Econometrics Journal (1998), volume **1**, pp. 1–55.

Statistical algorithms for models in state space using SsfPack 2.2*

Siem Jan Koopman♯, Neil Shephard♭, Jurgen A Doornik♭

♯*CentER, Tilburg University, 5000 LE Tilburg, The Netherlands*
s.j.koopman@kub.nl

♭*Nuffield College, Oxford, OX1 1NF, UK*
neil.shephard@nuffield.oxford.ac.uk, jurgen.doornik@nuffield.oxford.ac.uk

SsfPack: http://center.kub.nl/stamp/ssfpack.htm
Ox: http://www.nuff.ox.ac.uk/users/doornik/

Received: November 1998

Summary This paper discusses and documents the algorithms of *SsfPack 2.2*. *SsfPack* is a suite of C routines for carrying out computations involving the statistical analysis of univariate and multivariate models in state space form. The emphasis is on documenting the link we have made to the Ox computing environment. *SsfPack* allows for a full range of different state space forms: from a simple time-invariant model to a complicated time-varying model. Functions can be used which put standard models such as ARIMA and cubic spline models in state space form. Basic functions are available for filtering, moment smoothing and simulation smoothing. Ready-to-use functions are provided for standard tasks such as likelihood evaluation, forecasting and signal extraction. We show that *SsfPack* can be easily used for implementing, fitting and analysing Gaussian models relevant to many areas of econometrics and statistics. Some Gaussian illustrations are given.

Keywords: *Kalman filtering and smoothing; Markov chain Monte Carlo; Ox; Simulation smoother; State space.*

JEL classification: C10, C15, C22.

*We wish to thank Marius Ooms, Peter Boswijk, and an anonymous referee for many helpful comments and suggestions. Financial support from the Royal Netherlands Academy of Arts and Sciences (SJK), the UK Economic and Social Research Council (grant R000237500, JAD) is gratefully acknowledged.

1. INTRODUCTION

This paper documents the package *SsfPack 2.2* which carries out computations for the statistical analysis of general univariate and multivariate state space models. *SsfPack* allows for a full range of different state space forms: from a simple univariate autoregressive model to a complicated time-varying model for aggregated variables. In particular, it can be used in many areas of econometrics and statistics as will become apparent from the illustrations given.

Statistical and econometric packages such as SAS, S-PLUS, SPSS, PcGive, STAMP, and Minitab have many canned options for the fitting of standard time series models. However, when we work on new areas of time series modelling it is important to have generic programming tools which offer complete flexibility to carry out the computational problem. *SsfPack* provides such a tool, in the form of filtering, moment smoothing and simulation smoothing routines which are general, fast, and easy to use.

SsfPack is a suite of C routines collected into a library which can be linked to different computing environments. The version discussed here is linked to the *Ox 2.0* (or later) matrix programming language of Doornik (1998). All examples presented here are in the form of *Ox* code; this allows us focus on the important features of *SsfPack*. Although not discussed here, it is also possible to call the C functions of *SsfPack* from other computing environments.

SsfPack can be downloaded from the address given on the title page. It may be used freely for non-commercial purposes. The *SsfPack* web site also provides installation details. The *Ox* web site has tutorials and online help for *Ox*, as well as a downloadable version. Please cite this paper and Doornik (1998) when using *SsfPack*.

We begin by introducing the state space form, and the *SsfPack* notation (§2). Section 3 discusses the state space formulation for several econometric and statistical models. It also documents the functions provided by *SsfPack* for this purpose. This shows the generality of the state space form and the flexibility of *SsfPack*. The recursive algorithms associated with the Kalman filter are given in §4, including algorithms for smoothing and simulation. The emphasis is on efficient implementation; also, missing values are handled transparently. Examples are given at every stage, using artificially generated data. In §5 we turn to more practical problems, showing how the special functions for estimation, signal extraction, and forecasting can be used. The examples include estimation and forecasting of an ARMA model; estimation and outlier detection of an unobserved components model; spline interpolation when missing values are present; recursive estimation of a regression model. Section 6 considers more advanced applications, including seasonal adjustment; combining models; bootstrapping; Bayesian analysis of a gaussian state space model. The final section concludes. The Appendix summarizes the *SsfPack* functions and example programs for *Ox*. Starred sections are considered more technical and may be skipped on first reading.

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, Harvey (1993, Chapter 4), West and Harrison (1997), Kitagawa and Gersch (1996). This framework also encapsulates different specifications for nonparametric and spline regressions. The Gaussian state space form consists of a transition equation and a measurement equation; we formulate it as

$$\alpha_{t+1} = d_t + T_t\alpha_t + H_t\varepsilon_t, \qquad \alpha_1 \sim \mathrm{N}(a, P), \qquad t = 1, \ldots, n, \tag{1}$$

$$\theta_t = c_t + Z_t\alpha_t, \tag{2}$$

$$y_t = \theta_t + G_t\varepsilon_t, \qquad \varepsilon_t \sim \mathrm{NID}(0, I), \tag{3}$$

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 $(y_1, \ldots, y_n)$. The $m \times 1$ state vector α_t contains unobserved stochastic processes and unknown fixed effects. The state equation (1) has a Markovian structure which is an effective way to describe the serial correlation structure of the time series y_t. The initial state vector is assumed to be random with mean a and variance matrix P but more details are given in §2.4. The measurement equation (3) relates the observation vector y_t in terms of the state vector α_t through the signal θ_t of (2), and the vector of disturbances ε_t. The deterministic matrices T_t, Z_t, H_t and G_t are referred to as system matrices and they usually are 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.

2.1. The state space representation in SsfPack

The state space form in *SsfPack* is represented by:

$$\begin{pmatrix} \alpha_{t+1} \\ y_t \end{pmatrix} = \delta_t + \Phi_t\alpha_t + u_t, \qquad u_t \sim \mathrm{NID}(0, \Omega_t), \qquad t = 1, \ldots, n,$$

$$\delta_t = \begin{pmatrix} d_t \\ c_t \end{pmatrix}, \quad \Phi_t = \begin{pmatrix} T_t \\ Z_t \end{pmatrix}, \quad u_t = \begin{pmatrix} H_t \\ G_t \end{pmatrix}\varepsilon_t, \quad \Omega_t = \begin{pmatrix} H_tH_t' & H_tG_t' \\ G_tH_t' & G_tG_t' \end{pmatrix}, \tag{4}$$

$$\alpha_1 \sim \mathrm{N}(a, P).$$

The vector δ_t is $(m+N) \times 1$, the matrix Φ_t is $(m+N) \times m$ and Ω_t is $(m+N) \times (m+N)$. 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 (see §2.5). For

$\alpha_{t+1}, d_t, a:$	$m \times 1,$	$y_t, \theta_t, c_t:$	$N \times 1,$	$\varepsilon_t:$	$r \times 1,$
$T_t, P:$	$m \times m,$	$Z_t:$	$N \times m,$		
$H_t:$	$m \times r,$	$G_t:$	$N \times r.$		
$\Phi:$	$(m+N) \times m$	δ	$(m+N) \times 1$		
Ω	$(m+N) \times (m+N)$	Σ	$(m+1) \times m$		

m : dimension of the state vector;
N : number of variables;
n : number of observations;
r : dimension of the disturbance vector.

Table 1. Dimensions of state space matrices

this elementary case only two matrices are required, that is

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

The dimensions are summarized in Table 1.

For example, consider the local linear trend model:

$$\begin{aligned} \mu_{t+1} &= \mu_t + \beta_t + \eta_t, & \eta_t &\sim \text{NID}\left(0, \sigma_\eta^2\right), \\ \beta_{t+1} &= \beta_t + \zeta_t, & \zeta_t &\sim \text{NID}\left(0, \sigma_\zeta^2\right), \\ y_t &= \mu_t + \xi_t, & \xi_t &\sim \text{NID}\left(0, \sigma_\xi^2\right), \end{aligned} \tag{5}$$

with $\mu_1 \sim \text{NID}(0, \kappa)$ and $\beta_1 \sim \text{NID}(0, \kappa)$ where κ is large; for more details about this model, see §3.2. The state vector contains the trend component μ_t and the slope component β_t, that is $\alpha_t = (\mu_t, \beta_t)'$. The matrices Φ and Ω for model (5) are given by

$$\Phi = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \Omega = \begin{pmatrix} \sigma_\eta^2 & 0 & 0 \\ 0 & \sigma_\zeta^2 & 0 \\ 0 & 0 & \sigma_\xi^2 \end{pmatrix}.$$

In *Ox* code, when $\sigma_\eta^2 = 0$, $\sigma_\zeta^2 = 0.1$, and $\sigma_\xi^2 = 1$, these matrices can be created as follows:

```
mPhi = <1,1;0,1;1,0>;
mOmega = diag(<0,0.1,1>);
```

2.2. Data sets used in the illustrations

SsfPack expects all data variables to be in *row* vectors. This is different from most other *Ox* packages. Various data formats can be loaded easily in *Ox*, such as Excel and PcGive files. In this paper we use plain data files, with the first two entries in the file specifying the matrix dimensions (normally these are `.mat` files, but here we use the `.dat` extension). Many examples therefore start with a statement like:

```
mYt = loadmat("Nile.dat")';
```

which creates `mYt` as an $1 \times n$ matrix with the Nile data. This is a series of readings of the annual flow of the Nile river at Aswan for 1871 to 1970. This series is originally considered by Cobb (1978) and analysed more recently by Balke (1993).

A second data set used in this paper is the airline data, consisting of the number of UK airline passengers (in thousands, from January 1949 to December 1960), see Box and Jenkins (1976). Both are graphed in Figure 1.

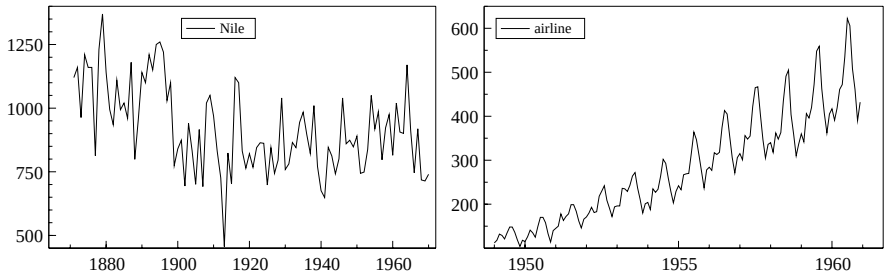


Figure 1. Nile and airline data

2.3. Initial conditions

The variance matrix P of the initial state vector α_1 may contain diffuse elements:

$$P = P_* + \kappa P_\infty, \qquad \kappa \text{ is large}, \tag{6}$$

where P_* is a symmetric $m \times m$ matrix, P_∞ is a diagonal $m \times m$ matrix composed of zero and unity values, and, for example, $\kappa = 10^6$. When the i-th diagonal element of P_∞ is unity, the corresponding i-th column and row of P_* are assumed to be zero. To specify the initial state conditions (6) in *SsfPack* explicitly, the $(m+1) \times m$ matrix

$$\Sigma = \begin{pmatrix} P \\ a' \end{pmatrix}, \tag{7}$$

is required. The block matrix P in Σ is equal to matrix P_* except when a diagonal element of P is equal to -1, indicating that the corresponding initial state vector element is diffuse. When a diagonal element of P is -1, the corresponding row and column of P are ignored. When the initial state conditions are not explicitly defined, it will be assumed that the state vector is fully diffuse, that is

$$a = 0, \qquad P_* = 0, \qquad P_\infty = I, \tag{8}$$

such that $\alpha_1 \sim \mathrm{N}(0, \kappa I)$ where κ is the numerical value 10^6. If any diagonal value of Ω is larger than unity, the constant κ will be multiplied by the maximum diagonal value of Ω. In short, we formally have

$$\kappa = 10^6 \times \max\{1, \mathrm{diag}(\Omega)\}.$$

In certain circumstances this automatic procedure of dealing with diffuse initialization may not be desirable and the user may wish to specify P freely. For example, the user may prefer to input

```
mSigma = <10^3,0;0,10^8;0,0>;
```

instead of

```
mSigma = <-1,0;0,-1;0,0>;
```

However, it is advisable to use the constant -1 in matrix Σ for a diffuse initial state element; for example, it will be more straightforward to calculate the appropriate likelihood function for certain models. The authors are working on a version of *SsfPack* which allows the limiting case $\kappa \to \infty$. This exact diffuse treatment requires specific adjustments to the basic functions of *SsfPack*; see Koopman (1997). Finally, for stationary time series models in state space, a well-defined initial variance matrix P can be constructed which does not depend on κ; see §3.1 for an example.

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. We introduce the index matrices J_Φ, J_Ω and J_δ which must have the same dimension as Φ, Ω and δ, respectively. The elements of the index matrices are all set to -1 except the elements for which the corresponding elements in Φ, Ω and δ are time varying. The non-negative index value indicates the row of some data matrix which contain the time varying values. When no element of a system matrix is time-varying, the corresponding index matrix can be set to an empty matrix; in *Ox*, that is <>. For example, the local linear trend model (5) with time-varying variances (instead of the variances being constant) is defined as

```
mJ_Phi = mJ_Delta = <>;
mJ_Omega = <4,-1,-1;-1,0,-1;-1,-1,2>;
```

indicating that the variances of ξ_t are found in the third row of an accompanying data matrix (note that indexing starts at value 0 in *Ox*). We could also have created J_Ω by first creating a matrix of -1's, and then setting the diagonal:

```
mJ_Omega = constant(-1, mOmega);
mJ_Omega = setdiagonal(mJ_Omega, <4,0,2>);
```

The variances of η_t and ζ_t are to be found in the fifth row and the first row, respectively, of the data matrix, which must have at least five rows and n columns. No element of Φ is time-varying, therefore we set J_Φ and J_δ to empty matrices. Examples of time-varying state space models can be found in §3.3 and §3.4.

2.5. Formulating the state space in SsfPack

The most elementary state space form is time-invariant and it only requires the matrix specifications of Φ and Ω; in this case, it is assumed that $\delta = 0$, $a = 0$ and $P = \kappa I$

with $\kappa = 10^6 \times \max\{1, \text{diag}(\Omega)\}$. In addition, initial conditions can explicitly be given by defining an appropriate matrix Σ. The time-invariant vector δ can also be given when it is nonzero. Thus, a time-invariant state space form can be inputted in one of three different formats:

```
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, mJ_Phi, mJ_Omega, mJ_Delta, mXt
```

where `mXt` is the data matrix with n columns as discussed in §2.4.

2.6. Missing values

The algorithms of *SsfPack* can handle missing values. A missing value is only recognised within the data matrix $(y_1, \ldots, y_n)$. A dot in an *Ox* matrix constant indicates a missing value. Alternatively, the constant value `M_NAN` may be used in any expression. For example, the second element of the vector

```
<1,.,3,4,5>;
```

is treated as missing. No missing values are allowed within the matrices Φ, Ω, Σ and δ or their time-varying counterparts.

The vector of observations y_t with missing entries will be reduced to the vector $y_t^\dagger$ without missing entries so that the measurement equation must be adjusted accordingly. For example, the measurement equation $y_t = c_t + Z_t\alpha_t + G_t\varepsilon_t$ with

$$y_t = \begin{pmatrix} 5 \\ . \\ 3 \\ . \end{pmatrix}, \quad c_t = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}, \quad Z_t = \begin{pmatrix} Z_{1,t} \\ Z_{2,t} \\ Z_{3,t} \\ Z_{4,t} \end{pmatrix}, \quad G_t = \begin{pmatrix} G_{1,t} \\ G_{2,t} \\ G_{3,t} \\ G_{4,t} \end{pmatrix},$$

reduces to the measurement equation $y_t^\dagger = c_t^\dagger + Z_t^\dagger\alpha_t + G_t^\dagger\varepsilon_t$ with

$$y_t^\dagger = \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \quad c_t^\dagger = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \quad Z_t^\dagger = \begin{pmatrix} Z_{1,t} \\ Z_{3,t} \end{pmatrix}, \quad G_t^\dagger = \begin{pmatrix} G_{1,t} \\ G_{3,t} \end{pmatrix}.$$

The algorithms of *SsfPack* automatically replace the observation vector y_t by $y_t^\dagger$ when some entries of y_t are missing. Other matrices are adjusted accordingly, so the input arguments as well as the output are in terms of y_t, G_t, etc., rather than $y_t^\dagger, G_t^\dagger$. The case when all entries are missing is discussed in §4.3.

3. PUTTING LINEAR MODELS 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 those functions. However, the system matrices may still be constructed or modified manually, even after using the provided routines.

3.1. ARMA models

The autoregressive moving average model of order p and q, denoted by ARMA(p, q), is given by

$$y_t = \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \xi_t + \theta_1 \xi_{t-1} + \ldots + \theta_q \xi_{t-q}, \quad \xi_t \sim \text{NID}\left(0, \sigma_\xi^2\right). \tag{9}$$

The lag polynomial of order d is defined as $A(L) = 1 + A_1 L + \ldots + A_d L^d$ where L is the lag operator such that $L^r y_t = y_{t-r}$. In this notation, we can write the ARMA model as

$$\phi(L) y_t = \theta(L) \xi_t.$$

The model (9) is stationary when the roots of the polynomial $\phi(L) = 1 - \phi_1 L - \ldots - \phi_p L^p$ are outside the unit circle and the model is invertible when the roots of the polynomial $\theta(L) = 1 + \theta_1 L + \ldots + \theta_q L^q$ are outside the unit circle. The parameter space can be restricted to obtain a stationary invertible ARMA model by following the arguments in Ansley and Kohn (1986). Any ARMA model can be written as a first order vector autoregressive, VAR(1), model. Such a representation, which is not unique, is called a companion form or Markov representation. The most commonly quoted companion form of the ARMA model is $y_t = (1, 0, 0, \ldots, 0)\alpha_t$ and

$$\alpha_{t+1} = \begin{pmatrix} \phi_1 & 1 & 0 & \cdots & 0 \\ \phi_2 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \\ \phi_{m-1} & 0 & 0 & & 1 \\ \phi_m & 0 & 0 & \cdots & 0 \end{pmatrix} \alpha_t + \begin{pmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_{m-2} \\ \theta_{m-1} \end{pmatrix} \xi_t, \quad \xi_t \sim \text{NID}\left(0, \sigma_\xi^2\right), \tag{10}$$

with $m = \max(p, q+1)$, see e.g. Harvey (1993, §4.4). This can be compactly written as $\alpha_{t+1} = T_a \alpha_t + h\xi_t$ where the time-invariant matrices T_a and h are given in (10). Multivariate or vector ARMA models can also be written in the companion VAR(1) form. In the case of a stationary ARMA model in state space form, the unconditional distribution of the state vector is $\alpha_t \sim \text{N}(0, V)$, where $V = T_a V T_a' + \sigma_\xi^2 hh'$. There are different ways of numerically solving out for V. The most straightforward way is to invert a matrix in order to solve the linear equations $(I - T_a \otimes T_a)\,\text{vec}(V) = \sigma^2 \text{vec}(hh')$ for V, where vec(V) operator stacks the columns of V; see, for example, Magnus and Neudecker (1988, Theorem 2, p. 30). The variance matrix of the initial state vector is in this case equal to the unconditional variance matrix of the state vector, that is $P = V$.

SsfPack implementation. The *SsfPack* routine `GetSsfArma` provides the appropriate system matrices for any univariate ARMA model. The routine requires two vectors containing the autoregressive parameters $\phi_1, \ldots, \phi_p$ and the moving average parameters $\theta_1, \ldots, \theta_q$ 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`, containing the autoregressive and the moving average parameters, respectively, should be either row vectors or column vectors. The scalar value `dStDev` represents σ_ξ in (10). The remaining three arguments are used to receive the system matrices Φ, Ω and Σ. The `&` is used to pass a reference to the variable, which is changed on return.

Example. The following example outputs the relevant state space matrices for the ARMA(2,1) model $y_t = 0.6y_{t-1} + 0.2y_{t-1} + \xi_t - 0.2\xi_{t-1}$ with $\xi_t \sim \text{NID}(0, 0.9)$. The *Ox* code and output are given in Listing 1.

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

main()
{
    decl mphi, momega, msigma;
    GetSsfArma(<0.6,0.2>, <-0.2>, sqrt(0.9), &mphi, &momega, &msigma);
    print("Phi =", mphi, "Omega =", momega, "Sigma =", msigma);
}
```

```
Phi =
      0.60000       1.0000
      0.20000      0.00000
       1.0000      0.00000
Omega =
      0.90000     -0.18000      0.00000
     -0.18000     0.036000      0.00000
      0.00000      0.00000      0.00000
Sigma =
       1.4068    -0.013984
    -0.013984     0.092271
      0.00000      0.00000
```

Listing 1. ssfarma.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;`. In most examples below we only list the salient contents of `main()`. Then the `include` statements, `main()`, and variable declarations must be added to create an *Ox* program which can be run. An AR(2) and MA(1) model is respectively created as:

```
GetSsfArma(<0.6,0.2>, <>, sqrt(0.9), &mphi, &momega, &msigma);
GetSsfArma(<>, <-0.2>, sqrt(0.9), &mphi, &momega, &msigma);
```

3.2. Unobserved components time series models

The state space model also deals directly with unobserved components time series models used in structural time series and dynamic linear models; see, for example, West and Harrison (1997), Kitagawa and Gersch (1996) and Harvey (1989). 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. For example, a specific time series model may include the addition of a trend μ_t, a seasonal γ_t, a cycle ψ_t and an irregular ε_t component to give

$$y_t = \mu_t + \gamma_t + \psi_t + \xi_t, \quad \text{where} \quad \xi_t \sim \text{NID}\left(0, \sigma_\xi^2\right), \qquad t = 1, \ldots, n. \tag{11}$$

Explanatory variables (*i.e.* regression and intervention effects) can be included in this model straightforwardly.

Trend component. The trend component μ_t is usually specified as

$$\begin{array}{rcll} \mu_{t+1} & = & \mu_t + \beta_t + \eta_t, & \eta_t \sim \text{NID}\left(0, \sigma_\eta^2\right), \\ \beta_{t+1} & = & \beta_t + \zeta_t, & \zeta_t \sim \text{NID}\left(0, \sigma_\zeta^2\right), \end{array} \tag{12}$$

with $\mu_1 \sim \text{N}(0, \kappa)$ and $\beta_1 \sim \text{N}(0, \kappa)$ where κ is large. The model with trend and irregular is easily placed into state space form; see also §2.3. Model (12) is called the local linear trend model; the local level model arises when β_t is set to zero. Sometimes σ_η^2 of (12) is set to zero, and so we refer to μ_t as a smooth trend or an integrated random walk component. When σ_η^2 and σ_ζ^2 are both set to zero, we obtain a deterministic linear trend in which $\mu_t = \mu_1 + \beta_1(t-1)$.

Seasonal component. The specification of the seasonal component γ_t is given by

$$S(L)\gamma_t = \omega_t, \quad \text{where} \quad \omega_t \sim \text{NID}(0, \sigma_\omega^2) \quad \text{and} \quad S(L) = 1 + L + \ldots + L^{s-1}, \tag{13}$$

with s equal to the number of seasons, for $t = 1, \ldots, n$. When σ_ω^2 of (13) is set to zero, the seasonal component is fixed. In this case, the seasonal effects sum to zero over the previous 'year'; this ensures that it cannot be confounded with the other components. The state space representation for $s = 4$ is given by

$$\begin{pmatrix} \gamma_t \\ \gamma_{t-1} \\ \gamma_{t-2} \end{pmatrix} = \begin{pmatrix} -1 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \gamma_{t-1} \\ \gamma_{t-2} \\ \gamma_{t-3} \end{pmatrix} + \begin{pmatrix} \omega_t \\ 0 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} \gamma_1 \\ \gamma_0 \\ \gamma_{-1} \end{pmatrix} \sim \text{N}(0, \kappa I_3).$$

Other representations are discussed in §6.1.

Cycle component. The cycle component ψ_t is specified as

$$\begin{pmatrix} \psi_{t+1} \\ \psi^*_{t+1} \end{pmatrix} = \rho \begin{pmatrix} \cos\lambda_c & \sin\lambda_c \\ -\sin\lambda_c & \cos\lambda_c \end{pmatrix} \begin{pmatrix} \psi_t \\ \psi^*_t \end{pmatrix} + \begin{pmatrix} \chi_t \\ \chi^*_t \end{pmatrix}, \tag{14}$$

$$\begin{pmatrix} \chi_t \\ \chi^*_t \end{pmatrix} \sim \text{NID}\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma^2_\psi \left(1-\rho^2\right) I_2 \right\},$$

for which $0 < \rho \leq 1$ is the 'damping factor'. The frequency is $\lambda_c = 2\pi/c$, and c is the 'period' of the cycle. The initial conditions are $\psi_0 \sim \text{N}(0, \sigma^2_\psi)$ and $\psi^*_0 \sim \text{N}(0, \sigma^2_\psi)$ with $\text{cov}(\psi_0, \psi^*_0) = 0$. The variance of χ_t and χ^*_t is given in terms of σ^2_ψ and ρ so that when $\rho \to 1$ the cycle component reduces to a deterministic (but stationary) sine-cosine wave; see Harvey and Streibel (1998).

SsfPack implementation. The *SsfPack* routine `GetSsfStsm` provides the relevant system matrices for any univariate structural time series model:

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

The routine requires one input matrix containing the model information as follows:

$$\begin{array}{lllll} \texttt{mStsm} = \texttt{<} & \texttt{CMP\_LEVEL}, & \sigma_\eta, & 0, & 0; \\ & \texttt{CMP\_SLOPE}, & \sigma_\zeta, & 0, & 0; \\ & \texttt{CMP\_SEAS\_DUMMY}, & \sigma_\omega, & s, & 0; \\ & \texttt{CMP\_CYC\_0}, & \sigma_\psi, & \lambda_c, & \rho; \\ & \vdots & \vdots & \vdots & \vdots \\ & \texttt{CMP\_CYC\_9}, & \sigma_\psi, & \lambda_c, & \rho; \\ & \texttt{CMP\_IRREG}, & \sigma_\xi, & 0, & 0 \quad \texttt{>;} \end{array}$$

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 organised in the sequence level, slope, seasonal, cycle and irregular. The first column of `mStsm` uses predefined constants, and the remaining columns contain real values. `CMP_SEAS_DUMMY` refers to (13), where s is the number of seasonal component. The function `GetSsfStsm` returns the three system matrices Φ, Ω, and Σ in a similar fashion to `GetSsfArma` (§3.1). The inclusion of a regression effect into the model is discussed in §6.3.

```
GetSsfStsm(<CMP_IRREG,      1.0, 0, 0;
            CMP_LEVEL,      0.5, 0, 0;
            CMP_SEAS_DUMMY, 0.2, 3, 0;
            CMP_SLOPE,      0.1, 0, 0>, &mphi, &momega, &msigma);
```

$$\Phi = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0.25 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0.04 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Listing 2. Part of ssfstsm.ox with corresponding output

Example. The code in Listing 2 outputs the relevant state space matrices for a basic structural time series model with trend (including slope), dummy seasonal with $s = 3$ and irregular. The output for `mphi`, `momega`, `msigma` is given as Φ, Ω, Σ respectively.

3.3. Regression models

The regression model can also be represented as a state space model. The Kalman filter for the regression model in state space form is equivalent to the 'recursive least squares' algorithm for the standard regression model; see Harvey (1993, §4.5). The state space form of the univariate multiple regression model $y_t = X_t\beta + \xi_t$ with $\xi_t \sim \text{NID}\left(0, \sigma_\xi^2\right)$ and the $k \times 1$ vector of coefficients β, for $t = 1, \ldots, n$, is given by:

$$\alpha_{t+1} = \alpha_t, \qquad y_t = X_t\alpha_t + G_t\varepsilon_t, \qquad t = 1, \ldots, n,$$

so that the system matrices are set to $T_t = I_k$, $Z_t = X_t$, $G_t = \sigma_\xi e_1'$, where $e_1' = (1\ 0\ \cdots)$, and $H_t = 0$. The vector of coefficients β is fixed and unknown so that the initial conditions are $\alpha_1 \sim \text{N}(0, \kappa I_k)$ where κ is large. The regression model in state space leads to the so-called marginal or modified-profile likelihood function for σ_ξ^2, which is known to have better small-sample behaviour than the standard concentrated likelihood; see, for example, Tunnicliffe-Wilson (1989, §4.5).

The regression model in state space form implies a time-varying system matrix $Z_t = X_t$ in the measurement equation. Time-varying regression coefficients may be introduced by setting H_t not equal to zero, for $t = 1, \ldots, n$.

SsfPack implementation. The *SsfPack* routine `GetSsfReg` provides the time-varying state space structure for a univariate (single equation) regression model:

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

where `mXt` is a $k \times n$ data matrix containing the explanatory variables. Although the whole X matrix must be given in the function call, internally only information on the number of rows is used. The function returns the composite matrices Φ, Ω, and Σ, as well as the index matrix J_Φ; see §2.4. The index matrix J_Φ refers to the inputted data matrix `mXt`. The structure of the output matrices is clarified in the example below.

Example. The example in Listing 3 outputs the relevant state space matrices for a standard regression model with three explanatory variables. The data matrix consists of a 3×20 matrix of standard normal random numbers.

```
GetSsfReg(rann(3,20), &mphi, &momega, &msigma, &mj_phi);
```

$$\Phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad J_\Phi = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \\ 0 & 1 & 2 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Listing 3. part of ssfreg.ox with corresponding output

*3.4. Nonparametric cubic spline models**

Suppose we work with a continuous variable t for which associated observations $y(t)$ are made at points $t_1, \ldots, t_n$; see the work by Bergstrom (1984). Define $\delta_i = t_{i+1} - t_i$, for $i = 1, \ldots, n$, as the gap between observations, with $\delta_i \geq 0$. The aim is to develop smoothing spline techniques for estimating a signal $\mu(t)$ from observations $y(t)$ via the relationship

$$y(t) = \mu(t) + \varepsilon(t), \tag{15}$$

where $\varepsilon(t)$ is a stationary error process. The task at hand is to find a curve which minimizes $\sum_{i=1}^{n} \{y(t_i) - \mu(t_i)\}^2$ subject to the function $\mu(t)$ being 'smooth'. The common approach is to select the fitted $\hat{\mu}(t)$ by maximizing the penalized Gaussian log-likelihood, that is minimizing

$$\sum_{i=1}^{n} \{y(t_i) - \mu(t_i)\}^2 + q^{-1} \int \left\{ \frac{\partial^2 \mu(t)}{\partial t^2} \right\}^2 \mathrm{d}t, \tag{16}$$

for a given value of q; see Kohn and Ansley (1987), Hastie and Tibshirani (1990) and Green and Silverman (1994).

The penalty function in (16) is equivalent to minus the log density function of the continuous-time Gaussian smooth-trend model for $\mu(t)$, that is

$$\mu(t) = \mu(0) + \int_0^t \beta(s)\mathrm{d}s = \mu(0) + \beta(0)t + \int_0^t W(s)\mathrm{d}s,$$

where the slope $\beta(t)$ is generated by $\mathrm{d}\beta(t) = \mathrm{d}W(t)$, and $W(t)$ is a Brownian motion with variance σ_W^2. The model can be represented as a bivariate Ornstein-Uhlenbeck process for $x(t) = \{\mu(t), \beta(t)\}'$, that is

$$\mathrm{d}x(t) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x(t)\mathrm{d}t + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mathrm{d}W(t),$$

where $\mathrm{d}W(t) \sim \mathrm{N}(0, \sigma_W^2 \mathrm{d}t)$; see, for example, Wecker and Ansley (1983), Kohn and Ansley (1987) and Harvey (1989, 9.1.2 and 9.2.1).

Taking the continuous time process $\mu(t)$ at discrete intervals leads to the following exact discrete time model for $\mu(t_i)$:

$$\begin{aligned} \mu(t_{i+1}) &= \mu(t_i) + \delta_i \beta(t_i) + \eta(t_i), \\ \beta(t_{i+1}) &= \beta(t_i) + \zeta(t_i), \end{aligned} \tag{17}$$

where

$$\begin{aligned} \eta(t_i) &= \mu(t_{i+1}) - \mu(t_i) - \delta_i \beta(t_i) \\ &= \int_{t_i}^{t_{i+1}} \beta(s)\mathrm{d}s - \delta_i \beta(t_i) = \int_{t_i}^{t_{i+1}} \{W(s) - W(t_i)\}\, \mathrm{d}s, \end{aligned}$$

and

$$\zeta(t_i) = W(t_{i+1}) - W(t_i).$$

It follows that

$$\left\{ \begin{array}{c} \eta(t_i) \\ \zeta(t_i) \end{array} \right\} \sim \mathrm{N}\left\{ \left(\begin{array}{c} 0 \\ 0 \end{array} \right), \sigma_\zeta^2 \delta_i \left(\begin{array}{cc} \frac{1}{3}\delta_i^2 & \frac{1}{2}\delta_i \\ \frac{1}{2}\delta_i & 1 \end{array} \right) \right\},$$

where $\sigma_\zeta^2 = \sigma_W^2$. This can be combined with the more straightforward measurement

$$y(t_i) = \mu(t_i) + \varepsilon(t_i),$$

where $\varepsilon(t_i) \sim \mathrm{N}(0, \sigma_\varepsilon^2)$ and is independent of $\eta(t_i)$ and $\zeta(t_i)$. The log-density of the discrete model equals the penalized likelihood (16) with signal-to-noise ratio $q = \sigma_\zeta^2/\sigma_\varepsilon^2$; see Wecker and Ansley (1983). Hence the usual state space framework with

$$\Phi_t = \left(\begin{array}{cc} 1 & \delta_t \\ 0 & 1 \\ 1 & 0 \end{array} \right), \qquad \Omega_t = \left(\begin{array}{ccc} q\delta_t^3/3 & q\delta_t^2/2 & 0 \\ q\delta_t^2/2 & q\delta_t & 0 \\ 0 & 0 & 1 \end{array} \right), \qquad t = 1, \ldots, n,$$

can be used for filtering, smoothing and prediction. Note that, when the observations are equally spaced, δ_t is a constant, and the state space form is time invariant.

SsfPack implementation. The *SsfPack* routine `GetSsfSpline` provides the time-varying state space structure for the cubic spline model (17). The function call is

```
GetSsfSpline(dq, mDelta, &mPhi, &mOmega, &mSigma, &mJ_Phi, &mJ_Omega, &mXt);
```

where `dq` is the signal-to-noise ratio q and `mDelta` is the $1 \times n$ data matrix with δ_t ($\delta_t \geq 0$). The routine returns the state space matrices Φ and Ω together with J_Φ, J_Ω, and the $4 \times n$ data matrix X (see the example below). If `mDelta` is empty, or only the first four arguments are provided, δ_t is assumed to be one, and only Φ and Ω are returned.

Example. The example in Listing 4 outputs the relevant state space matrices for the nonparametric cubic spline model with $q = 0.2$.

```
mt = <2,3,5,9,12,17,20,23,25>;     // t_0 ... t_n
mdelta = diff0(mt', 1)[1:][]';     // delta_1 ... delta_n
GetSsfSpline(0.2, mdelta, &mphi, &momega, &msigma, &mj_phi, &mj_omega, &mx);
```

$$\Phi = \left(\begin{array}{cc} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{array} \right), J_\Phi = \left(\begin{array}{cc} -1 & 0 \\ -1 & -1 \\ -1 & -1 \end{array} \right), \Omega = \left(\begin{array}{ccc} 1/15 & 1/10 & 0 \\ 1/10 & 1/5 & 0 \\ 0 & 0 & 1 \end{array} \right), J_\Omega = \left(\begin{array}{ccc} 3 & 2 & -1 \\ 2 & 1 & -1 \\ -1 & -1 & -1 \end{array} \right),$$

$$X' = \left(\begin{array}{cccc} \delta_1 & q\delta_1 & q\delta_1^2/2 & q\delta_1^3/3 \\ \delta_2 & q\delta_2 & q\delta_2^2/2 & q\delta_2^3/3 \\ \vdots & \vdots & \vdots & \vdots \end{array} \right) = \left(\begin{array}{cccc} 1.0 & 0.2 & 0.1 & 0.0667 \\ 2.0 & 0.4 & 0.4 & 0.5333 \\ 4.0 & 0.8 & 1.6 & 4.2667 \\ 3.0 & 0.6 & 0.9 & 1.8000 \\ 5.0 & 1.0 & 2.5 & 8.3333 \\ 3.0 & 0.6 & 0.9 & 1.8000 \\ 3.0 & 0.6 & 0.9 & 1.8000 \\ 2.0 & 0.4 & 0.4 & 0.5333 \end{array} \right).$$

Listing 4. Part of ssfspl.ox with corresponding output

4. ALGORITHMS

4.1. State space matrices in SsfPack: {Ssf}

In §2.5, we listed four possible formats for specifying the state space form in *SsfPack*:

```
mPhi, mOmega
mPhi, mOmega, mSigma
mPhi, mOmega, mSigma, mDelta
mPhi, mOmega, mSigma, mDelta, mJ_Phi, mJ_Omega, mJ_Delta, mXt
```

where the arguments may be the empty matrix `<>`. In this section we use `{Ssf}` to refer to any of these four forms.

4.2. Simulating from state space models

To generate samples from the unconditional distribution implied by a statistical model in state space form, or to generate artificial data sets, we use the state space form (4) as a recursive set of equations. Actual values for $\alpha_{t+1}^{(i)}$ and $y_t^{(i)}$ for replication (i) can be generated recursively from standard normal random numbers $\varepsilon_t^{(i)}$ using $(H_t', G_t')'\varepsilon_t^{(i)} = u_t^{(i)}$ and:

$$\left\{ \begin{array}{c} \alpha_{t+1}^{(i)} \\ y_t^{(i)} \end{array} \right\} = \delta_t + \Phi_t \alpha_t^{(i)} + u_t^{(i)}, \qquad t = 1, \ldots, n, \tag{18}$$

with the initialization $\alpha_1^{(i)} = a + Qu_0^{(i)}$, where $u_0^{(i)}$ is a vector of standard normal random numbers, and Q is such that $P = QQ'$. The quantities a and Q must be placed in the *SsfPack* matrix Σ:

$$\Sigma = \left(\begin{array}{c} Q \\ a' \end{array} \right).$$

Note that this is different from the usual formulation (7) which is used elsewhere. Only in this particular case Q plays the role of P.

SsfPack implementation. The *SsfPack* function `SsfRecursion` implements the recursion (18) for a given sample of $u_t^{(i)}$ $(t = 0, \ldots, n)$:

```
mD = SsfRecursion(mR, {Ssf});
```

where `mR` is the $(m + N) \times (n + 1)$ data matrix with structure

$$\texttt{mR} = \left(\begin{array}{cccc} u_0^{(i)} & u_1^{(i)} & \ldots & u_n^{(i)} \end{array} \right).$$

Missing values are not allowed in *Ox* is not allowed in `mR`. Although the matrix Ω must be provided as part of {`Ssf`}, it does not play a role in this routine. As pointed out above, Σ should contain Q rather than P. The function `SsfRecursion` returns the $(m + N) \times (n + 1)$ matrix

$$\texttt{mD} = \left(\begin{array}{cccc} \alpha_1^{(i)} & \alpha_2^{(i)} & \ldots & \alpha_{n+1}^{(i)} \\ 0 & y_1^{(i)} & & y_n^{(i)} \end{array} \right).$$

Example. The *Ox* program of Listing 5 generates artificial data from the local linear trend model (5) with $\sigma_\eta^2 = 0$, $\sigma_\zeta^2 = 0.1$ and $\sigma_\xi^2 = 1$. The *Ox* function `rann` produces a matrix of standard normal random deviates. The initial state vector $\alpha_1 = (\mu_1, \beta_1)'$ is set equal to $(1, 0.5)'$.

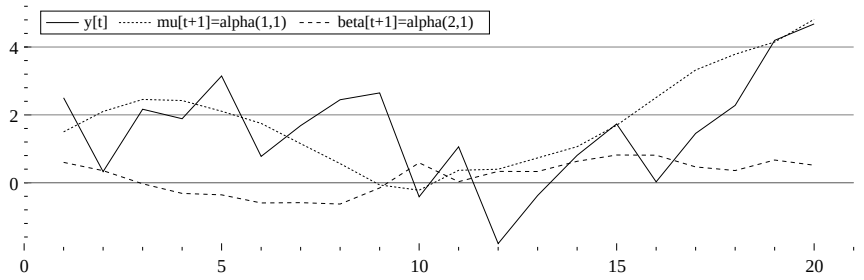
```
#include <oxdraw.h>
#include <packages/ssfpack/ssfpack.h>

main()
{
    decl mphi = <1,1;0,1;1,0>;
    decl momega = diag(<0,0.1,1>);
    decl msigma = <0,0;0,0;1,.5>;       // Note that Q is zero

    decl mr = sqrt(momega) * rann(3, 21);
    decl md = SsfRecursion(mr, mphi, momega, msigma);
    decl myt = md[2][1:];               // 20 observations

    print("Generated data (t=10)",
        "%c", {"alpha(1,1)[t+1]","alpha(2,1)[t+1]","y[t]"}, md[][10]');
    DrawTMatrix(0, myt | md[:1][1:],
        {"y[t]", "mu[t+1]=alpha(1,1)","beta[t+1]=alpha(2,1)"}, 1, 1, 1);
    ShowDrawWindow();
}
```

```
Generated data (t=10)
alpha(1,1)[t+1]  alpha(2,1)[t+1]           y[t]
       -0.22002          0.58665       -0.41452
```



Data generated by ssfrec.ox

Listing 5. ssfrec.ox with output

4.3. Kalman filter

The Kalman filter is a recursive algorithm for the evaluation of moments of the normal distribution of state vector α_{t+1} conditional on the data set $Y_t = \{y_1, \ldots, y_t\}$, that is

$$a_{t+1} = \mathrm{E}\left(\alpha_{t+1}|Y_t\right), \qquad P_{t+1} = \mathrm{cov}\left(\alpha_{t+1}|Y_t\right),$$

for $t = 1, \ldots, n$; see Anderson and Moore (1979, page 36) and Harvey (1989, page 104). The Kalman filter is given by (with dimensions in parentheses):

$$\begin{array}{lcll} v_t & = & y_t - c_t - Z_t a_t & (N \times 1) \\ F_t & = & Z_t P_t Z_t' + G_t G_t' & (N \times N) \\ K_t & = & (T_t P_t Z_t' + H_t G_t')\, F_t^{-1} & (m \times N) \\ a_{t+1} & = & d_t + T_t a_t + K_t v_t & (m \times 1) \\ P_{t+1} & = & T_t P_t T_t' + H_t H_t' - K_t F_t K_t' & (m \times m) \end{array} \tag{19}$$

where $a_1 = a$, and $P_1 = P_* + \kappa P_\infty$ with $\kappa = 10^7$, for $t = 1, \ldots, n$.

Missing values. In §2.6 it was shown how missing values are deleted internally to create $y_t^\dagger, c_t^\dagger, Z_t^\dagger, G_t^\dagger$. Consequently, when missing values are present, the Kalman filter at time t are based on $y_t^\dagger$ instead of y_t. The smoothers which are to be introduced in the next sections are adjusted accordingly.

When the full vector y_t is missing, for example when a single observation is missing in univariate cases, the Kalman filter reduces to a prediction step, that is

$$a_{t+1} = d_t + T_t a_t, \qquad P_{t+1} = T_t P_t T_t' + H_t H_t',$$

such that $v_t = 0$, $F_t^{-1} = 0$ and $K_t = 0$. The moment and simulation smoother deal with these specific values of v_t, F_t^{-1} and K_t without further complications. See the example in §5.3.

Algorithm*. The *SsfPack* implementation for the Kalman filter is written in a computationally efficient way. The steps are given by

(i) Set $t = 1$, $a_1 = a$ and $P_1 = P_* + 10^7 P_\infty$.

(ii) Calculate:

$$\begin{pmatrix} \bar{a}_{t+1} \\ \hat{y}_t \end{pmatrix} = \delta_t + \Phi_t a_t, \qquad \begin{pmatrix} \bar{P}_{t+1} & M_t \\ M_t' & F_t \end{pmatrix} = \Phi_t P_t \Phi_t' + \Omega_t, \qquad K_t = M_t F_t^{-1},$$

where δ_t, Φ_t and Ω_t are defined in (4), and $M_t = (T_t P_t Z_t' + H_t G_t')$.

(iii) Update:

$$v_t = y_t - \hat{y}_t, \qquad a_{t+1} = \bar{a}_{t+1} + K_t v_t, \qquad P_{t+1} = \bar{P}_{t+1} - K_t M_t'.$$

(iv) If $t = n$ then stop, else set $t = t + 1$ and go to (ii).

The program stops with an error message when $|F_t| \leq 0$ or when insufficient computer memory is available.

SsfPack implementation. The *SsfPack* function `KalmanFil` calls the Kalman filter and returns the output v_t, F_t and K_t ($t = 1, \ldots, n$) as a data matrix:

```
mKF = KalmanFil(mYt, {Ssf});
```

where `mYt` is an $N \times n$ data matrix. The Kalman filter is available for univariate and multivariate state space models: the row dimension of `mYt` determines whether the univariate or the multivariate Kalman filter is used. The function returns a matrix `mKF` with dimension $q \times n$ where

$$q = N + mN + \frac{N\,(N+1)}{2}$$

consists of the number of unique elements in v_t, K_t, and F_t^{-1} respectively. For univariate models in state space form, the returned storage matrix is simply the $(m+2) \times n$ matrix

$$\texttt{mKF} = \begin{bmatrix} v_1 & \dots & v_n \\ (K_{11})_1 & \dots & (K_{11})_n \\ \vdots & & \vdots \\ (K_{m1})_1 & \dots & (K_{m1})_n \\ F_1^{-1} & \dots & F_n^{-1} \end{bmatrix}.$$

In multivariate cases, the returned data matrix is organized as

$$\texttt{mKF} = \begin{bmatrix} v_1 & \dots & v_n \\ (K_{*1}^{-1})_1 & \dots & (K_{*1}^{-1})_n \\ (F_{*1}^{-1})_1 & \dots & (F_{*1}^{-1})_n \\ \vdots & & \vdots \\ (K_{*N}^{-1})_1 & \dots & (K_{*N}^{-1})_n \\ (F_{*N}^{-1})_1 & \dots & (F_{*N}^{-1})_n \end{bmatrix}.$$

Here we write $(K_{*j})_t$ for column j of K_t, which has m elements; $(F_{*j}^{-1})_t$ refers to column j of F_t^{-1} with the lower diagonal discarded: $(F_{*1}^{-1})_t$ has 1 element, and $(F_{*N}^{-1})_t$ has N elements.

Example. The *Ox* code on the next page (Listing 6) applies the Kalman filter to the data `myt` generated in Listing 5.

4.4. Moment smoothing

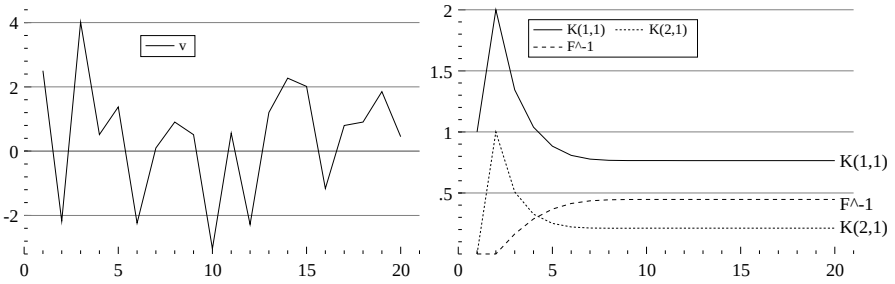
The Kalman filter is a forward recursion which evaluates one-step ahead estimators. The associated moment smoothing algorithm is a backward recursion which evaluates the mean and variance of specific conditional distributions given the data set $Y_n = \{y_1, \dots, y_n\}$ using the output of the Kalman filter; see Anderson and Moore (1979), Kohn and Ansley (1989), de Jong (1988b), de Jong (1989) and Koopman (1993). The backward recursions are given by

$$\begin{array}{lcll} e_t & = & F_t^{-1} v_t - K_t' r_t & (N \times 1) \\ D_t & = & F_t^{-1} + K_t' N_t K_t & (N \times N) \\ r_{t-1} & = & Z_t' F_t^{-1} v_t + L_t' r_t & (m \times 1) \\ N_{t-1} & = & Z_t' F_t^{-1} Z_t + L_t' N_t L_t & (m \times m) \end{array} \tag{20}$$

with $L_t = T_t - K_t Z_t$ and with the initialization $r_n = 0$ and $N_n = 0$, for $t = n, \dots, 1$.

```
decl mkf = KalmanFil(myt, mphi, momega);
print("mKF\' (t=10)", "%c", {"v", "K(1,1)", "K(2,1)", "F^-1"}, mkf[][9]');
DrawTMatrix(0, mkf[0][], {"v"}, 1, 1, 1);
DrawTMatrix(1, mkf[1:][], {"K(1,1)", "K(2,1)", "F^-1"}, 1, 1, 1);
ShowDrawWindow();
```

```
mKF' (t=10)
             v       K(1,1)       K(2,1)         F^-1
      -3.0347      0.76491      0.21161      0.44669
```



Output generated by ssfkf.ox

Listing 6. Part of ssfkf.ox with output

Disturbance smoothing. The moment smoother (20) generates quantities from which different kinds of estimators can be obtained. For example, it can be shown that the mean and variance of the conditional density $f(\varepsilon_t|Y_n)$ is given by, respectively,

$$\begin{aligned} \mathrm{E}(\varepsilon_t|Y_n) &= G_t' e_t + H_t' r_t, \\ \mathrm{var}(\varepsilon_t|Y_n) &= G_t'(D_t G_t - K_t' N_t H_t) + H_t'(N_t H_t - N_t K_t G_t), \end{aligned}$$

and expressions for $\mathrm{E}(u_t|Y_n)$ and $\mathrm{var}(u_t|Y_n)$, where u_t is defined in (4), follow directly from this. It is also clear that, when $H_t G_t' = 0$,

$$\begin{aligned} \mathrm{E}(H_t\varepsilon_t|Y_n) &= H_t H_t' r_t, \\ \mathrm{var}(H_t\varepsilon_t|Y_n) &= H_t H_t' N_t H_t H_t', \\ \mathrm{E}(G_t\varepsilon_t|Y_n) &= G_t G_t' e_t, \\ \mathrm{var}(G_t\varepsilon_t|Y_n) &= G_t G_t' D_t G_t G_t', \end{aligned}$$

for $t = 1, \ldots, n$; see Koopman (1993) for more general results. In these computations r_0 and N_0 are not used, although the are calculated in (20).

Algorithm*. The *SsfPack* implementation for the moment smoother is similar to the Kalman filter:

(i) Set $t = n$, $r_n = 0$ and $N_n = 0$.

(ii) Calculate:

$$r_t^* = \begin{pmatrix} r_t \\ e_t = F_t^{-1} v_t - K_t' r_t \end{pmatrix}, \qquad N_t^* = \begin{pmatrix} N_t & -N_t K_t \\ -K_t' N_t & D_t = F_t^{-1} + K_t' N_t K_t \end{pmatrix}.$$

(iii) Update:

$$r_{t-1} = \Phi_t' r_t^*, \qquad N_{t-1} = \Phi_t' N_t^* \Phi_t.$$

where Φ_t is defined in (4).

(iv) If $t = 1$ then stop, else set $t = t - 1$ and go to (ii).

The program stops with an error message when insufficient memory is available. The vector δ_t and the matrix Ω_t do not play a role in the basic smoothing recursions. Finally, it should be noted that the smoothed estimator $\hat{u}_t = \mathrm{E}(u_t|Y_n)$, where u_t is from (4), is simply obtained by $\Omega_t r_t^*$; the corresponding variance matrix is $\mathrm{var}(u_t|Y_n) = \Omega_t N_t^* \Omega_t$; see §5.3 for further details.

Quick state smoothing. The generated output from the basic smoothing recursions can also be used to obtain $\hat{\alpha}_t = \mathrm{E}(\alpha_t|Y_n)$, that is, the smoothed estimator of the state vector, using the recursion

$$\hat{\alpha}_{t+1} = d_t + T_t \hat{\alpha}_t + H_t \hat{\varepsilon}_t, \qquad t = 1, \ldots, n,$$

with $\hat{\alpha}_1 = a + Pr_0$ and $\hat{\varepsilon}_t = \mathrm{E}(\varepsilon_t|Y_n) = G_t' e_t + H_t' r_t$; see Koopman (1993) for details. This simple recursion is similar to the state space recursion (18), and therefore we can trick `SsfRecursion` into generating α_{t+1} (but note that here Σ contains P in the standard way, and not Q). A further discussion on state smoothing is found in §5.3 (with examples) and §4.6. This method of state smoothing is illustrated in the example below using the *SsfPack* function `SsfRecursion`.

SsfPack implementation. The *SsfPack* function `KalmanSmo` implements the moment smoother and stores the output e_t, D_t, r_{t-1} and N_{t-1} for $t = 1, \ldots, n$, into a data matrix:

```
mKS = KalmanSmo(mKF, {Ssf});
```

The input matrix `mKF` is the data matrix which is produced by the function `KalmanFil` using the same state space form `{Ssf}`. The return value `mKS` is a data matrix of dimension $2(m + N) \times (n + 1)$. The structure of the matrix is

$$\texttt{mKS} = \begin{bmatrix} r_0 & r_1 & \ldots & r_n \\ 0 & e_1 & \ldots & e_n \\ \mathrm{diag}(N_0) & \mathrm{diag}(N_1) & \ldots & \mathrm{diag}(N_n) \\ 0 & \mathrm{diag}(D_1) & \ldots & \mathrm{diag}(D_n) \end{bmatrix},$$

where $\mathrm{diag}(A)$ vectorizes the diagonal elements of the square matrix A. The number of elements in r_t, e_t, $\mathrm{diag}(N_t)$, and $\mathrm{diag}(D_t)$ is respectively: m, N, m, N. The output matrix is organised in this way partly because the first $(m + N)$ rows of `mKS` can be used as input to `SsfRecursion`, as discussed above. More elaborate and more 'easy-to-use' functions for moment smoothing of the disturbance and state vector are given in §4.6.

Example. The following *Ox* code (Listing 7) applies the Kalman filter smoother to the results from Listing 6. It outputs the matrix `mKS`, the smoothed disturbances, and smoothed states.

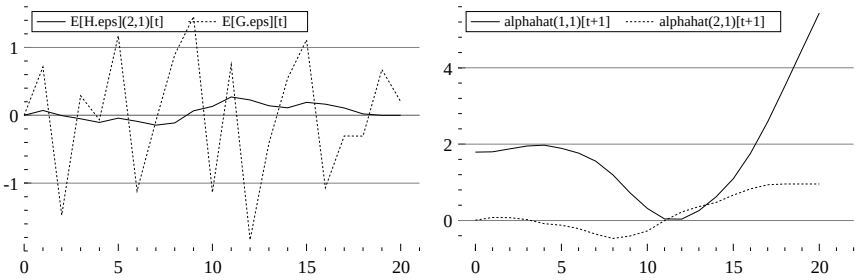
```
decl mks = KalmanSmo(mkf, mphi, momega);
print("Basic smoother output: mKS\' (t=10)",
    "%c", {"r","e(1,1)","e(2,1)","N","D(1,1)","D(2,2)"}, mks[][10]');

decl msmodist = mks[0:2][0] ~ momega * mks[0:2][1:];
print("Smoothed disturbances (t=10)",
    "%c", {"E[H.eps](1,1)","E[H.eps](2,1)","E[G.eps]"}, msmodist[][10]');

decl msmostat = SsfRecursion(msmodist, mphi, momega);
print("Smoothed states (t=10)", "%c",
    {"alphahat(1,1)[t+1]","alphahat(2,1)[t+1]","y[t]"}, msmostat[][10]');

DrawTMatrix(0, msmodist[1:2][],
    {"E[H.eps](2,1)[t]","E[G.eps][t]"}, 0, 1, 1);
DrawTMatrix(1, msmostat[0:1][],
    {"alphahat(1,1)[t+1]","alphahat(2,1)[t+1]"}, 0, 1, 1);
ShowDrawWindow();
```

```
Basic smoother output: mKS' (t=10)
            r       e(1,1)       e(2,1)            N       D(1,1)       D(2,2)
     -0.64966       1.3099      -1.1358      0.60208       2.0578      0.79365
Smoothed disturbances (t=10)
     E[H.eps](1,1)       E[H.eps](2,1)       E[G.eps]
           0.00000             0.13099        -1.1358
Smoothed states (t=10)
alphahat(1,1)[t+1]  alphahat(2,1)[t+1]           y[t]
           0.31542             -0.27487       -0.41452
```



Output generated by ssfsmo.ox

Listing 7. Part of ssfsmo.ox with output

4.5. Simulation smoother

Disturbance simulation smoothing. The simulation smoother is developed by de Jong and Shephard (1995) and allows drawing random numbers from the multivariate conditional Gaussian density of

$$\tilde{u} = (\tilde{u}_1', \ldots, \tilde{u}_n')', \qquad \text{where } \tilde{u} \sim \Gamma u | Y_n, \qquad t = 1, \ldots, n, \tag{21}$$

with $u = (u_1', \ldots, u_n')'$ and u_t as defined in (4). The $(m+N) \times (m+N)$ diagonal selection matrix Γ consists of unity and zero values on the diagonal. It is introduced to avoid degeneracies in sampling and, and to allow generating subsets of u_t, which is more efficient, especially when the state vector is large and only a small subset is required.

For example, when we consider the local linear trend model (5) and wish to generate samples (for $t = 1, \ldots, n$) from the multivariate conditional density of the disturbance ζ_t, then:

$$\Gamma = \text{diag}\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}.$$

In order to generate samples from the multivariate joint conditional density of η_t and ζ_t for this model:

$$\Gamma = \text{diag}\begin{pmatrix} 1 & 1 & 0 \end{pmatrix}.$$

Generating conditional samples for $G_t \varepsilon_t$ of the state space form, which for univariate cases requires

$$\Gamma = \text{diag}\begin{pmatrix} 0 & \ldots & 0 & 1 \end{pmatrix},$$

also implicitly produces samples from $f(\theta_t | Y_n)$, with signal $\theta_t = c_t + Z_t \alpha_t$, since $y_t - G_t \varepsilon_t = \theta_t$.

The simulation algorithms use the $s \times (m+N)$ zero-unity matrix Γ^* which is the same as Γ but where the zero rows are deleted from Γ. For example,

$$\Gamma = \text{diag}\begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \text{ becomes } \Gamma^* = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } \Gamma^{*\prime}\Gamma^* = \Gamma,$$

with $s = 2$ in this case.

The simulation smoother is a backward recursion and requires the output of the Kalman filter. The equations are given by

$$\begin{aligned}
C_t &= \Gamma^* \begin{pmatrix} H_t \\ G_t \end{pmatrix} \left(I - G_t' F_t^{-1} G_t - J_t' N_t J_t\right) \begin{pmatrix} H_t \\ G_t \end{pmatrix}' \Gamma^{*\prime} && (s \times s) \\
W_t &= \Gamma^* \begin{pmatrix} H_t \\ G_t \end{pmatrix} \left(G_t' F_t^{-1} Z_t + J_t' N_t L_t\right), \quad \xi_t \sim \text{N}(0, C_t) && (s \times m) \\
r_{t-1} &= Z_t' F_t^{-1} v_t - W_t' C_t^{-1} \xi_t + L_t' r_t, && (m \times 1) \\
N_{t-1} &= Z_t' F_t^{-1} Z_t + W_t' C_t^{-1} W_t + L_t' N_t L_t && (m \times m)
\end{aligned} \tag{22}$$

where $L_t = T_t - K_t Z_t$ and $J_t = H_t - K_t G_t$, for $t = n, \ldots, 1$. The initialization is $r_n = 0$ and $N_n = 0$. The notation for r_t and N_t is the same as for the moment smoother (20)

since the nature of both recursions is very similar. However, their actual values are different. It can be shown that

$$\tilde{u}_t = \Gamma^{*\prime}\left\{\Gamma^* \begin{pmatrix} H_t \\ G_t \end{pmatrix}\left(G_t' F_t^{-1} v_t + J_t' r_t\right) + \xi_t\right\}, \tag{23}$$

is a draw as indicated by (21). The selection matrix Γ must be chosen so that $\Gamma^*\Omega_t\Gamma^{*\prime}$ is nonsingular and $\text{rank}(\Gamma^*\Omega_t\Gamma^{*\prime}) \leq m$; the latter condition is required to avoid degenerate sampling and matrix C_t being singular. These conditions are not sufficient to avoid degenerate sampling; see de Jong and Shephard (1995). However, the conditions firmly exclude the special case of $\Gamma = I_{m+N}$.

Algorithm*. The structure of the *SsfPack* implementation for the simulation smoother is similar to the moment smoother. In the following we introduce the $s \times m$ matrix $A_t = C_t^{-\frac{1}{2}} W_t$. The steps of the program are given by

(i) Set $t = n$, $r_n = 0$ and $N_n = 0$.
(ii) Calculate:

$$r_t^* = \begin{pmatrix} r_t \\ e_t = F_t^{-1} v_t - K_t' r_t \end{pmatrix}, \qquad N_t^* = \begin{pmatrix} N_t & -N_t K_t \\ -K_t' N_t & D_t = F_t^{-1} + K_t' N_t K_t \end{pmatrix}.$$

(iii) Calculate:

$$C_t = \Gamma^* \left(\Omega_t - \Omega_t N_t^* \Omega_t\right) \Gamma^{*\prime},$$

apply a Choleski decomposition to C_t such that

$$C_t = B_t B_t',$$

and solve recursively with respect to A_t:

$$B_t A_t = \Gamma^* \Omega_t N_t^*.$$

The matrices Φ_t and Ω_t are defined in (4).
(iv) Update:

$$r_{t-1} = \Phi_t' \left(r_t^* - A_t' \pi_t\right), \qquad N_{t-1} = \Phi_t' \left(N_t^* + A_t' A_t\right) \Phi_t.$$

with $\pi_t \sim \text{N}\left(0, I_s\right)$.
(v) If $t = 1$ then stop, else set $t = t - 1$ and go to (ii).

The program stops with an error message when the Choleski decomposition for C_t fails or when insufficient memory is available. The vector δ_t does not play a role in simulation smoothing.

Generating multiple samples. A draw from the Gaussian density for (21) is obtained by (23), which can be written as:

$$\tilde{u}_t = \Gamma^{*\prime}\left(\Gamma^* \Omega_t r_t^* + B_t \pi_t\right), \qquad t = 1, \ldots, n.$$

When M different samples are required from the same model and conditional on the same data-set Y_n, the simulation smoother can be simplified to generate multiple draws.

The matrices A_t and B_t (the so-called weights) need to be stored; now M samples can be generated via the recursion:

$$\begin{array}{rcll} r_{t-1} & = & \Phi_t' \left(r_t^* - A_t' \pi_t^{(i)} \right), & \pi_t^{(i)} \sim \mathrm{N}\left(0, I\right), \\ \tilde{u}_t^{(i)} & = & \Gamma^{*\prime} \left(\Gamma^* \Omega_t r_t^* + B_t \pi_t^{(i)} \right), & t = n, \ldots, 1, \quad i = 1, \ldots, M \end{array} \tag{24}$$

which is computationally efficient. Note that we omitted the superscript $^{(i)}$ from r_t, r_t^*, and that $r_t^* = (r_t', e_t')'$; see step (ii) of the algorithm. When $s = 1$, the storage of A_t and B_t $(t = 1, \ldots, n)$ requires a matrix of dimension $(1 + m + N) \times n$.

State simulation smoothing. As mentioned earlier, generated samples from the simulation smoother (22) can be used to get simulation samples from the multivariate density $f\left(\theta|Y_n\right)$, where $\theta = (\theta_1', \ldots, \theta_n')'$ and $\theta_t = c_t + Z_t \alpha_t$, by setting Γ such that $\Gamma^* \left(H_t', G_t'\right)' = G_t^*$, for $t = 1, \ldots, n$, and where G_t^* is equal to G_t but without the zero rows (in the same spirit of Γ and Γ^*). This follows from the identity $\theta_t = y_t - G_t \varepsilon_t$. In a similar way, it is also possible to obtain samples from the multivariate density $f\left(\alpha|Y_n\right)$, where $\alpha = (\alpha_1', \ldots, \alpha_n')'$, by applying the simulation smoother (22) with Γ such that $\Gamma^* \left(H_t', G_t'\right)' = H_t^*$, for $t = 1, \ldots, n$, and where H_t^* is H_t but without the zero rows. Then the generated sample $\tilde{u}_t$ $(t = 1, \ldots, n)$ is inputted into the state space recursion (18) with initialization $\alpha_1^{(i)} = a + P r_0^{(i)}$; see de Jong and Shephard (1995) for details. In this way a sample from $f\left(\theta|Y_n\right)$ can also be obtained but now via the identity $\theta_t^{(i)} = c_t + Z_t \alpha_t^{(i)}$ (rather than $\theta_t^{(i)} = y_t - \{G_t \varepsilon_t\}^{(i)}$) so that this sample is consistent with the sample from $f\left(\alpha|Y_n\right)$. Note that sampling directly from $f\left(\alpha, \theta|Y_n\right)$ is not possible because of degeneracies; this matter is further discussed in §4.6. A simple illustration is given by the example below.

SsfPack implementation. The *SsfPack* function `SimSmoWgt` implements the simulation smoother, but only for C_t, W_t and N_t. It stores the output $A_t = B_t^{-1} W_t$ and B_t (remember that $C_t = B_t B_t'$), for $t = 1, \ldots, n$, into a data matrix. The call is given by

```
mWgt = SimSmoWgt(mGamma, mKF, {Ssf});
```

where `mGamma` is the $m+N$ diagonal 'selection' matrix Γ and `mKF` is the data matrix which is produced by the function `KalmanFil` for the same state space form implied by `{Ssf}`. The return value `mWgt` is a data matrix of dimension $q \times n$ where $q = s(m+N)+s(s+1)/2$, and the structure of the matrix is

$$\texttt{mWgt} = \begin{pmatrix} \mathrm{vec}\left(A_1\right) & \ldots & \mathrm{vec}\left(A_n\right) \\ \mathrm{vech}\left(B_1\right) & \ldots & \mathrm{vech}\left(B_n\right) \end{pmatrix}$$

where $\mathrm{vec}(A_t)$ vectorizes matrix A_t, resulting in $s(m + N)$ elements, and $\mathrm{vech}(B_t)$ vectorizes the lower triangular part (including its diagonal) of matrix B_t, giving s(s+1)/2 elements.

The *SsfPack* function `SimSmoDraw` generates a sample from the distribution (21) which is calculated by the equations (24). This function requires the weight matrices A_t and B_t for $t = 1, \ldots, n$. The function call is given by

```
mD = SimSmoDraw(mGamma, mPi, mWgt, mKF, {Ssf});
```

where `mGamma` is the diagonal 'selection' matrix Γ, `mPi` is an $s \times n$ data matrix containing the random deviates from the standard normal distribution, matrix `mWgt` is the matrix obtained from function `SimSmoWgt`, matrix `mKF` is the matrix returned by the function `KalmanFil`. The `SimSmoDraw` function returns the $(m+N) \times (n+1)$ matrix `mD` where

$$\texttt{mD} = \begin{pmatrix} r_0^* & \tilde{u}_1 & \dots & \tilde{u}_n \end{pmatrix}.$$

where $r_0^{*\prime} = (r_0', 0')$ and $\tilde{u}_t$ is defined in (21). Repeated samples can be generated consecutively; see example below. The return value `mD` is constructed such that it can be used as the input matrix `mR` for the *SsfPack* function `SsfRecursion` which enables state simulation samples, as illustrated in the next example.

Example. The *Ox* program in Listing 8 draws from the multivariate conditional Gaussian density $f(\zeta|Y_n)$, with $\zeta = (\zeta_1, \dots, \zeta_n)'$, of the local linear trend model (5) used in Listings 5–7 ($\sigma_\eta^2 = 0$, $\sigma_\zeta^2 = 0.1$, $\sigma_\varepsilon^2 = 1$). This draw is also used to generate samples from the densities $f(\alpha|Y_n)$ and $f(\theta|Y_n)$. Note that Γ is selected such that $\Gamma^* (H', G')' = H$ but without the zero rows. Thus $\Gamma = \mathrm{diag}(0, 1, 0)$ because $\sigma_\eta^2 = 0$, so $\Gamma^* = (0, 1, 0)$ and therefore $s = 1$. Three drawings are shown in Figure 2.

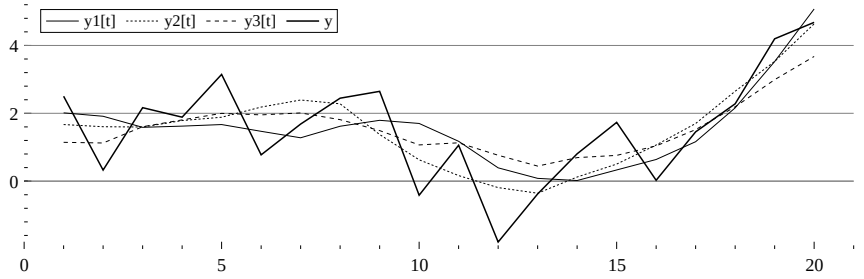


Figure 2. Graphical output generated by ssfsim.ox

4.6. The conditional density: mean calculation and simulation

Efficient methods are provided to estimate the mean of, and draw random numbers from, the conditional density of the state and disturbance vector (given the observations). These can be used when only signal estimation and simulation is required; §5 provides more general functions.

Mean calculation of states. When only the mean of the multivariate conditional density $f(\alpha_1, \dots, \alpha_n|Y_n)$, *i.e.* the smoothed state vector $\hat{\alpha}_t = \mathrm{E}(\alpha_t|Y_n)$, is required, the following simple recursion can be used:

$$\hat{\alpha}_{t+1} = d_t + T_t \hat{\alpha}_t + H_t \hat{\varepsilon}_t, \qquad t = 1, \dots, n-1,$$

with $\hat{\alpha}_1 = a + Pr_0$ and $\hat{\varepsilon}_t = \mathrm{E}(\varepsilon_t|Y_n) = H_t' r_t + G_t' e_t$; see Koopman (1993) and §4.4. The smoothing quantities e_t and r_t are obtained from (20). This algorithm is computationally more efficient, and avoids storage of a_t and P_t, $t = 1, \dots, n$, as required in the general state moment smoothing algorithm.

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

main()
{
    decl mphi = <1,1;0,1;1,0>;
    decl momega = diag(<0,0.1,1>);
    decl msigma = <0,0;0,0;1,.5>;

    decl md = SsfRecursion(sqrt(momega) * rann(3, 21), mphi, momega, msigma);
    decl myt = md[2][1:];
    decl mkf = KalmanFil(myt, mphi, momega);

    decl ct = columns(myt); // 20 observations
    decl mgamma = diag(<0,1,0>);
    decl mwgt = SimSmoWgt(mgamma, mkf, mphi, momega);
    print("Simulation smoother weights (t=10)",
        "%c", {"A(1,1)","A(1,2)","A(1,3)","B(1,1)"}, mwgt[][9]');

    // draw 1
    md = SimSmoDraw(mgamma, rann(1, ct), mwgt, mkf, mphi, momega);
    print("Draw 1 for slope disturbances (t=10)",
        "%c", {"H.eps(1,1)","H.eps(2,1)","G.eps"}, md[][10]');
    md = SsfRecursion(md, mphi, momega);
    print("Draw 1 for state and signal (t=10)",
        "%c", {"alpha(1,1)[t+1]","alpha(2,1)[t+1]","y[t]"}, md[][10]');
    // draw 2
    decl md2 = SimSmoDraw(mgamma, rann(1, ct), mwgt, mkf, mphi, momega);
    md2 = SsfRecursion(md2, mphi, momega);
    // draw 3
    decl md3 = SimSmoDraw(mgamma, rann(1, ct), mwgt, mkf, mphi, momega);
    md3 = SsfRecursion(md3, mphi, momega);

    DrawTMatrix(0, md[2][1:] | md2[2][1:] | md3[2][1:] | myt,
        {"y1[t]","y2[t]","y3[t]", "y"}, 1, 1, 1);
    ShowDrawWindow();
}
```

```
Simulation smoother weights (t=10)
         A(1,1)           A(1,2)         A(1,3)         B(1,1)
       -0.40350           1.5248      -0.014001        0.24905
Draw 1 for slope disturbances (t=10)
     H.eps(1,1)       H.eps(2,1)          G.eps
        0.00000         -0.25514        0.00000
Draw 1 for state and signal (t=10)
alpha(1,1)[t+1]    alpha(2,1)[t+1]          y[t]
         1.1744         -0.78113          1.7004
```

Listing 8. ssfsim.ox with output

Simulation for states. The simulation smoother can also generate simulations from $f(\alpha|Y_n)$, where $\alpha' = (\alpha_1', \ldots, \alpha_n')$, for a given model in state space form; see de Jong and Shephard (1995) and §4.5. The simulations are denoted by $\alpha^{(i)\prime} = (\alpha_1^{(i)\prime}, \ldots, \alpha_n^{(i)\prime})$. The simulation smoother, with an appropriate choice of the selection matrix, outputs the draws $H_1\varepsilon_1^{(i)}, \ldots, H_n\varepsilon_n^{(i)}$ from which the simulated states can be obtained via the recursion

$$\alpha_{t+1}^{(i)} = d_t + T_t\alpha_t^{(i)} + H_t\varepsilon_t^{(i)}, \qquad t = 1, \ldots, n,$$

with the initialization $\alpha_1^{(i)} = a + Pr_0$, where r_0 is obtained from the simulation smoother (22). Consistent simulations for the signal θ_t are obtained via the relation $\theta_t^{(i)} = c_t + Z_t\alpha_t^{(i)}$, for $t = 1, \ldots, n$. Note that, when no consistency is required between $\theta^{(i)}$ and $\alpha^{(i)}$, it is easier to obtain simulation samples using $\theta_t^{(i)} = y_t - G_t\varepsilon_t^{(i)}$; see §4.5.

Mean calculation of disturbances. The mean of the multivariate conditional density $f(u_1, \ldots, u_n|Y_n)$, where $u_t = (H_t', G_t')'\,\varepsilon_t$ as defined in (4), is denoted by $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_n)$ and its calculation is discussed in §4.4 and §5.3.

Simulation for disturbances. Generating samples from $f(u|Y_n)$ for a given model in state space form is done via the simulation smoother; the details are given in §4.5. As pointed out by de Jong and Shephard (1995), the simulation smoother cannot draw from $f(u|Y_n)$ directly because of the implied identities within the state space form (4); this problem is referred to as degenerate sampling. However it can simulate from $f(H_1\varepsilon_1, \ldots, H_n\varepsilon_n|Y_n)$ directly and then compute the sample $\theta^{(i)}$ as discussed under 'Simulation for states' above. The identity $G_t\varepsilon_t = y_t - \theta_t$ allows the generation of simulation samples from $f(G_1\varepsilon_1, \ldots, G_n\varepsilon_n|Y_n)$ which are consistent with the sample from $f(H_1\varepsilon_1, \ldots, H_n\varepsilon_n|Y_n)$. Finally, when the rank of H_t is smaller than G_t the described method of getting simulations from $f(u|Y_n)$ is not valid. In that case, the simulation smoother should be applied directly as described in §4.5.

SsfPack implementation. The *SsfPack* call for calculating mean and simulation for the multivariate conditional densities of the disturbances and the states is given by

```
mD = SsfCondDens(iSel, mYt, {Ssf});
```

where the structure of the output matrix `mD` depends on the value of `iSel` which must be one of the predefined constants:

`iSel`	computes	`mD` =
`ST_SMO`	mean of $f(\alpha\|Y_n)$;	$\begin{bmatrix} \hat{\alpha}_1 & \ldots & \hat{\alpha}_n \\ \hat{\theta}_1 & \ldots & \hat{\theta}_n \end{bmatrix}$,
`ST_SIM`	simulation sample from $f(\alpha\|Y_n)$;	$\begin{bmatrix} \alpha_1^{(i)} & \ldots & \alpha_n^{(i)} \\ \theta_1^{(i)} & \ldots & \theta_n^{(i)} \end{bmatrix}$,
`DS_SMO`	mean of $f(u\|Y_n)$;	$\begin{bmatrix} \hat{u}_1 & \ldots & \hat{u}_n \end{bmatrix}$,
`DS_SIM`	simulation sample from $f(u\|Y_n)$.	$\begin{bmatrix} u_1^{(i)} & \ldots & u_n^{(i)} \end{bmatrix}$.

Here $\hat{\theta}_t = c_t + Z_t\hat{\alpha}_t$ is the smoothed estimate of the signal $c_t + Z_t\alpha_t$ and $\theta_t^{(i)}$ is the associated simulation. The inputs `mYt` and `{Ssf}` are as usual. An application is given in the next section.

5. USING SSFPACK IN PRACTICE

5.1. Likelihood and score evaluation for general models

The Kalman filter allows the computation of the Gaussian log-likelihood function via the prediction error decomposition; see Schweppe (1965), Jones (1980) and Harvey (1989, §3.4). The log-likelihood function is given by

$$\begin{aligned} l = \log p\left(y_1, \ldots, y_n; \varphi\right) &= \sum_{t=1}^{n} \log p\left(y_t | y_1, \ldots, y_{t-1}; \varphi\right) \\ &= -\frac{nN}{2}\log\left(2\pi\right) - \frac{1}{2}\sum_{t=1}^{n}\left(\log|F_t| + v_t' F_t^{-1} v_t\right) \end{aligned} \tag{25}$$

where φ is the vector of parameters for a specific statistical model represented in state space form (19). The innovations v_t and its variances F_t are computed by the Kalman filter for a given vector φ.

In the remainder we require d, defined as the number of elements in the state vector which have a diffuse initial distribution. Usually, d is the number of nonstationary elements and fixed regression effects in the state vector. In terms of the initial variance matrix Σ, d is the number of diagonal elements of Σ which are set equal to -1; see §2.3. If -1 is not used to indicate diffuse elements, d will be zero in the *SsfPack* computations. In subsequent output where d is involved, explicit adjustment must be made afterwards. Note that the summation in (25) is from 1 to n, but the first d summations will be approximately zero as F_t^{-1} will be very small for $t = 1, \ldots, d$.

SsfPack output includes the scale factor

$$\hat{\sigma}^2 = \frac{1}{Nn - d}\sum_{t=1}^{n} v_t' \left(F_t\right)^{-1} v_t. \tag{26}$$

When starting the iterative optimization of the log-likelihood, it can be helpful to choose starting values such that initially $\hat{\sigma}^2 \approx 1$; see the example below. In general, after likelihood estimation, $\hat{\sigma}^2$ will be equal, or close to, one.

The score vector for Gaussian models in state space form is usually evaluated numerically. Koopman and Shephard (1992) present a method to calculate the exact score for any parameter within the system matrices T, Z, H and G. Let the ith element of φ, that is φ_i, be associated with the time-invariant system matrix Ω of (4), then the exact score for this element is given by

$$\frac{\partial l}{\partial \varphi_i} = \frac{1}{2}\mathrm{trace}\left(S\frac{\partial \Omega}{\partial \varphi_i}\right), \quad \text{with} \quad S = \sum_{t=1}^{n} r_t^* r_t^{*\prime} - N_t^*, \tag{27}$$

where r_t^* and N_t^* are defined in (and calculated by) the smoothing algorithm of §4.4. SsfPack only implements the analytical scores for parameters in Ω, resulting in more efficient computation than when numerical derivatives are used.

Usually it is possible to solve explicitly for one scale factor, by concentrating it out of the likelihood; see, e.g. Harvey (1989, pages 126-127). Let σ be the scale factor, and

use superscript ^{c} to denote the scaled version of the measurement equation (3):

$$y_t = \theta_t + G_t^c \varepsilon_t^c, \qquad \varepsilon_t^c \sim \mathrm{N}\left(0, \sigma^2 I\right), \qquad \sigma^2 > 0,$$

with unknown variance σ^2. The state space form (1) and (3) applies but with $G_t = \sigma G_t^c$ and $H_t = \sigma H_t^c$. This formulation implies that one non-zero element of σG_t^c or σH_t^c is kept fixed, usually at unity. This reduces the dimension of φ by one. Equation (25) can be solved explicitly for σ^2, giving:

$$\tilde{\sigma}^2 = \frac{1}{Nn - d} \sum_{t=1}^{n} v_t' \left(F_t^c\right)^{-1} v_t, \tag{28}$$

The concentrated or profile log-likelihood is given by

$$l_c = -\frac{nN}{2} \log(2\pi) - \frac{nN - d}{2} \left(\log \tilde{\sigma}^2 + 1\right) - \frac{1}{2} \sum_{t=1}^{n} \log |F_t^c| \,. \tag{29}$$

Exact scores for the concentrated log-likelihood are not available.

SsfPack implementation. The following *SsfPack* functions are provided for log-likelihood and score evaluation:

```
SsfLik(&dLogLik, &dVar, mYt, {Ssf});
SsfLikConc(&dLogLikConc, &dVar, mYt, {Ssf});
SsfLikSco(&dLogLik, &dVar, &mSco, mYt, {Ssf});
```

All functions return a `1` to indicate that they were successful, and `0` otherwise. The input arguments are the data matrix ($N \times n$; `mYt`), and the state space model, written here as `{Ssf}` (see §4.1).

Additional values are returned in the arguments prefixed by `&`. These are:

`SsfLik:` (25) in `&dLogLik` and (26) in `&dVar`;
`SsfLikConc:` (29) in `&dLogLikConc` and (28) in `&dVar`;
`SsfLikSco:` (25) in `&dLogLik`, (26) in `&dVar` and S from (27) in `&mSco`.

All values returned in arguments are scalars, except for the `mSco`, which is an $(m+N) \times (m+N)$ matrix.

Application: Maximum likelihood estimation of ARMA models. The example implemented in Listing 9, is the well-known airline model, see Box and Jenkins (1976):

$$\begin{aligned} \Delta\Delta_{12} y_t &= (1 + \theta_1 L)\left(1 + \theta_{12} L^{12}\right) \epsilon_t \\ &= \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_{12} \epsilon_{t-12} + \theta_1 \theta_{12} \epsilon_{t-13}, \qquad \epsilon_t \sim \mathrm{N}(0, \sigma_\epsilon^2), \end{aligned}$$

where the y_t are in logs.

The likelihood (25) can be maximized numerically using the `MaxBFGS` routine from *Ox*; see Doornik (1998, page 243). There are three parameters to estimate:

$$\varphi = (\theta_1, \theta_{12}, \log(\sigma_\epsilon))' \,.$$

`MaxBFGS` works with φ, so we need to map this into the state space formulation. In Listing 9 this is done in two steps:

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

static decl s_mY, s_cT;                 // data (1 x T) and T
static decl s_vAR, s_vMA;               // AR and MA parameters
static decl s_dSigma, s_dVar;           // residual std.err. and scale factor

SetAirlineParameters(const vP)
{
    // map to airline model: y[t] = (1+aL)(1+bL^12)e[t]
    s_vAR = <>;
    s_vMA = vP[0] ~ zeros(1,10) ~ vP[1] ~ vP[0] * vP[1];
    s_dSigma = exp(vP[2]);
}
ArmaLogLik(const vY, const pdLik, const pdVar)
{
    decl mphi, momega, msigma, ret_val;
                                        // get state space model and loglik
    GetSsfArma(s_vAR, s_vMA, s_dSigma, &mphi, &momega, &msigma);
    ret_val = SsfLik(pdLik, pdVar, vY, mphi, momega, msigma);
    return ret_val;                     // 1 indicates success, 0 failure
}
Likelihood(const vP, const pdLik, const pvSco, const pmHes)
{                                       // arguments dictated by MaxBFGS()
    decl ret_val;

    SetAirlineParameters(vP);           // map vP to airline model
    ret_val = ArmaLogLik(s_mY, pdLik, &s_dVar); // evaluate at vP
    pdLik[0] /= s_cT;                   // log-likelihood scaled by sample size
    return ret_val;                     // 1 indicates success, 0 failure
}
ArmaStderr(const vP)
{
    decl covar, invcov, var = s_vAR, vma = s_vMA, dsig = s_dSigma, result;

    result = Num2Derivative(Likelihood, vP, &covar);
    s_vAR = var, s_vMA = vma, s_dSigma = dsig;  // reset after Num2Der
    if (!result)
    {   print("Covar() failed in numerical second derivatives\n");
        return zeros(vP);
    }
    invcov = invertgen(-covar, 30);
    return sqrt(diagonal(invcov) / s_cT)';
}
```

Listing 9. ssfair.ox (first part)

1. `SetAirlineParameters` splits φ in AR parameters, MA parameters and σ_ϵ;
2. `ArmaLogLik` creates the state space for this model.

`ArmaLogLik` also computes the log-likelihood. `MaxBFGS` accepts a function as its first argument, but requires it to be in a specific format, which is called `Likelihood` here.

```
main()
{
    decl vp, ir, dlik, dvar, my, mdy;

    my = log(loadmat("Airline.dat"));   // log(airline)
    mdy = diff0(my, 1)[1:][];           // Dlog(airline)
    s_mY = diff0(mdy, 12)[12:][]';      // D12D(log(airline)) transposed!!
    s_cT = columns(s_mY);               // no of observations

    vp = <0.5;0.5;0>;                   // starting values
    // scale initial parameter estimates for better starting values
    SetAirlineParameters(vp);           // map parameters to airline model
    ArmaLogLik(s_mY, &dlik, &dvar);     // evaluate
    vp[sizerc(vp)-1] = 0.5 * log(dvar); // update starting log(sigma)

    MaxControl(-1, 5, 1);               // get some output from MaxBFGS
    MaxControlEps(1e-7, 1e-5);          // tighter convergence criteria
    ir = MaxBFGS(Likelihood, &vp, &dlik, 0, TRUE);

    println("\n", MaxConvergenceMsg(ir),
          " using numerical derivatives",
          "\nLog-likelihood = ", "%.8g", dlik * s_cT,
          "; variance = ", sqr(s_dSigma),
          "; n = ", s_cT, "; dVar = ", s_dVar);
    print("parameters with standard errors:",
        "%cf", {"%12.5g", "  (%7.5f)"}, vp ~ ArmaStderr(vp));
}
```

```
Ox version 2.00f (Windows) (C) J.A. Doornik, 1994-98
it0    f=      1.079789 df=     1.087 e1=    0.5434 e2=    0.2174 step=1
it5    f=      1.851184 df=    0.2023 e1=    0.1631 e2=   0.01153 step=1
it10   f=      1.867910 df=  0.001848 e1=  0.006105 e2= 0.0002357 step=1
it14   f=      1.867912 df=7.130e-008 e1=4.256e-008 e2=1.977e-008 step=1

Strong convergence using numerical derivatives
Log-likelihood = 244.69649; variance = 0.00134809; n = 131; dVar = 1.00001
parameters with standard errors:
    -0.40182  (0.08964)
    -0.55694  (0.07311)
     -3.3045  (0.06201)
```

Listing 9. ssfair.ox (continued) with output

We prefer to maximize l/n rather then l, to avoid dependency on n in the convergence criteria.

A starting value for $\log(\sigma_\epsilon)$ is chosen as follows:

1. first evaluate the likelihood with $\sigma_\epsilon = 1$;
2. next, use `dVar` as returned by `SsfLik` for the initial value of σ_ϵ^2.

SsfPack only provides analytical derivatives for parameters in Ω, so only for the MA part of ARMA models.

Compact output during iteration is given for every fifth iteration. It consists of the function value (`f`), the largest (in absolute value) score (`df`), the largest of both convergence criteria (`e1`) and (`e2`), and the step length. Upon convergence, the coefficient standard errors are computed using numerical second derivatives.

To illustrate the use of the concentrated log-likelihood, we adjust the program slightly (see Listing 10). The listing only provides `ArmaLogLikc`, where `GetSsfArma` is now called with standard deviation set to one, and σ_ϵ is obtained from `SsfLikConc`. In this setup, there are only the two MA parameters to estimate, and the maximization process is more efficient. Note that the attained likelihoods are the same (this is not necessarily the case when using the *SsfPack* functions: they are identical when σ from (26) equals one). Also note that the reported variance in Listing 10 equals the last parameter value in Listing 9: $\log(0.0367165) = -3.3045$.

```
ArmaLogLikc(const vY, const pdLik, const pdVar)
{
    decl mphi, momega, msigma, ret_val;
                                        // use 1 in GetSsfArma
    GetSsfArma(s_vAR, s_vMA, 1, &mphi, &momega, &msigma);
    ret_val = SsfLikConc(pdLik, pdVar, vY, mphi, momega, msigma);
    s_dSigma = sqrt(pdVar[0]);          // get sigma from SsfLikConc
    return ret_val;                     // 1 indicates success, 0 failure
}
```

```
it0    f=      1.079789 df=     1.087 e1=    0.5434 e2=    0.2174 step=1
it5    f=      1.867912 df= 0.0001485 e1=8.269e-005 e2=1.818e-005 step=1
it7    f=      1.867912 df=2.702e-008 e1=1.086e-008 e2=6.982e-009 step=1
Strong convergence using numerical derivatives
Log-likelihood = 244.69649; variance = 0.0367165 (= dVar); n=131
parameters with standard errors:
    -0.40182  (0.08964)
    -0.55694  (0.07311)
```

Listing 10. Part of ssfairc.ox with output

5.2. Prediction and forecasting

Prediction. The Kalman filter of §4.3 produces the one-step ahead prediction of the state vector, that is, the conditional mean $\mathrm{E}(\alpha_t|Y_{t-1})$, denoted by a_t, together with the variance matrix P_t, for $t = 1, \ldots, n$. The *SsfPack* function `SsfMomentEst` can be used to obtain these quantities.

SsfPack implementation. The call of the state prediction function is given by

```
mState = SsfMomentEst(ST_PRED, &mPred, mYt, {Ssf});
```

where the returned matrix `mState` contains a_{n+1} and P_{n+1}. The constant `ST_PRED` is pre-defined and must be given when state prediction is required. The data matrix `mYt` and the sequence {`Ssf`} are as usual. This function returns in `mPred` a matrix containing

a_t and the diagonal elements of P_t, for $t = 1, \ldots, n$. If these are not required, use 0 as the second argument. The structure of the output matrices is given by

$$\texttt{mState} = \begin{bmatrix} P_{n+1} \\ a'_{n+1} \end{bmatrix}, \qquad \texttt{mPred} = \begin{bmatrix} a_1 & \ldots & a_n \\ \hat{y}_1 & \ldots & \hat{y}_n \\ \mathrm{diag}\,(P_1) & \ldots & \mathrm{diag}\,(P_n) \\ \mathrm{diag}\,(F_1) & \ldots & \mathrm{diag}\,(F_n) \end{bmatrix},$$

where $\hat{y}_t = \mathrm{E}\,(y_t|Y_{t-1})$ and $F_t = \mathrm{var}\,(y_t|Y_{t-1}) = \mathrm{var}\,(v_t)$ with $v_t = y_t - \hat{y}_t$. The output is directly obtained from the Kalman filter.

The dimensions of the two matrices returned by `SsfMomentEst` are:

$$\texttt{mState}: \begin{bmatrix} m \times m \\ 1 \times m \end{bmatrix}, \qquad \texttt{mPred}, \texttt{mStSmo}, \texttt{mDisturb}: \begin{bmatrix} m \times 1 \\ N \times 1 \\ m \times 1 \\ N \times 1 \end{bmatrix},$$

where `mStSmo` and `mDisturb` are defined in §5.3.

Forecasting. Out-of-sample predictions, together with their mean square errors, can be generated by the Kalman filter by extending the data set $y_1, \ldots, y_n$ with a set of missing values. When y_τ is missing, the Kalman filter step at time $t = \tau$ reduces to

$$a_{\tau+1} = T_\tau a_\tau, \qquad P_{\tau+1} = T_\tau P_\tau T'_\tau + H_\tau H'_\tau,$$

which are the state space forecasting equations; see Harvey (1989, page 147) and West and Harrison (1997, page 39). A sequence of missing values at the end of the sample will therefore produce a set of multi-step forecasts.

Application: Forecasting from ARMA models. In Listing 11 `SsfMomentEst` is used to forecast from the airline model estimated in Listing 9. Listing 11 reproduces `ArmaForc`, which calls `SsfMomentEst` twice:

1. first to obtain the state at $t = n + 1$;
2. in the next call, Σ is replaced by `mState`$= (P'_{n+1}, a_{n+1})'$, and a $1 \times h$ matrix of missing values is used instead of $(y_1, \ldots, y_n)$.

`ArmaForc` returns the original data, with forecasts appended; the second column contains the forecast standard errors. The graph presents the forecasts in levels (but still in logs).

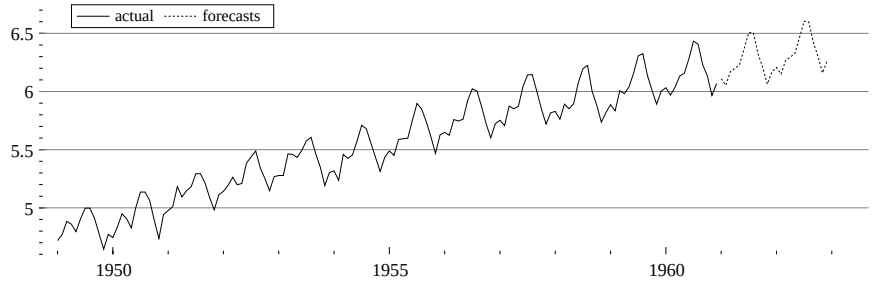
5.3. Smoothing

State smoothing. The evaluation of $\hat{\alpha}_t = \mathrm{E}(\alpha_t|Y_n)$ and variance matrix $V_t = \mathrm{var}(\alpha_t|Y_n)$ is referred to as moment state smoothing. The usual state smoothing algorithm can be found in Anderson and Moore (1979, page 165) and Harvey (1989, page 149). Computationally more efficient algorithms are developed by de Jong (1988a) and Kohn and Ansley (1989). Koopman (1998) shows how the different algorithms are related. The state smoother in *SsfPack* is given by

$$\hat{\alpha}_t = a_t + P_t r_{t-1}, \qquad V_t = P_t - P_t N_{t-1} P_t, \qquad t = n, \ldots, 1, \tag{30}$$

```
ArmaForc(const vY, const cForc)
{
    decl mphi, momega, msigma, mstate, mfor, m;

    GetSsfArma(s_vAR, s_vMA, s_dSigma, &mphi, &momega, &msigma);
    m = columns(mphi);
    mstate = SsfMomentEst(ST_PRED, 0, vY, mphi, momega, msigma);
    SsfMomentEst(ST_PRED, &mfor, constant(M_NAN,1,cForc), mphi, momega, mstate);
    return (vY ~ mfor[m][]) | (zeros(vY) ~ sqrt(mfor[2 * m + 1][]));
}
// ... in main:
    decl mforc = ArmaForc(s_mY, 24)';           // forecasts of D1D12
    mforc = Cum1Cum12(mforc[][0], mdy, my)';    // translate to levels
```



Forecast generated by ssfairf.ox

Listing 11. Part of ssfairf.ox with output

where r_{t-1} and N_{t-1} are evaluated by (20). Note that a_t and P_t are evaluated in a forward fashion and the state smoother is a backward operation. Evaluation of these quantities requires a substantial amount of storage space for the a_t and P_t. This is in addition to the storage space required in order to evaluate r_{t-1} and N_{t-1}; see §4.4. When only the smoothed state $\hat{\alpha}_t$ is required, more efficient methods of calculation are provided; see §4.6.

SsfPack implementation. The call for the state smoothing function is

```
mSmo = SsfMomentEst(ST_SMO, &mStSmo, mYt, {Ssf});
```

The constant `ST_SMO` is predefined and must be given when state smoothing is required. The structure of the returned matrices is given by:

$$\texttt{mSmo} = \begin{bmatrix} N_0 \\ r_0' \end{bmatrix}, \qquad \texttt{mStSmo} = \begin{bmatrix} \hat{\alpha}_1 & \dots & \hat{\alpha}_n \\ \hat{\theta}_1 & \dots & \hat{\theta}_n \\ \operatorname{diag}(V_1) & \dots & \operatorname{diag}(V_n) \\ \operatorname{diag}(S_1) & \dots & \operatorname{diag}(S_n) \end{bmatrix},$$

where $\hat{\theta}_t = c_t + Z_t\hat{\alpha}_t$ is the smoothed estimate of the signal $\theta_t = c_t + Z_t\alpha_t$ with variance matrix $S_t = Z_t V_t Z_t'$.

Disturbance smoothing. The smoothed estimate of the disturbance vector of the state space form (4), $u_t = (H_t', G_t')'\varepsilon_t$, denoted by $\hat{u}_t$ $(t = 1, \ldots, n)$, is discussed in §4.4. Disturbance smoothing can be represented by the simple algorithm:

$$\begin{array}{rclcrcll} \hat{u}_t & = & \Omega_t r_t^*, & & \mathrm{var}\,(\hat{u}_t) & = & \Omega_t N_t^* \Omega_t, & \\ r_{t-1} & = & \Phi_t r_t^*, & & N_{t-1} & = & \Phi_t N_t^*, \Phi_t', & t = n, \ldots, 1, \end{array}$$

where r_t^* and N_t^* are defined in step (ii) of the algorithm in §4.4; see also Koopman (1993).

SsfPack implementation. The call of the disturbance smoothing function is given by

```
mSmo = SsfMomentEst(DS_SMO, &mDisturb, mYt, {Ssf});
```

The constant `DS_SMO` is pre-defined and must be given when disturbance smoothing is required. The structure of the returned matrices is given by:

$$\texttt{mSmo} = \begin{bmatrix} N_0 \\ r_0' \end{bmatrix}, \qquad \texttt{mDisturb} = \begin{bmatrix} H_1\hat{\varepsilon}_1 & \ldots & H_n\hat{\varepsilon}_n \\ G_1\hat{\varepsilon}_1 & \ldots & G_n\hat{\varepsilon}_n \\ \mathrm{diag}\,\{\mathrm{var}\,(H_1\hat{\varepsilon}_1)\} & \ldots & \mathrm{diag}\,\{\mathrm{var}\,(H_n\hat{\varepsilon}_n)\} \\ \mathrm{diag}\,\{\mathrm{var}\,(G_1\hat{\varepsilon}_1)\} & \ldots & \mathrm{diag}\,\{\mathrm{var}\,(G_n\hat{\varepsilon}_n)\} \end{bmatrix}.$$

Application: Detecting outliers and structural breaks. The example, partially reproduced in Listing 12, estimates a local level model for the Nile data, and performs outlier and structural break detection.

`MaxBFGS` is used again to estimate the local level model:

$$\begin{aligned} y_t &= \mu_t + \xi_t, \qquad \xi_t \sim \mathrm{NID}\left(0, \sigma_\xi^2\right), \\ \mu_{t+1} &= \mu_t + \eta_t, \qquad \eta_t \sim \mathrm{NID}\left(0, \sigma_\eta^2\right), \qquad t = 1, \ldots, n, \end{aligned} \tag{31}$$

with $\mu_1 \sim \mathrm{N}(0, \kappa)$ and κ large. This model has two unknown variances which are re-parameterized as

$$\sigma_\eta^2 = \exp\left(2\varphi_0\right), \qquad \sigma_\xi^2 = \exp\left(2\varphi_1\right),$$

so that the likelihood criterion can be maximized without constraints with respect to $\varphi = (\varphi_0, \varphi_1)'$. The score (27) is calculated by

$$\left.\frac{\partial l}{\partial \varphi_0}\right|_{\varphi = \varphi^*} = \sigma_\eta^{2*} S_{00}, \qquad \left.\frac{\partial l}{\partial \varphi_1}\right|_{\varphi = \varphi^*} = \sigma_\xi^{2*} S_{11},$$

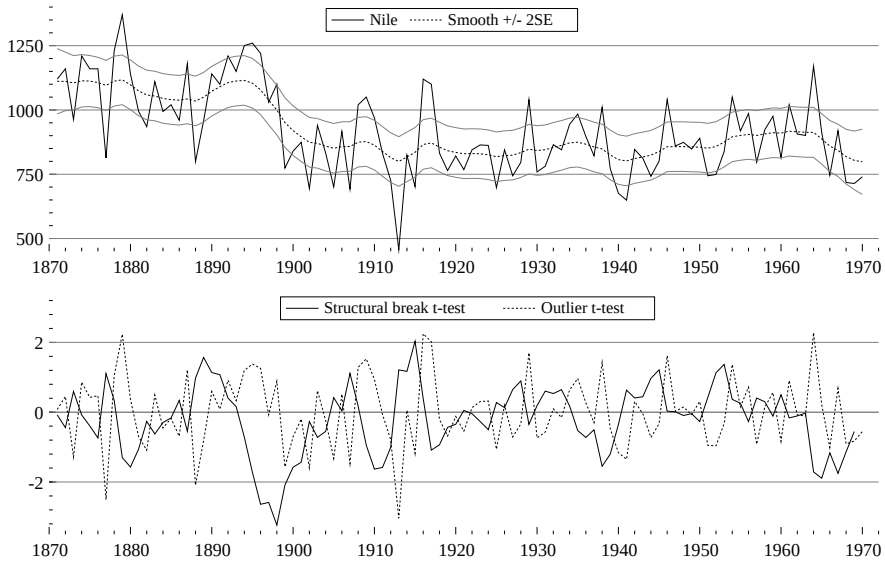
where S_{ij} is the (i, j)-th element of matrix S in (27) for $\varphi = \varphi^*$. The log-likelihood and S are obtained from `SsfLikSco`. The first graph in Listing 12 shows the estimated local level with a band of $\pm$ two standard errors.

General procedures for testing for outliers and structural breaks based on models in state space form are discussed by Harvey, Koopman, and Penzer (1998). Such irregularities in data can be modelled in terms of impulse interventions applied to the equations of the state space form. For example, an outlier can be captured within the measurement equation by a dummy explanatory variable, known as an impulse intervention variable, which takes the value one at the time of the outlier and zero elsewhere. The estimated

```
    // smoothed state vector
    mstate = SsfMomentEst(ST_SMO, &mks, s_mYt, s_mPhi, s_mOmega);
    // smoothed disturbance vector
    SsfMomentEst(DS_SMO, &md, s_mYt, s_mPhi, s_mOmega, s_mSigma);
    // auxiliary residuals
    ms = md[0:1][] ./ sqrt(md[2:3][]);

    DrawTMatrix(0, s_mYt, {"Nile"}, 1871, 1, 1);
    DrawTMatrix(0, mks[1][], {"Smooth +/- 2SE"}, 1871, 1, 1, 0, 3);
    DrawZ(sqrt(mks[3][]), "", ZMODE_BAND, 2.0, 14);
    DrawTMatrix(1, ms,
        {"Structural break t-test", "Outlier t-test"}, 1871, 1, 1);
    ShowDrawWindow();
```

```
Strong convergence using analytical derivatives
Log-likelihood = -633.465; dVar = 1.00002; parameters:
       3.6462       4.8112
Omega
       1469.2      0.00000
      0.00000       15098.
```



Estimated level, outlier and break tests generated by ssfnile.ox

Listing 12. Part of ssfnile.ox with output

regression coefficient of this variable is an indication whether an outlying observation is present. In the case of unobserved component time series models, this approach reduces to a procedure based on the so-called auxiliary residuals. The standardized residuals associated with the measurement and system equations are computed via a single filter and

smoothing step; see §4.4 and §5.3. These auxiliary residuals are introduced and studied in detail by Harvey and Koopman (1992); they show that these residuals are an effective tool for detecting outliers and breaks in time series and for distinguishing between them. It was shown by de Jong and Penzer (1998) that auxiliary residuals are equivalent to t-statistics for the impulse intervention variables. The second graph in Listing 12 shows the auxiliary residuals for ξ_t and η_t.

Application: Regression analysis. When the standard regression model

$$y_t = X_t\beta + \xi_t \text{ with } \xi_t \sim \text{NID}\left(0, \sigma_\xi^2\right)$$

with k vector of explanatory variables $X_t = (x_{1,t}, \ldots, x_{k,t})$ is placed in the state space form, the Kalman filter reduces to what is known as 'recursive least squares' algorithm. The state prediction a_t is the least squares estimate $(\sum_{j=1}^{t-1} X_j X_j')^{-1}(\sum_{j=1}^{t-1} X_j y_j)$ and matrix P_t is the matrix $(\sum_{j=1}^{t-1} X_j X_j')^{-1}$, see Harvey (1993, §4.5). Therefore, the *SsfPack* function `SsfMomentEst` can be used to obtain these quantities and to obtain the final OLS estimates, that is a_{n+1} and P_{n+1}.

Additional statistical output is obtained from smoothing. Following the arguments of de Jong and Penzer (1998), the output of the basic smoothing recursions can be used to construct t-tests for structural changes in regression models. The null hypothesis $\beta_i = \beta_i^*$ with respect to the ith explanatory variable in

$$\begin{aligned} y_t &= \ldots + x_{i,t}\beta_i + \ldots + \xi_t, \qquad \text{for } t = 1, \ldots, \tau, \\ y_t &= \ldots + x_{i,t}\beta_i^* + \ldots + \xi_t, \qquad \text{for } t = \tau + 1, \ldots, n, \end{aligned}$$

against the alternative $\beta_i \neq \beta_i^*$ can be tested via the t-test

$$r_{i,\tau}/\sqrt{N_{ii,\tau}}, \qquad \tau = 1, \ldots, n-1,$$

where $r_t = (r_{1,t}, \ldots r_{p,t})'$ and N_t, with the element (i,i) denoted as $N_{ii,t}$, are evaluated using the basic smoothing recursions (20). The $(n-1)k$ t-tests can be computed from a single run of the basic smoother. The test has a t distribution with $n-k$ degrees of freedom. A relatively large t-test provides evidence against the null hypothesis.

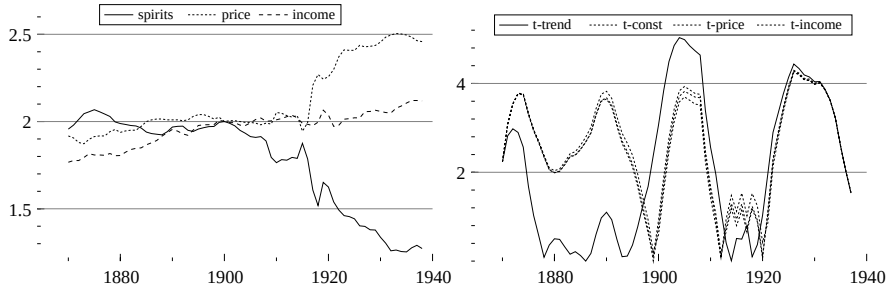


Figure 3. Spirits data and stability tests generated by ssfspirits.ox

The regression application is based on per capita consumption of spirits in the UK from 1870 to 1938. This data set was collected and first analysed by Prest (1949); also

see Kohn and Ansley (1989) and Koopman (1992, pages 127–129) among others. As can be seen in Figure 3, there is strong evidence of structural breaks in this model. This can be no surprise, as the model is clearly misspecified (lack of dynamics, special periods such as World War I, and so on).

```
    GetSsfReg(mx, &mphi, &momega, &msigma, &mj_phi);// regression in state space
    // calculate likelihood and error variance
    SsfLikConc(&dlik, &dvar, myt, mphi, momega, msigma, <>, mj_phi, <>, <>, mx);
    // regression
    momega *= dvar;
    mstate = SsfMomentEst(ST_PRED, <>, myt, mphi, momega, msigma, <>,
        mj_phi, <>, <>, mx);
    vse  = sqrt(diagonal(mstate[0:ck-1][]));
    mols = mstate[ck][] | vse | fabs(mstate[ck][] ./ vse);
    // stability tests
    mkf  = KalmanFil(myt,  mphi, momega, msigma, <>, mj_phi, <>, <>, mx);
    mks  = KalmanSmo(mkf, mphi, momega, msigma, <>, mj_phi, <>, <>, mx);
    mstab= fabs(mks[:ck-1][1:ct-1] ./ sqrt(mks[ck+1:(2*ck)][1:ct-1]));
```

```
                      coef         s.e.      t-value
const                1.8277      0.36964       4.9445
trend           -0.0091153    0.0011578       7.8731
price             -0.85994     0.059011       14.572
income              1.0618      0.16930       6.2718
Modified profile log-likelihood 104.968 log-likelihood 123.353
variance 0.00174044 RSS 0.113128
```

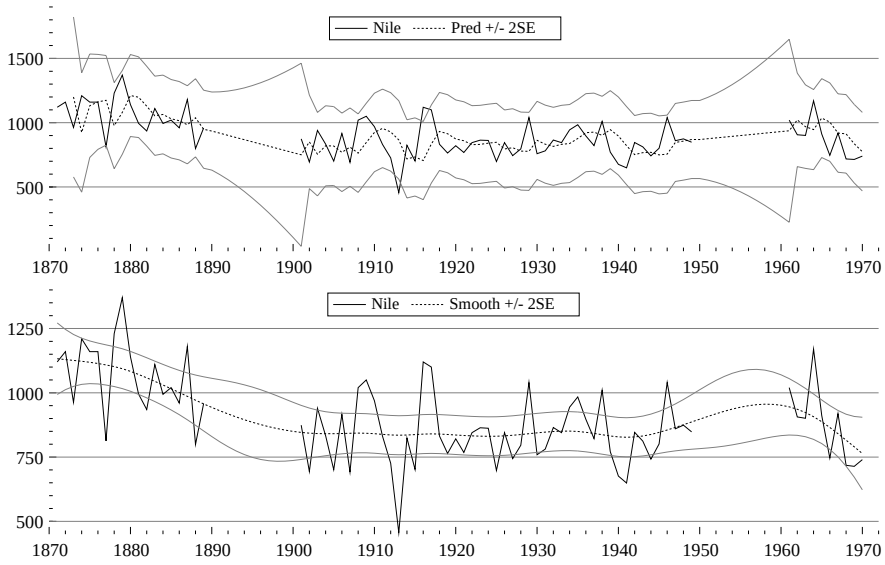
Listing 13. Part of ssfspirits.ox with output

Application: Spline with missing values. The nonparametric spline method can be regarded as an interpolation technique. Consider a set of observations which are spaced at equal intervals but some observations are missing. To 'fill in the gaps' the spline model of §3.4 can be considered. Applying filtering and smoothing to this model, we obtain the estimated signal. In this way, a graphical representation of the nonparametric spline can be produced.

The spline application is based on the Nile data set, where we replaced observations from 1890–1900 and 1950–1960 by missing values. We fixed q at 0.004 in the spline model, but, of course, in the state space setup it would be easy to estimate q. The first graph in Listing 14 presents $\hat{y}_t$, the filtered estimate of the signal; the second shows $\hat{\theta}_t$, the smoothed estimate. The graphs show a distinct difference between filtering and smoothing, corresponding to extrapolation and interpolation respectively.

```
myt = loadmat("Nile.dat")';
myt[][1890-1871:1900-1871] = M_NAN;          // set 1890..1900 to missing
myt[][1950-1871:1960-1871] = M_NAN;          // set 1850..1960 to missing

GetSsfSpline(0.004, <>, &mphi, &momega, &msigma);   // SSF for spline
SsfLik(&dlik, &dvar, myt, mphi, momega);     // need dVar
cm = columns(mphi);                          // dimension of state
momega *= dvar;                              // set correct scale of Omega
SsfMomentEst(ST_PRED, &mpred, myt, mphi, momega);
SsfMomentEst(ST_SMO, &mstsmo, myt, mphi, momega);
```



Output generated by ssfnilesp.ox

Listing 14. Part of ssfnilesp.ox with output

6. FURTHER APPLICATIONS

6.1. Seasonal components

The unobserved components model was discussed in section 3.2. The basic model consisted of trend, seasonal and irregular components. For the seasonal component we formulated a simple model which was based on seasonal dummy variables. There are however other seasonal models; for example, they can be based on a set of trigonometric terms which are made time-varying in a similar way as for the cycle of section 3.2 but with $\rho = 1$. This so-called trigonometric seasonal model for γ_t is given by

$$\gamma_t = \sum_{j=1}^{[s/2]} \gamma_{j,t}^{+}, \quad \text{where} \quad \begin{pmatrix} \gamma_{j,t+1}^{+} \\ \gamma_{j,t+1}^{*} \end{pmatrix} = \begin{pmatrix} \cos\lambda_j & \sin\lambda_j \\ -\sin\lambda_j & \cos\lambda_j \end{pmatrix} \begin{pmatrix} \gamma_{j,t}^{+} \\ \gamma_{j,t}^{*} \end{pmatrix} + \begin{pmatrix} \omega_{j,t}^{+} \\ \omega_{j,t}^{*} \end{pmatrix}, \tag{32}$$

with $\lambda_j = 2\pi j/s$ as the j-th seasonal frequency and

$$\begin{pmatrix} \omega_{j,t}^{+} \\ \omega_{j,t}^{*} \end{pmatrix} \sim \text{NID}\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma_\omega^2 I_2 \right\}, \qquad j = 1, \ldots, [s/2].$$

Note that for s even $[s/2] = s/2$, while for s odd, $[s/2] = (s-1)/2$. For s even, the process $\gamma_{j,t}^{*}$, with $j = s/2$, can be dropped. The state space representation is straightforward and the initial conditions are $\gamma_{j,1}^{+} \sim \text{N}(0, \kappa)$ and $\gamma_{j,1}^{*} \sim \text{N}(0, \kappa)$, for $j = 1, \ldots, [s/2]$. We have assumed that the variance σ_ω^2 is the same for all trigonometric terms. However, we can impose different variances for the terms associated with different frequencies; in the quarterly case we can estimate two different σ_ω^2 's rather than just one.

The dummy and trigonometric specifications for γ_t have different dynamic properties; see Harvey (1989, page 56). For example, the trigonometric seasonal process evolves more smoothly; it can be shown that the sum of the seasonals over the past 'year' follows an MA($s-2$) rather than white noise. The same property holds for the Harrison and Stevens seasonal representation for which all s individual seasonal effects collected in the vector $\gamma_t^{\times}$ follow a random walk, that is

$$\gamma_{t+1}^{\times} = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_s \end{pmatrix}_{t+1} = \gamma_t^{\times} + \omega_t, \quad \text{where} \quad \omega_t = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_s \end{pmatrix}_{t} \sim \text{NID}\left\{ 0, \sigma_\omega^2 \left(\frac{sI_s - i_s i_s'}{s-1} \right) \right\},$$

and i_s is a $s \times 1$ vector of ones; see Harrison and Stevens (1976). The specific covariance structure between the s disturbance terms enforces the seasonal effects to sum to zero over the previous 'year'. Also, the covariances between the s seasonal disturbances are equal. The state space form is set up such that it selects the appropriate seasonal effect from $\gamma_t^{\times}$; this implies a time-varying state space framework. However, the state space representation can be modified to a time-invariant form as follows. Let $\gamma_t = (1, 0')\gamma_t^{\times}$, then

$$\begin{aligned} \gamma_{t+1}^{\times} = \begin{pmatrix} 0 & I_{s-1} \\ 1 & 0 \end{pmatrix} \gamma_t^{\times} + \omega_t, \quad &\text{where} \quad \omega_t \sim \text{NID}\left(0, \sigma_\omega^2 \tfrac{sI_s - i_s i_s'}{s-1}\right), \\ &\text{and} \quad \gamma_1^{\times} \sim \text{N}\left(0, \kappa \tfrac{sI_s - i_s i_s'}{s-1}\right). \end{aligned} \tag{33}$$

The implications of the different seasonal specifications are discussed in more detail by Harvey, Koopman, and Penzer (1998).

An interesting extension of the Harrison and Stevens seasonal is given by $\gamma_t = (1, 0')\gamma_t^{\times}$ with

$$\gamma_{t+1}^{\times} = \rho\gamma_t^{\times} + \omega_t, \quad \text{where} \quad \omega_t \sim \text{NID}\left\{\bar{\gamma}, \sigma_\omega^2\left(1-\rho^2\right)\frac{sI_s - i_s i_s'}{s-1}\right\},$$
$$\text{and} \qquad \gamma_1^{\times} \sim \text{N}\left\{\bar{\gamma}, \sigma_\omega^2\frac{sI_s - i_s i_s'}{s-1}\right\}.$$

This specification provides a stationary seasonal model around some average seasonal pattern given by the unknown fixed $s \times 1$ vector of means $\bar{\gamma}$. It is possible to have both stationary and nonstationary seasonal components in a single unobserved components model, but in that case identification requirements stipulate that $\bar{\gamma}$ is set to zero.

In *SsfPack* the dummy seasonal specification (13) was set as the pre-defined constant `CMP_SEAS_DUMMY` for the function `GetSsfStsm`; see §3.2. The constant `CMP_SEAS_TRIG` must be used if a trigonometric specification (32) is required; use `CMP_SEAS_HS` for the Harrison and Stevens specification (33).

Application: Seasonal adjustment with trigonometric seasonals. Seasonal adjustment is a relatively easy task when time series are modelled as an unobserved components time series model in which a seasonal component is included; see §3.2. The estimated seasonal component is subtracted from the original time series in order to get the seasonally adjusted series. In the same way the original time series is detrended by subtracting the estimated trend component. In the example below we model the monthly airline data with trend, seasonal and irregular components. The trigonometric seasonal specification is used but without the restriction that the variances of the six time-varying trigonometric terms are the same. This model for the airline data is estimated in three steps. Firstly, we estimate the variances with the restriction of one variance for all trigonometric terms. It turned out that the slope variance was estimated to be zero resulting in a fixed slope. Secondly, the model is estimated without the restriction of equal variances for the six trigonometric terms, but with the restriction of a zero variance for the slope. Two variances associated with the trigonometric terms were estimated to be zero. Finally, the model is estimated with three zero restrictions on the variances imposed. The results of this model are presented in Listing 15. The values of the estimated variances are given, together with a set of eight graphs. The last four plot the four trigonometric terms which have been estimated. Together, these make up the seasonal component of the second graph.

6.2. Combining models

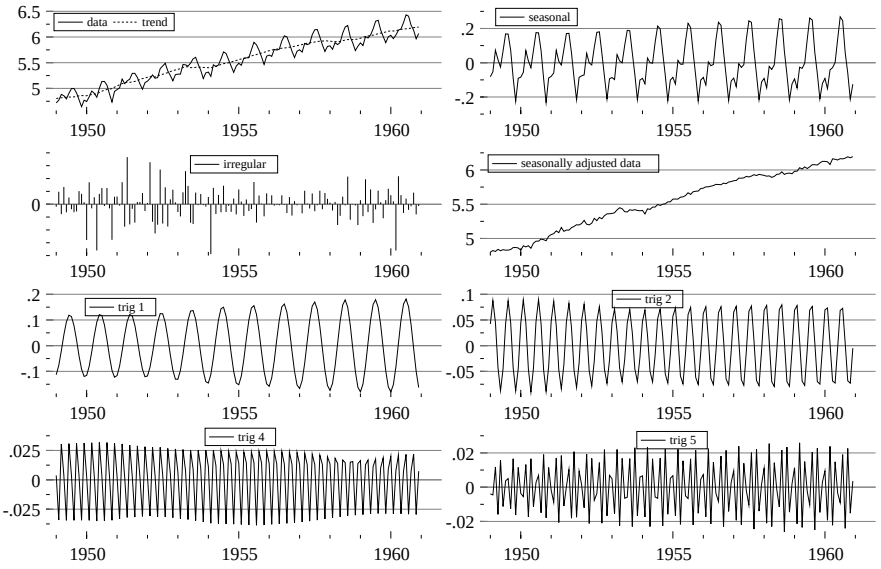
The system matrices of two different models can be combined into the corresponding system matrices for the joint model. Consider model A and B, where

$$\Phi^A = \begin{bmatrix} T^A \\ Z^A \end{bmatrix}, \qquad \Phi^B = \begin{bmatrix} T^B \\ Z^B \end{bmatrix}.$$

```
SetStsmModel(const vP)
{
    // map to sts model with level, slope and trig seasonal
    s_mStsm = <CMP_LEVEL,     1,  0, 0;
               CMP_SLOPE,     0,  0, 0;
               CMP_SEAS_TRIG, 1, 12, 0; // 12 for monthly data
               CMP_IRREG,     1,  0, 0>;

    decl vr = exp(2.0 * vP);            // from log(s.d.) to variance
    s_vVarCmp =                         // s_vVarCmp is diagonal(Omega)
     // level  slope  ---------  monthly trigonometric seasonal  --------  irreg
        vr[0] |  0  | ((vr[1] | vr[2] | 0 | vr[3] | vr[4]) ** <1;1>) | 0 | vr[5];
}
LogLikStsm(const vY, const pdLik, const pdVar)
{
    decl mphi, momega, msigma, ret_val;
    GetSsfStsm(s_mStsm, &mphi, &momega, &msigma); // get state space model
    momega = diag(s_vVarCmp);           // create Omega from s_vVarCmp
    ret_val = SsfLik(pdLik, pdVar, vY, mphi, momega, msigma);
    return ret_val;                     // 1 indicates success, 0 failure
}
```

```
Log-likelihood = 223.46337; n = 144;
variance parameters (* 10,000):
 2.38    0 0.11 0.11 0.05 0.05    0    0 0.02 0.02 0.01 0.01    0 3.27
```



Output generated by ssfairstsm.ox

Listing 15. Part of ssfairstsm.ox with output

Z^A and Z^B have the same number of rows. The combined system matrix Φ is:

$$\Phi = \begin{bmatrix} T^A & 0 \\ 0 & T^B \\ Z^A & Z^B \end{bmatrix}.$$

The matrices Σ^A, Σ^B, and δ^A, δ^B can be combined in the same way. This procedure also applies when combining the index matrices J_Φ^A and J_Φ^B into J_Φ. However, where Φ has two blocks of zeros, J_Φ must have two blocks with -1s.

To combine the variance system matrices Ω^A and Ω^B, where

$$\Omega^A = \begin{bmatrix} (HH')^A & (HG')^A \\ (GH')^A & (GG')^A \end{bmatrix}, \qquad \Omega^B = \begin{bmatrix} (HH')^B & (HG')^B \\ (GH')^B & (GG')^B \end{bmatrix},$$

use:

$$\Omega = \begin{bmatrix} (HH')^A & 0 & (HG')^A \\ 0 & (HH')^B & (HG')^B \\ (GH')^A & (GH')^B & (GG')^A \end{bmatrix},$$

noting that matrix $(GG')^B$ is lost. This procedure can also be used for the index matrices J_Ω^A and J_Ω^B but, again, where Ω has blocks of zeros, J_Ω must have -1s.

ARMA-plus-noise model. In certain cases, models can be combined in a simple fashion. For example, the ARMA plus noise model is defined as

$$\begin{aligned} y_t &= \mu_t + \epsilon_t, & \epsilon_t &\sim \text{NID}\left(0, \sigma_\epsilon^2\right), \\ \phi(L)\mu_t &= \theta(L)\xi_t, & \xi_t &\sim \text{NID}\left(0, \sigma_\xi^2\right), \end{aligned}$$

where the disturbances are mutually uncorrelated. The state space form of this model is simply

$$y_t = (1, 0, 0, \ldots, 0)\alpha_t + G_t\varepsilon_t,$$

with α_t as given by (10). The *SsfPack* function `GetSsfArma` can be used for the ARMA model, and afterwards, when the element of Ω, associated with GG', is set to a non-zero value, we obtain the ARMA-plus-noise model. A time-varying sequence for G_tG_t' can also be imposed.

SsfPack implementation. Two Ox functions are supplied to facilitate model combination:

```
SsfCombine(mPhiA, mPhiB, dValue);
SsfCombineSym(mOmegaA, cStA, mOmegaB, dValue);
```

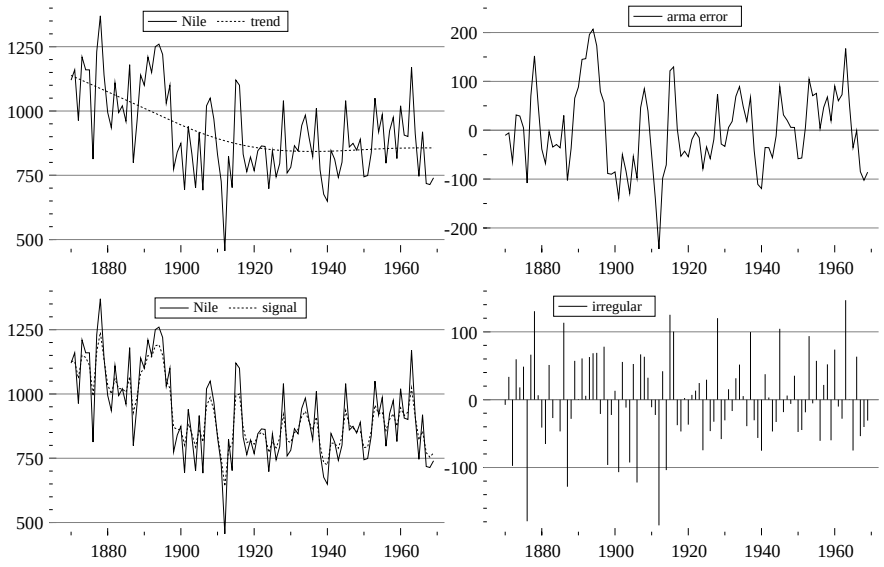
The function `SsfCombine` can be used to create the matrices Φ, Σ, δ (using 0 for the `dValue` argument), as well as for J_Φ and J_δ (using -1 for the `dValue` argument). The function `SsfCombineSym` is used to create Ω and J_Ω, setting `dValue` to 0 and -1 respectively. `SsfCombineSym` requires `cStA`, the dimension m^A of the state vector of model A.

```
SetSplArmaParameters(const vP)
{
    s_vAR = vP[0];                      // AR(1) model
    s_vMA = <>;
    s_q = exp(2. * vP[1]);
}
SplArmaLogLikc(const vY, const pdLik, const pdVar)
{
    decl mphi, momega, msigma, mphiB, momegaB, msigmaB, ret_val;

    GetSsfSpline(s_q, <>, &mphi, &momega, &msigma);
    GetSsfArma(s_vAR, s_vMA, 1, &mphiB, &momegaB, &msigmaB);

    mphi   = SsfCombine(mphi, mphiB, 0);// combining models
    momega = SsfCombineSym(momega, 2, momegaB, 0);
    msigma = SsfCombine(msigma, msigmaB, 0);

    ret_val = SsfLikConc(pdLik, pdVar, vY, mphi, momega, msigma);
    s_dSigma = sqrt(pdVar[0]);          // get sigma from SsfLikConc
    return ret_val;                     // 1 indicates success, 0 failure
}
```



Output generated by ssfsplarma.ox

Listing 16. Part of ssfsplarma.ox with output

Application: Cubic spline model with ARMA errors. A particular example for which we can use the provided functions `SsfCombine` and `SsfCombineSym` is the cubic spline model with a stationary ARMA specification for $\epsilon(t)$ in (15). Standard estimation methods for nonparametric splines as discussed in Green and Silverman (1994) can not

deal with such generalisations, while the state space framework can do this easily. To illustrate this model we consider the Nile data and model it by a cubic spline for the trend with serially correlated errors. The *Ox* code in Listing 16 combines a cubic spline model with AR(1) errors. Apart from the likelihood evaluation, the *Ox* code of `ssfsplarma.ox` is very similar to `ssfair.ox` and `ssfairc.ox`. We see a very smoothly estimated spline in the reported figure, because the remaining 'local' movements around the fitted line are captured by the AR(1) process.

6.3. Regression effects in time-invariant models

Stochastic models such as the ARMA model or the unobserved components model can be extended by including explanatory variables or fixed unknown effects.

Regression model with ARMA errors. For example, we may wish to extended the standard ARMA model by including a constant and a number of regression variables:

$$y_t = \mu^* + x_t'\delta + \mu_t + \epsilon_t,$$

where δ is a vector of regression coefficients, and μ_t is the ARMA part of the model. The state space form of this model is given by

$$\begin{aligned} y_t &= (1, x_t', 1, 0, \ldots, 0)\alpha_t + \varepsilon_t, \\ \alpha_{t+1} &= \begin{pmatrix} \mu^* \\ \delta \\ \alpha^*_{t+1} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & T_a \end{bmatrix} \begin{pmatrix} \mu^* \\ \delta \\ \alpha^*_t \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ h \end{pmatrix} \xi_t, \quad \xi_t \sim \text{NID}\left(0, \sigma^2_\xi\right), \end{aligned}$$

where α^*_t is the state vector of (10), and therefore $\mu_t = (1, 0, \ldots, 0)$. The initial state variance matrix is given by

$$P = \begin{bmatrix} \kappa & 0 & 0 \\ 0 & \kappa I & 0 \\ 0 & 0 & V \end{bmatrix},$$

with κ being the diffuse constant as discussed in §2.3.

Unobserved components and regression effects. Extending equation (11) with regressors gives:

$$y_t = \mu_t + \gamma_t + \psi_t + x_t'\delta + \xi_t, \quad \text{where} \quad \xi_t \sim \text{NID}\left(0, \sigma^2_\xi\right), \qquad t = 1, \ldots, n,$$

with x_t the vector of explanatory variables with coefficients δ. A constant can not be included in the model when μ_t is present: this would cause a problem closely related to the well-known regression problem of multicollinearity. The same applies to the time index as an explanatory variable when the slope term is included in the specification for μ_t. The state space set-up is extended in the same way as for the ARMA model with regression effects.

SsfPack implementation. *SsfPack* provides the function `AddSsfReg` to include regressors to a time-invariant model:

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

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

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

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

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

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

6.4. Monte Carlo simulations and parametric bootstrap tests

Statistical methods such as Monte Carlo and bootstrap require random samples from the unconditional distribution implied by the model in state space form. The *SsfPack* function `SsfRecursion` can be useful in this respect. For illustrative purposes, we will present a simple parametric bootstrap procedure for testing for a unit root when the null is stationarity. This problem has been extensively studied in the literature. The initial work was carried out by Nyblom and Makelainen (1983) and Tanaka (1983), while the more recent work is reviewed in Tanaka (1996, Ch. 10).

Consider the local level model (31) and the vector of univariate observations $y = (y_1, \ldots, y_n)'$. The hypothesis

$$H_0 : \sigma_\eta^2 = 0, \qquad H_1 : \sigma_\eta^2 > 0,$$

implies that y_t is a stationary series under the null hypothesis, and that y_t has a unit root otherwise. The null also implies a constant level, $\mu_0 = \cdots = \mu_n = \mu$, and that the constrained maximum likelihood estimators of μ and σ_ξ^2 are simply the sample average $\overline{y}$ and the sample variance $\widehat{\sigma_\xi^2} = \frac{1}{n}\sum_{t=1}^n (y_t - \overline{y})^2$.

The null hypothesis can be tested using a score test:

$$s = \left. \frac{\partial l(y;\theta)}{\partial \sigma_\eta^2} \right|_{\sigma_\eta^2 = 0, \mu = \overline{y}, \sigma_\xi^2 = \widehat{\sigma_\xi^2}},$$

and the null hypothesis is rejected if the score is relatively large. This statistic (up to a constant) is the same as the locally best invariant (LBI) test, and is known to be asymptotically pivotal; see, for example, Tanaka (1996, Ch. 10.7). The form of the distribution

is complicated, and has to be derived by numerically inverting a characteristic function, or by simulation.

A bootstrap test for the null hypothesis is particularly straightforward for this problem. Define $y^{(i)}$ as a sample of size n drawn from $NID(\overline{y}, \widehat{\sigma_\xi^2})$. Then for each draw the corresponding score statistic is computed:

$$s^{(i)} = \left.\frac{\partial l(y^{(i)};\theta)}{\partial \sigma_\eta^2}\right|_{\sigma_\eta^2 = 0, \mu = \overline{y^{(i)}}, \sigma_\xi^2 = \frac{1}{n}\sum_{t=1}^{n}\left(y_t^{(i)} - \overline{y^{(i)}}\right)^2} .$$

The observed value $\hat{s}$ is compared with a population of simulated score statistics $s^{(i)}$, $j = 1, \ldots, M$, where M is the number of bootstrap replications. This bootstrap test is easily generalized to more general settings.

Interestingly the bootstrap test for the local level model can be made exact if we simulate $y^{(i)}$ in a slightly different way. Define $u^{(i)}$ as a sample of size n drawn from NID$(0, I)$. Transforming the generated sample by

$$y^{*(i)} = \overline{y} + \widehat{\sigma_\xi} \frac{u^{(i)} - \overline{u^{(i)}}}{\sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(u_i^{(i)} - \overline{u^{(i)}}\right)^2}},$$

it follows that:

$$\overline{y^{*(i)}} = \overline{y} \text{ and } \frac{1}{n}\sum_{t=1}^{n}\left(y_t^{*(i)} - \overline{y^{*j}}\right)^2 = \widehat{\sigma_\xi^2}.$$

Thus, under the null hypothesis y^* is being simulated conditionally on the sufficient statistics, consequently, the distribution of y^* is parameter free. As a result, simulations from

$$s^{*(i)} = \left.\frac{\partial l(y^{*j};\theta)}{\partial \sigma_\eta^2}\right|_{\sigma_\eta^2 = 0, \mu = \overline{y}, \sigma_\xi^2 = \widehat{\sigma_\xi^2}} ,$$

provide an exact benchmark for the distribution of s. For example, a test with 5% size can be constructed using 100 simulations by recording $\hat{s}$ and then simulating $s_1^*, \ldots, s_{99}^*$. If $\hat{s}$ is one of the largest five in $\hat{s}, s^{*(1)}, \ldots, s^{*(99)}$ then the hypothesis is rejected.

This exact testing procedure is difficult to extend to more complicated dynamic models and one usually relies on the asymptotic pivotal nature of the score statistic to produce good results.

Application: Bootstrap test of stationarity. The exact testing procedure is implemented for the local level model for the Nile data, with the null hypothesis $\sigma_\eta^2 = 0$ and $M = 1000$. The output in Listing 17 shows that the null hypothesis is strongly rejected.

6.5. Bayesian parameter estimation

The basics. Bayesian inference on parameters indexing models has attracted a great deal of interest recently. Recall that if we have a prior on the parameters φ of $f(\varphi)$, then

```
    SsfLikSco(&dlik, &msco, my, mphi, momega, msigma);
    vboot[0][0] = msco[0][0];       // first is actual test value

    for (i = 1; i < cboot; i++)     // bootstrap loop
    {
        y_i = mn_y + (sd_y * standardize(rann(ct, 1))');
        SsfLikSco(&dlik, &msco, y_i, mphi, momega, msigma);
        vboot[0][i] = msco[0][0];
    }
    vquant = quantiler(vboot, <0.9, 0.95, 0.99>);
```

```
Test for fixed level (Nile data) = 0.832334
                                      90%          95%          99%
Bootstrap critical values:     0.072170      0.10187      0.21229
```

Listing 17. Part of ssfboot.ox with output

$$f(\varphi|y) \propto f(\varphi) \int f(y|\alpha,\varphi) f(\alpha|\varphi) d\alpha = f(\varphi) f(y|\varphi).$$

In the Gaussian case we can evaluate $f(y|\varphi) = \int f(y|\alpha,\varphi) f(\alpha|\varphi) d\alpha$ using the Kalman filter. Although we have the posterior density up to proportionality, it is not easy to compute posterior moments or quantiles about φ, as this involves a further level of integration. Thus it appears as if Bayesian inference is more difficult than maximum likelihood estimation.

However, recent advances in numerical methods for computing functionals of the posterior density $f(\varphi|y)$ have changed this situation. These developments, referred to as Markov chain Monte Carlo (MCMC), consisting of the Metropolis-Hastings algorithm and its special case the Gibbs sampling algorithm, have had a widespread influence on the theory and practice of Bayesian inference; see for example Chib and Greenberg (1996), and Gilks, Richardson, and Spiegelhalter (1996).

The idea behind MCMC methods is to produce variates from a given multivariate density (the posterior density in Bayesian applications) by repeatedly sampling a Markov chain whose invariant distribution is the target density of interest — $f(\varphi|y)$ in the above case. There are typically many different ways of constructing a Markov chain with this property, and an important goal of the literature on MCMC methods in state space models is to isolate those that are simulation efficient. It should be kept in mind that sample variates from a MCMC algorithm are a high-dimensional (correlated) sample from the target density of interest. The resulting draws can be used as the basis for making inferences by appealing to suitable ergodic theorems for Markov chains. For example, posterior moments and marginal densities can be estimated (simulated consistently) by averaging the relevant function of interest over the sampled variates. The posterior mean of φ is simply estimated by the sample mean of the simulated φ values. These estimates can be made arbitrarily accurate by increasing the simulation sample size. The accuracy of the resulting estimates (the so-called numerical standard error) can be assessed by

standard time series methods that correct for the serial correlation in the draws. Indeed, the serial correlation can be quite high for badly behaved algorithms.

To be able to use an MCMC algorithm we need to be able to evaluate the target density up to proportionality. This is the case for our problem as we know $f(\varphi|y) \propto f(\varphi)f(y|\varphi)$ using the Kalman filter. The next subsection will review the nuts and bolts of the sampling mechanism.

Metropolis algorithm. We will use an independence chain Metropolis algorithm to simulate from the abstract joint distribution of $\psi_1, \psi_2, .., \psi_m$. Proposals z are made to possibly replace the current ψ_i, keeping constant $\psi_{\backslash i}$, where $\psi_{\backslash i}$ denotes all elements of ψ except ψ_i. The proposal density is proportional to $q(z, \psi_{\backslash i})$, while the true density is proportional to $f(\psi_i|\psi_{\backslash i})$. Both densities are assumed to be everywhere positive, with compact support and known up to proportionality. If $\psi^{(k)}$ is the current state of the sampler, then the proposal to take $\psi^{(k+1)} = (z, \psi_{\backslash i}^{(k)})$ is accepted if

$$c < \min\left\{\frac{f(z|\psi_{\backslash i}^{(k)})q(\psi_i^{(k)}, \psi_{\backslash i}^{(k)})}{f(\psi_i^{(k)}|\psi_{\backslash i}^{(k)})q(z, \psi_{\backslash i}^{(k)})}, 1\right\}, \qquad \text{where} \qquad c \sim \text{UID}(0,1).$$

If it is rejected, we set $\psi^{(k+1)} = \psi^{(k)}$. Typically, we wish to design $q(\psi_i^{(k)}, \psi_{\backslash i}^{(k)})$ to be close to $f(z|\psi_{\backslash i}^{(k)})$, but preferably with heavier tails (see, for example, Chib and Greenberg (1996)).

In the context of learning about parameters in a Gaussian state space model, this algorithm has $\psi = \varphi|y$. Then the task of performing MCMC on the parameters is one of designing a proposal density $q(\varphi_i^{(k)}, \varphi_{\backslash i}^{(k)})$ which will typically be close to being proportional to $f(\varphi_i|\varphi_{\backslash i}^{(k)}, y) \propto f(\varphi_i, \varphi_{\backslash i}^{(k)}|y)$. This is not particularly easy to do, although generic methods are available: see, for example, Gilks, Best, and Tan (1995).

In the rather simpler case where we can choose

$$q(\psi_i, \psi_{\backslash i}^{(k)}) = f(\psi_i|\psi_{\backslash i}^{(k)}),$$

the Metropolis algorithm is called a Gibbs sampler; see Geman and Geman (1984) and Gelfand and Smith (1990). In that case the suggestions are never rejected. Unfortunately, for the unknown parameter problems in a Gaussian model, $f(\varphi_i|\varphi_{\backslash i}^{(k)}, y)$ is only known up to proportionality, and consequently the simplicity of the Gibbs sampler is not available.

Augmentation. As the design of proposal densities for the Metropolis algorithm is sometimes difficult an alternative method has been put forward by Fruhwirth-Schnatter (1994). This suggestion is of added interest because it is the only available way to make progress when we move to non-Gaussian problems, where evaluating $f(y|\varphi) = \int f(y|\alpha, \varphi)f(\alpha|\varphi)d\alpha$ is generally not possible.

The suggestion is to design MCMC methods for simulating from the density $\pi(\varphi, \alpha|y)$, where $\alpha = (\alpha_1, \ldots, \alpha_n)$ is the vector of n latent states, rather than $\pi(\varphi|y)$. The draws from this joint density provide draws from the marginal density $\pi(\varphi|y)$, by simply ignoring the draws from the states, and therefore solve the original problem. It turns out

that rather simple Markov chain Monte Carlo procedures can be developed to sample $\pi(\varphi, \alpha|y)$. In particular we could

1. Initialize φ
2. Sample from the multivariate Gaussian distribution of $\alpha|y, \varphi$ using a simulation smoother.
3. Sample from $\varphi|y, \alpha$ directly or do a Gibbs or Metropolis update on the elements.
4. Go to 2.

The key features are that the simulation smoother allows all the states to be drawn as a block in a simple and generic way, and secondly that we can usually draw from $\varphi|y, \alpha$ in a relatively trivial way. This second point is illustrated in the next section.

Illustration. Suppose the model is a local linear trend (12) with added measurement error $\xi_t \sim \text{NID}(0, \sigma_\xi^2)$. When we draw from $\varphi|y, \alpha$ we act as if y, α is known. Knowing α gives us both $\{\mu_t\}$ and $\{\beta_t\}$. Thus we can unwrap the disturbances

$$\begin{array}{rcll} \eta_t & = & \mu_{t+1} - \mu_t - \beta_t & \sim \text{NID}(0, \sigma_\eta^2), \\ \zeta_t & = & \beta_{t+1} - \beta_t & \sim \text{NID}(0, \sigma_\zeta^2), \\ \xi_t & = & y_t - \mu_t & \sim \text{NID}(0, \sigma_\xi^2). \end{array}$$

Let the prior densities be given by

$$\sigma_\xi^2 \sim \mathcal{IG}\left(\frac{c_\xi}{2}, \frac{S_{\sigma_\xi}}{2}\right), \qquad \sigma_\eta^2 \sim \mathcal{IG}\left(\frac{c_\eta}{2}, \frac{S_{\sigma_\eta}}{2}\right), \qquad \sigma_\zeta^2 \sim \mathcal{IG}\left(\frac{c_\zeta}{2}, \frac{S_{\sigma_\zeta}}{2}\right),$$

for some choices of shape parameters c_ξ, c_η, c_ζ and scales $S_{\sigma_\xi}, S_{\sigma_\eta}, S_{\sigma_\zeta}$. For example, the inverse gamma distribution $\mathcal{IG}$ for σ_ξ^2 implies that the prior mean and variance of σ_ξ^2 is given by

$$\frac{S_{\sigma_\xi}}{c_\xi - 2}, \qquad \frac{2S_{\sigma_\xi}^2}{(c_\xi - 2)^2 (c_\xi - 4)},$$

respectively. The posteriors are then given by

$$\sigma_\xi^2|y, \alpha \sim \mathcal{IG}\left(\frac{c_\xi + n}{2}, \frac{S_{\sigma_\xi} + \sum \xi_t^2}{2}\right), \quad \sigma_\eta^2|y, \alpha \sim \mathcal{IG}\left(\frac{c_\eta + n}{2}, \frac{S_{\sigma_\eta} + \sum \eta_t^2}{2}\right),$$
$$\sigma_\zeta^2|y, \alpha \sim \mathcal{IG}\left(\frac{c_\zeta + n}{2}, \frac{S_{\sigma_\zeta} + \sum \zeta_t^2}{2}\right).$$

Each of these densities are easy to sample from as shown in the *Ox* example program.

Although it is not always possible to sample the $\varphi|y, \alpha$ this easily, it is usually the case that it is much easier to update the parameters having augmented the MCMC with the states, than when the states are integrated out. Of course, it is often the case that the MCMC algorithm has such a large dimension that the algorithm converges rather slowly. This danger needs to be assessed carefully in applied work.

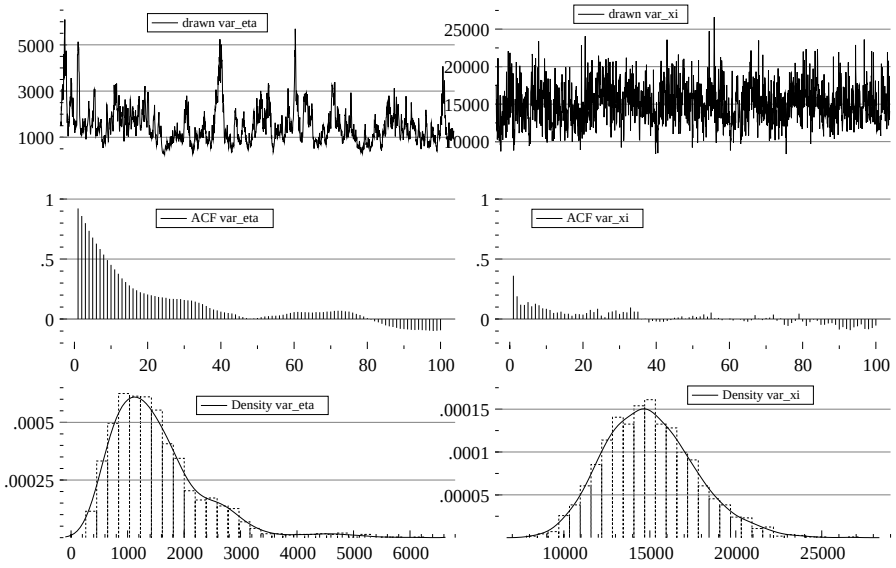
Application: Bayesian estimation of local level model. The Bayesian procedure for σ_ξ^2 and σ_η^2 is implemented for the local linear trend model with $\beta_t = 0$ (i.e. the local level model(31)) using the Nile data; see Listing 18. The prior density parameters are set to $c_\eta = c_\xi = 5$ and $S_\eta = 5000, S_\xi = 5000$. We use 2000 replications.

```
    GetSsfStsm(<CMP_LEVEL, 1.0, 0, 0;
                CMP_IRREG, 1.0, 0, 0>, &mphi, &momega, &msigma);

    s_eta = 5000; s_xi = 50000;
    c_eta = c_eps = 2.5 + (0.5 * columns(myt));

    for (i = 0, mpsi = zeros(2, crep); i < crep; ++i)
    {
        md = SsfCondDens(DS_SIM, myt, mphi, momega, msigma);
        md = md * md';
        mpsi[0][i] = 1.0 / rangamma(1,1, c_eta, (s_eta + md[0][0])/2);
        mpsi[1][i] = 1.0 / rangamma(1,1, c_eps, (s_xi + md[1][1])/2);
        momega = diag(mpsi[][i]);
    }
```

```
                        mean        st.dev.
var_eta              1519.168        818.687
var_eps             15011.850       2637.460
```



Histogram and estimated density of σ_ε^2 and σ_η^2

Listing 18. Part of ssfbayes.ox with output

7. CONCLUSION

In this paper we have discussed *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. We have shown that a wide variety of models can be handled in this unified framework: from a simple regression model with ARMA errors to a Bayesian model with unobserved components. Many applications are given and the *Ox* code is provided. They illustrate the enormous flexibility of this approach. Furthermore, *SsfPack* allows the researcher to concentrate on the problem at hand, rather than on programming issues. Here we have concentrated on Gaussian univariate models, but the algorithms can deal as easily with multivariate models and with certain classes of non-Gaussian model. For an example of the latter, which uses *SsfPack*, see the stochastic volatility models in Kim, Shephard, and Chib (1998). A general overview is given in Koopman, Shephard, and Doornik (1998).

Although the algorithms are implemented using efficiently written computer code, *SsfPack* can be relatively slow when the model implies a large state vector (for example, when we deal with monthly observations). This is mainly due to the generality of the package: the algorithms do not take account of sparse structures in system matrices. Some ARMA models and unobserved components models imply sparse system matrices; this happened with the airline model (§5.1), for which maximization was relatively slow. We are currently developing algorithms which are able to recognise sparse matrix structures without losing the generality of *SsfPack*.

The Kalman filter and smoothing algorithms as implemented in *SsfPack* are not able to take account of diffuse initial conditions. We have solved this by setting the initial variances associated with diffuse elements of the state vector to a large value (the so-called big-κ method). However, there are methods available to address this issue in an exact way. The next release of *SsfPack* will provide such routines for filtering and smoothing which are based on the methods of Koopman (1997).

A. APPENDIX: SSFPACK FUNCTIONS AND SAMPLE PROGRAMS

Models in state space form

`AddSsfReg`	§6.2	adds regression effect to time-invariant state space.
`GetSsfArma`	§3.1	puts ARMA model in state space.
`GetSsfReg`	§3.3	puts regression model in state space.
`GetSsfSpline`	§3.4	puts nonparametric cubic spline model in state space.
`GetSsfStsm`	§3.2	puts structural time series model in state space.
`SsfCombine`	§6.2	combines system matrices of two models.
`SsfCombineSym`	§6.2	combines symmetric system matrices of two models.

General state space algorithms

`KalmanFil`	§4.3	returns output of the Kalman filter.
`KalmanSmo`	§4.4	returns output of the basic smoothing algorithm.
`SimSmoDraw`	§4.5	returns a sample from the simulation smoother.
`SimSmoWgt`	§4.5	returns covariance output of the simulation smoother.

Ready-to-use functions

`SsfCondDens`	§4.6	returns mean or a draw from the conditional density.
`SsfLik`	§5.1	returns log-likelihood function.
`SsfLikConc`	§5.1	returns profile log-likelihood function.
`SsfLikSco`	§5.1	returns score vector.
`SsfMomentEst`	§5.2, §5.3	returns output from prediction, forecasting and smoothing.
`SsfRecursion`	§4.2	returns output of the state space recursion.

		`GetSsfArma`			`GetSsfStsm`		`SimSmoDraw`		`SsfLik`		`SsfMomentEst`			
Listing		`GetSsfReg`			`KalmanFil`		`SimSmoWgt`		`SsfLikConc`		`SsfRecursion`			
program			`GetSsfSpline`		`KalmanSmo`		`SsfCondDens`		`SsfLikSco`			`SsfCombine`		
`ssfair`	9	X			.		.		X			.		
`ssfairc`	10	X	.		.	.	.	.	.	X		.	.	
`ssfairf`	11	X	.	.	.	.	.	.	.	.	X	.	X	.	.
`ssfairstsm`	15	.	.	.	X	.	.	.	.	X	.	X	.	.	.
`ssfarma`	1	X	.	.	.	.	.	.	.	.	.	.	.	.	.
`ssfbayes`	18	.	.	.	X	.	.	.	X	.	.	.	.	.	.
`ssfboot`	17	.	.	.	X	.	.	.	.	.	.	X	.	.	.
`ssfkf`	6	.	.	.	.	X	.	.	.	.	.	.	.	X	.
`ssfnile`	12	.	.	.	X	.	.	.	.	X	.	X	X	.	.
`ssfnilesp`	14	.	.	X	.	.	.	.	.	X	.	.	X	.	.
`ssfrec`	5	.	.	.	.	.	.	.	.	.	.	.	.	X	.
`ssfreg`	3	.	X	.	.	.	.	.	.	.	.	.	.	.	.
`ssfsim`	8	.	.	.	.	X	.	X	X	.	.	.	.	X	.
`ssfsmo`	7	.	.	.	.	X	X	.	.	.	.	.	.	X	.
`ssfspirits`	13	.	X	.	.	X	X	.	.	.	X	.	X	.	.
`ssfspl`	4	.	.	X	.	.	.	.	.	.	.	.	.	.	.
`ssfsplarma`	16	X	.	X	.	.	.	.	.	X	X	X	.	.	.
`ssfstsm`	2	.	.	.	X	.	.	.	.	.	.	.	.	.	.

REFERENCES

Anderson, B. D. O. and J. B. Moore (1979). *Optimal Filtering*. Englewood Cliffs: Prentice-Hall.

Ansley, C. F. and R. Kohn (1986). A note on reparameterizing a vector autoregressive moving average model to enforce stationarity. *J. Statistical Computation and Simulation 24*, 99–106.

Balke, N. S. (1993). Detecting level shifts in time series. *J. Business and Economic Statist. 11*, 81–92.

Bergstrom, A. R. (1984). Gaussian estimation of structural parameters in higher order continuous time dynamic models. In Z. Griliches and M. Intriligator (Eds.), *The Handbook of Econometrics, Volume 2*, pp. 1145–1212. North-Holland.

Box, G. E. P. and G. M. Jenkins (1976). *Time Series Analysis: Forecasting and Control* (2nd ed.). San Francisco, CA: Holden-Day.

Chib, S. and E. Greenberg (1996). Markov chain Monte Carlo simulation methods in econometrics. *Econometric Theory 12*, 409–31.

Cobb, G. W. (1978). The problem of the Nile: conditional solution to a change point problem. *Biometrika 65*, 243–51.

de Jong, P. (1988a). A cross validation filter for time series models. *Biometrika 75*, 594–600.

de Jong, P. (1988b). The likelihood for a state space model. *Biometrika 75*, 165–169.

de Jong, P. (1989). Smoothing and interpolation with the state space model. *J. American Statistical Association 84*, 1085–8.

de Jong, P. and J. Penzer (1998). Diagnosing shocks in time series. *J. American Statistical Association 93*, 796–806.

de Jong, P. and N. Shephard (1995). The simulation smoother for time series models. *Biometrika 82*, 339–50.

Doornik, J. A. (1998). *Object-Oriented Matrix Programming using Ox 2.0*. London: Timberlake Consultants Press.

Fruhwirth-Schnatter, S. (1994). Data augmentation and dynamic linear models. *J. Time Series Analysis 15*, 183–202.

Gelfand, A. E. and A. F. M. Smith (1990). Sampling-based approaches to calculating marginal densities. *J. American Statistical Association 85*, 398–409.

Geman, S. and D. Geman (1984). Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. *IEEE Transactions, PAMI 6*, 721–41.

Gilks, W. K., S. Richardson, and D. J. Spiegelhalter (1996). *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.

Gilks, W. R., N. G. Best, and K. K. C. Tan (1995). Adaptive rejection Metropolis sampling within Gibbs sampling. *Applied Statistics 44*, 155–73.

Green, P. and B. W. Silverman (1994). *Nonparameteric Regression and Generalized Linear Models: A Roughness Penalty Approach*. London: Chapman & Hall.

Harrison, J. and C. F. Stevens (1976). Bayesian forecasting (with discussion). *J. Royal Statistical Society B 38*, 205–247.

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

Harvey, A. C. (1993). *Time Series Models* (2nd ed.). Hemel Hempstead: Harvester Wheatsheaf.

Harvey, A. C. and S. J. Koopman (1992). Diagnostic checking of unobserved components time series models. *J. Business and Economic Statist. 10*, 377–389.

Harvey, A. C., S. J. Koopman, and J. Penzer (1998). Messy time series. In T. B. Fomby and R. C. Hill (Eds.), *Advances in Econometrics, volume 13*. New York: JAI Press.

Harvey, A. C. and M. Streibel (1998). Testing for nonstationary unobserved components. *J. Time Series Analysis 19*. Forthcoming.

Hastie, T. and R. Tibshirani (1990). *Generalized Additive Models*. London: Chapman & Hall.

Jones, R. H. (1980). Maximum likelihood fitting of ARIMA models to time series with missing observations. *Technometrics 22*, 389–95.

Kim, S., N. Shephard, and S. Chib (1998). Stochastic volatility: likelihood inference and comparison with ARCH models. *Rev. Economic Studies 65*, 361–93.

Kitagawa, G. and W. Gersch (1996). *Smoothness Priors Analysis of Time Series*. New York: Springer Verlag.

Kohn, R. and C. F. Ansley (1987). A new algorithm for spline smoothing based on smoothing a stochastic process. *SIAM J Sci. Statistical Computing 8*, 33–48.

Kohn, R. and C. F. Ansley (1989). A fast algorithm for signal extraction, influence and cross-validation. *Biometrika 76*, 65–79.

Koopman, S. J. (1992). *Diagnostic Checking and Intra-daily Effects in Time Series Models*, Volume 27 of *Tinbergen Institute Research Series*. Amsterdam: Thesis Publishers.

Koopman, S. J. (1993). Disturbance smoother for state space models. *Biometrika 80*, 117–126.

Koopman, S. J. (1997). Exact initial Kalman filtering and smoothing for non-stationary time series models. *J. American Statistical Association 92*, 1630–1638.

Koopman, S. J. (1998). Kalman filtering and smoothing. In P. Armitage and T. Colton (Eds.), *Encyclopedia of Biostatistics*. Chichester: Wiley and Sons.

Koopman, S. J. and N. Shephard (1992). Exact score for time series models in state space form. *Biometrika 79*, 823–6.

Koopman, S. J., N. Shephard, and J. A. Doornik (1998). Fitting non-Gaussian state space models in econometrics: Overview, developments and software. Unpublished paper.

Magnus, J. R. and H. Neudecker (1988). *Matrix Differential Calculus with Applications in Statistics and Econometrics*. New York: Wiley.

Nyblom, J. and T. Makelainen (1983). Comparison of tests of for the presence of random walk coefficients in a simple linear models. *J. American Statistical Association 78*, 856–64.

Prest, A. R. (1949). Some experiments with demand analysis. *Review of Economics and Statistics 31*, 33–49.

Schweppe, F. (1965). Evaluation of likelihood functions for Gaussian signals. *IEEE Transactions on Information Theory 11*, 61–70.

Tanaka, K. (1983). Non-normality of the Lagrange multiplier statistics for testing the constancy of regression coefficients. *Econometrica 51*, 1577–82.

Tanaka, K. (1996). *Time Series Analysis: Nonstationary and Noninvertible Distribution Theory*. New York: Wiley.

Tunnicliffe-Wilson, G. (1989). On the use of marginal likelihood in time series model estimation. *J. Royal Statistical Society B 51*, 15–27.

Wecker, W. E. and C. F. Ansley (1983). The signal extraction approach to nonlinear regression and spline smoothing. *J. American Statistical Association 78*, 81–89.

West, M. and J. Harrison (1997). *Bayesian Forecasting and Dynamic Models* (2 ed.). New York: Springer-Verlag.