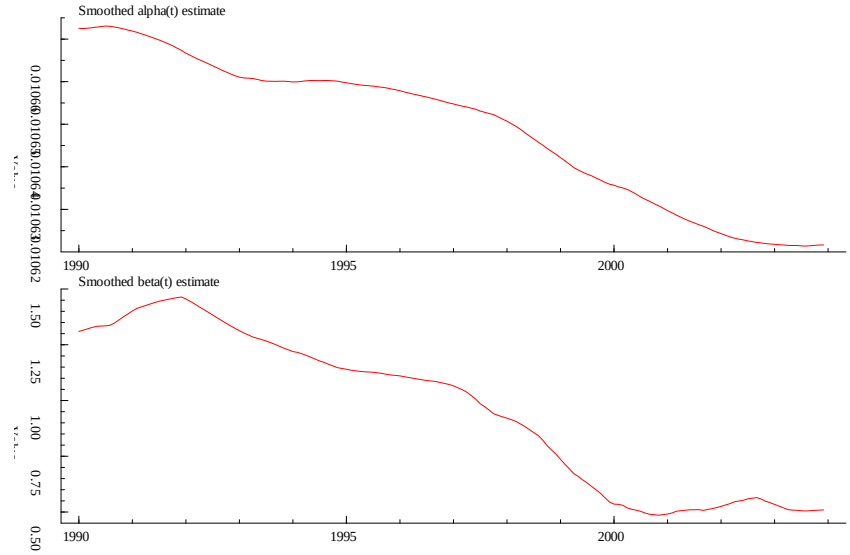


Figure 2: Time plots of the smoothed estimates of α_t and β_t for a time-varying CAPM applied to the monthly simple excess returns of Pfizer stock. The S&P 500 composite index return is used as the market return.

monthly PPI. It turns out that an AR(3) model is adequate for y_t if the minor seasonal dependence is ignored. Let y_t be the sample mean corrected series of z_t. The obtained AR(3) model to y_t, by using the *Arfima* package as in Section 4.1.1, is

$$y_t = \underset{(0.030)}{0.281} y_{t-1} + \underset{(0.033)}{0.073} y_{t-2} + \underset{(0.032)}{0.181} y_{t-3} + e_t, \quad \widehat{\sigma}_e = 0.00991.$$

Next we consider the AR(3) model

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \phi_3 x_{t-3} + a_t, \qquad a_t \overset{d}{=} \mathrm{N}(0, \sigma_a^2),$$

and suppose that the observed data are

$$y_t = x_t + e_t, \qquad e_t \overset{d}{=} \mathrm{N}(0, \sigma_e^2),$$

where $\{e_t\}$ and $\{a_t\}$ are independent and the initial values of x_j with $j \leq 0$ are independent of e_t and a_t for $t > 0$. We used a state space form to estimate parameters, including the inovational variances to the state and σ_e^2. The estimated fitted model is

$$\begin{aligned} x_t &= 1.30 x_{t-1} - 0.028 x_{t-2} + -0.27 x_{t-3} + a_t, \quad \sigma_a^2 = 5.19 \times 10^{-7} \\ y_t &= x_t + e_t, \quad \sigma_e^2 = 8.21 \times 10^{-5}, \end{aligned} \tag{9}$$

with state space matrices

$$\Phi = \begin{pmatrix} 1.3031 & 1 & 0 \\ -0.02841 & 0 & 1 \\ -0.27463 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 5.19 \times 10^{-7} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8.21 \times 10^{-5} \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -0.13039 & 0.039514 & 0.035809 \\ 0.039514 & -0.011974 & -0.010852 \\ 0.035809 & -0.010852 & -0.0098344 \\ 0 & 0 & 0 \end{pmatrix}.$$

Also, we show the time plots of the smoothed estimate of x_t and the time plot of filtered response residuals of the fitted state space model, see Figure 3.

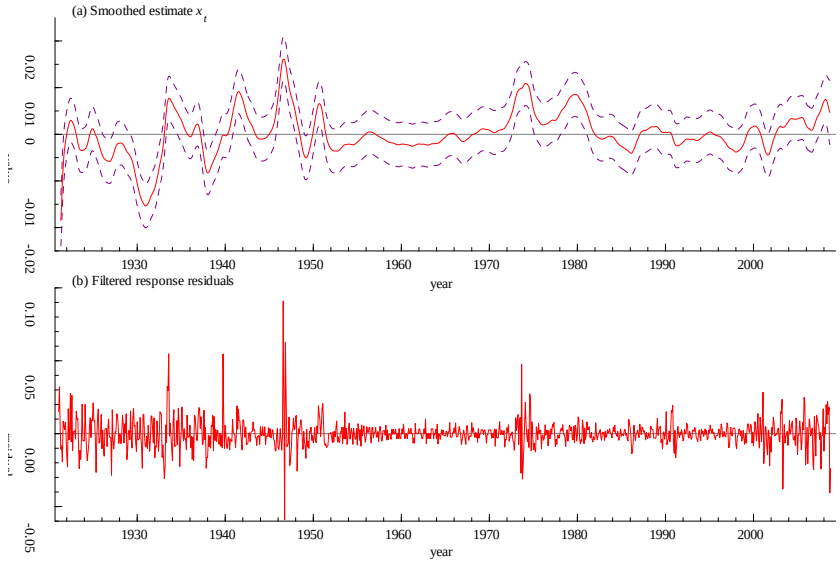


Figure 3: Time series plots of fitted model 9 to the logarithms of monthly producer price index from January 1921 to August 2008. (a) smoothed estimate of x_t, (b) smoothed residuals of response variable.

5 Conclusions

In this paper we have discussed examples of *SsfPack*, which is a library of statistical and econometric algorithms for state space models. The functionality is presented here as an extension to the *Ox* language. A wide variety of models can be handled in this unified framework. The *Ox* code is provided for all illustrations. This paper is established as supplement to Koopman et al. (1999), as it contains more examples of the *SsfPack* commands used to perform state space analysis.

Acknowledgements

We gratefully acknowledge Ruey S. Tsay for allowing to use his examples and data, from Analysis of Financial Time Series (2005), as illustrations for state space modelling using *SsfPack* in *Ox* programming language.

References

Doornik, J.A., 1998. Object-oriented Matrix Programming using Ox 2.0. *Timberlake Consultants Press*, London.

Doornik, J.A., Ooms, M., 2006a. A Package for Estimating, Forecasting and Simulating ArfimaModels: Arfima package 1.04 for Ox. *Discussion paper*, Nuffield College, Oxford, UK

Doornik, J.A., Ooms, M. 2006b. Introduction to Ox, London: *Timberlake Consultants Press*

Durbin, J., Koopman, S.J., 2001. Time Series Analysis by State Space Methods. *Oxford University Press*, Oxford, UK

Koopman, S.J., Shephard, N., Doornik, J.A., 1999. Statistical algorithms for models in state space using SsfPack 2.2. *Econometrics Journal 2*, 133-166.

Tsay, R.S., 2005. Analysis of Financial Time Series. *Wiley-Interscience*, New Jersey.