14
State Space Models

14.1 Introduction

The state space modeling tools in `S+FinMetrics` are based on the algorithms in *SsfPack* 3.0 developed by Siem Jan Koopman and described in Koopman, Shephard and Doornik (1999, 2001)[1]. *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 routines allow for a variety of state space forms from simple time invariant models to complicated time-varying models. Functions are available to put standard models like ARMA and spline models in state space form. General routines are available for filtering, smoothing, simulation smoothing, likelihood evaluation, forecasting and signal extraction. Full details of the statistical analysis is provided in Durbin and Koopman (2001). This chapter gives an overview of state space modeling and the reader is referred to the papers by Koopman, Shephard and Doornik for technical details on the algorithms used in the `S+FinMetrics/SsfPack` functions.

This chapter is organized as follows. Section 14.2 describes the general state space model and state space representation required for the `S+FinMetrics/SsfPack` state space functions. Subsections describe the various `S+FinMetrics/SsfPack` functions for putting common time series models into state space form. The process of simulating observations from a given state space model is also covered. Section 14.3 summarizes the main

[1] Information about *Ssfpack* can be found at `http://www.ssfpack.com`.

algorithms used for the analysis of state space models. These include the Kalman filter, Kalman smoother, moment smoothing, disturbance smoothing and forecasting. Estimation of the unknown parameters in a state space model is described in Section 14.4. The chapter concludes with a short discussion of simulation smoothing.

Textbook treatments of state space models are given in Harvey (1989, 1993), Hamilton (1994), West and Harrison (1997), and Kim and Nelson (1999). Interesting applications of state space models in finance are given in Engle and Watson (1987), Bomhoff (1994), Duan and Simonato (1999), and Harvey, Ruiz and Shephard (1994), Carmona (2001) and Chan (2002).

14.2 State Space Representation

Many dynamic time series models in economics and finance may be represented in *state space form*. Some common examples are ARMA models, time-varying regression models, dynamic linear models with unobserved components, discrete versions of continuous time diffusion processes, stochastic volatility models, non-parametric and spline regressions. The linear Gaussian *state space model* may be represented as the system of equations

$$\underset{m\times 1}{\boldsymbol{\alpha}_{t+1}} = \underset{m\times 1}{\mathbf{d}_t} + \underset{m\times m}{\mathbf{T}_t} \cdot \underset{m\times 1}{\boldsymbol{\alpha}_t} + \underset{m\times r}{\mathbf{H}_t} \cdot \underset{r\times 1}{\boldsymbol{\eta}_t} \tag{14.1}$$

$$\underset{N\times 1}{\boldsymbol{\theta}_t} = \underset{N\times 1}{\mathbf{c_t}} + \underset{N\times m}{\mathbf{Z_t}} \cdot \underset{m\times 1}{\boldsymbol{\alpha}_t} \tag{14.2}$$

$$\underset{N\times 1}{\mathbf{y}_t} = \underset{N\times 1}{\boldsymbol{\theta}_t} + \underset{N\times N}{\mathbf{G}_t} \cdot \underset{N\times 1}{\boldsymbol{\varepsilon}_t} \tag{14.3}$$

where $t = 1, \ldots, n$ and

$$\boldsymbol{\alpha}_1 \sim N(\mathbf{a}, \mathbf{P}), \tag{14.4}$$

$$\boldsymbol{\eta}_t \sim iid\ N(0, \mathbf{I}_r) \tag{14.5}$$

$$\boldsymbol{\varepsilon}_t \sim iid\ N(\mathbf{0}, \mathbf{I}_N) \tag{14.6}$$

and it is assumed that

$$E[\boldsymbol{\varepsilon}_t \boldsymbol{\eta}_t'] = \mathbf{0}$$

In (14.4), $\mathbf{a}$ and $\mathbf{P}$ are fixed and known but that can be generalized. The *state vector* $\boldsymbol{\alpha}_t$ contains unobserved stochastic processes and unknown fixed effects and the *transition equation* (14.1) describes the evolution of the state vector over time using a first order Markov structure. The *measurement equation* (14.3) describes the vector of observations $\mathbf{y}_t$ in terms of the state vector $\boldsymbol{\alpha}_t$ through the signal $\boldsymbol{\theta}_t$ and a vector of disturbances $\boldsymbol{\varepsilon}_t$. It is assumed that the innovations in the transition equation and the innovations in the measurement equation are independent, but this assumption

can be relaxed. The deterministic matrices $\mathbf{T}_t$, $\mathbf{Z}_t$, $\mathbf{H}_t$, $\mathbf{G}_t$ are called *system matrices* and are usually sparse selection matrices. The vectors $\mathbf{d}_t$ and $\mathbf{c}_t$ contain fixed components and may be used to incorporate known effects or known patterns into the model; otherwise they are equal to zero.

The state space model (14.1)-(14.6) may be compactly expressed as

$$\underset{(m+N)\times 1}{\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ \mathbf{y}_t \end{pmatrix}} = \underset{(m+N)\times 1}{\boldsymbol{\delta}_t} + \underset{(m+N)\times m}{\boldsymbol{\Phi}_t} \cdot \underset{m\times 1}{\boldsymbol{\alpha}_t} + \underset{(m+N)\times 1}{\mathbf{u}_t}, \tag{14.7}$$

$$\boldsymbol{\alpha}_1 \sim N(\mathbf{a}, \mathbf{P}) \tag{14.8}$$

$$\mathbf{u}_t \sim iid\ N(\mathbf{0}, \boldsymbol{\Omega}_t) \tag{14.9}$$

where

$$\boldsymbol{\delta}_t = \begin{pmatrix} \mathbf{d}_t \\ \mathbf{c}_t \end{pmatrix}, \boldsymbol{\Phi}_t = \begin{pmatrix} \mathbf{T}_t \\ \mathbf{Z}_t \end{pmatrix}, \mathbf{u}_t = \begin{pmatrix} \mathbf{H}_t\boldsymbol{\eta}_t \\ \mathbf{G}_t\boldsymbol{\varepsilon}_t \end{pmatrix}, \boldsymbol{\Omega}_t = \begin{pmatrix} \mathbf{H}_t\mathbf{H}_t' & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_t\mathbf{G}_t' \end{pmatrix}$$

The initial value parameters are summarized in the $(m+1)\times m$ matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \mathbf{P} \\ \mathbf{a}' \end{pmatrix} \tag{14.10}$$

For multivariate models, i.e. $N > 1$, it is assumed that the $N \times N$ matrix $\mathbf{G}_t\mathbf{G}_t'$ is diagonal. In general, the system matrices in (14.7) are time varying.

14.2.1 Initial Conditions

The variance matrix $\mathbf{P}$ of the initial state vector $\boldsymbol{\alpha}_1$ is assumed to be of the form

$$\mathbf{P} = \mathbf{P}_* + \kappa\mathbf{P}_\infty \tag{14.11}$$

where $\mathbf{P}_\infty$ and $\mathbf{P}_*$ are symmetric $m \times m$ matrices with ranks r_∞ and r_*, respectively, and κ is a large scalar value, e.g. $\kappa = 10^6$. The matrix $\mathbf{P}_*$ captures the covariance structure of the stationary components in the initial state vector, and the matrix $\mathbf{P}_\infty$ is used to specify the initial variance matrix for nonstationary components. When the ith diagonal element of $\mathbf{P}_\infty$ is negative, the corresponding ith column and row of $\mathbf{P}_*$ are assumed to be zero, and the corresponding row and column of $\mathbf{P}_\infty$ will be taken into consideration. When some elements of state vector are nonstationary, the `S+FinMetrics/SsfPack` algorithms implement an "exact diffuse prior" approach as described in Durbin and Koopman (2001) and Koopman, Shephard and Doornik (2001).

14.2.2 State Space Representation in S+FinMetrics/SsfPack

State space models in `S+FinMetrics/SsfPack` utilize the compact representation (14.7) with initial value information (14.10). The following ex-

amples describe the specification of a state space model for use in the `S+FinMetrics/SsfPack` state space modeling functions.

Example 92 *State space representation of the local level model*

Consider the following simple model for the stochastic evolution of the logarithm of an asset price y_t

$$\begin{aligned} \alpha_{t+1} &= \alpha_t + \eta_t^*,\ \eta_t^* \sim iid\ N(0, \sigma_\eta^2) && (14.12) \\ y_t &= \alpha_t + \varepsilon_t^*,\ \varepsilon_t^* \sim iid\ N(0, \sigma_\varepsilon^2) && (14.13) \\ \alpha_1 &\sim N(a, P) && (14.14) \end{aligned}$$

where it is assumed that $E[\varepsilon_t^* \eta_t^*] = 0$. In the above model, the observed asset price y_t is the sum of two unobserved components, α_t and ε_t^*. The component α_t is the state variable and represents the fundamental value (signal) of the asset. The transition equation (14.12) shows that the fundamental values evolve according to a random walk. The component ε_t^* represents random deviations (noise) from the fundamental value that are assumed to be independent from the innovations to α_t. The strength of the signal in the fundamental value relative to the random deviation is measured by the signal-to-noise ratio of variances $q = \sigma_\eta^2/\sigma_\varepsilon^2$. The model (14.12)-(14.14) is called the *random walk plus noise model*, *signal plus noise model* or the *local level model*.[2]

The state space form (14.7) of the local level model has time invariant parameters

$$\boldsymbol{\delta} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Phi = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Omega = \begin{pmatrix} \sigma_\eta^2 & 0 \\ 0 & \sigma_\varepsilon^2 \end{pmatrix} \tag{14.15}$$

with errors $\sigma_\eta \eta_t = \eta_t^*$ and $\sigma_\varepsilon \varepsilon_t = \varepsilon_t^*$. Since the state variable α_t is $I(1)$, the unconditional distribution of the initial state α_1 doesn't have finite variance. In this case, it is customary to set $a = E[\alpha_1] = 0$ and $P = var(\alpha_1)$ to some large positive number, e.g. $P = 10^7$, in (14.14) to reflect that no prior information is available. Using (14.11), the initial variance is specified with $P_* = 0$ and $P_\infty = 1$. Therefore, the initial state matrix (14.10) for the local level model has the form

$$\Sigma = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \tag{14.16}$$

where -1 implies that $P_\infty = 1$.

In `S+FinMetrics/SsfPack`, a state space model is specified by creating either a list variable with components giving the minimum components

[2] A detailed technical analysis of this model is given in Durbin and Koopman (2001), chapter 2.

State space parameter	List component name
$\boldsymbol{\delta}$	mDelta
$\boldsymbol{\Phi}$	mPhi
$\boldsymbol{\Omega}$	mOmega
$\boldsymbol{\Sigma}$	mSigma

TABLE 14.1. State space form list components

necessary for describing a particular state space form or by creating an "ssf" object. To illustrate, consider creating a list variable containing the state space parameters in (14.15)-(14.16), with $\sigma_\eta = 0.5$ and $\sigma_\varepsilon = 1$

```
> sigma.e = 1
> sigma.n = 0.5
> a1 = 0
> P1 = -1
> ssf.ll.list = list(mPhi=as.matrix(c(1,1)),
+ mOmega=diag(c(sigma.n^2,sigma.e^2)),
+ mSigma=as.matrix(c(P1,a1)))
> ssf.ll.list
$mPhi:
     [,1]
[1,]    1
[2,]    1

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    1

$mSigma:
     [,1]
[1,]   -1
[2,]    0
```

In the list variable ssf.ll.list, the component names match the state space form parameters in (14.7) and (14.10). This naming convention, summarized in Table 14.1, must be used for the specification of any valid state space model. Also, notice the use of the coercion function as.matrix. This ensures that the dimensions of the state space parameters are correctly specified.

An "ssf" object may be created from the list variable ssf.ll.list using the S+FinMetrics/SsfPack function CheckSsf:

```
> ssf.ll = CheckSsf(ssf.ll.list)
> class(ssf.ll)
[1] "ssf"
```

```
> names(ssf.ll)
 [1] "mDelta"  "mPhi"    "mOmega"  "mSigma"  "mJPhi"
 [6] "mJOmega" "mJDelta" "mX"      "cT"      "cX"
[11] "cY"      "cSt"
> ssf.ll
$mPhi:
     [,1]
[1,]    1
[2,]    1

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    1

$mSigma:
     [,1]
[1,]   -1
[2,]    0

$mDelta:
     [,1]
[1,]    0
[2,]    0

$mJPhi:
[1] 0

$mJOmega:
[1] 0

$mJDelta:
[1] 0

$mX:
[1] 0

$cT:
[1] 0

$cX:
[1] 0

$cY:
[1] 1
```

```
$cSt:
[1] 1

attr(, "class"):
[1] "ssf"
```

The function `CheckSsf` takes a list variable with a minimum state space form, coerces the components to matrix objects and returns the full parameterization of a state space model used in many of the `S+FinMetrics/SsfPack` state space modeling functions. See the online help for `CheckSsf` for descriptions of the components of an "`ssf`" object.

Example 93 *State space representation of a time varying parameter regression model*

Consider the Capital Asset Pricing Model (CAPM) with time varying intercept and slope

$$\begin{aligned} r_t &= \alpha_t + \beta_t r_{M,t} + \nu_t,\ \nu_t \sim GWN(0, \sigma_\nu^2) && (14.17)\\ \alpha_{t+1} &= \alpha_t + \xi_t,\ \xi_t \sim GWN(0, \sigma_\xi^2) && (14.18)\\ \beta_{t+1} &= \beta_t + \varsigma_t,\ \varsigma_t \sim GWN(0, \sigma_\varsigma^2) && (14.19) \end{aligned}$$

where r_t denotes the return on an asset in excess of the risk free rate, and $r_{M,t}$ denotes the excess return on a market index. In this model, both the abnormal excess return α_t and asset risk β_t are allowed to vary over time following a random walk specification. Let $\boldsymbol{\alpha}_t = (\alpha_t, \beta_t)'$, $y_t = r_t$, $\mathbf{x}_t = (1, r_{M,t})'$, $\mathbf{H}_t = diag(\sigma_\xi, \sigma_\varsigma)'$ and $G_t = \sigma_\nu$. Then the state space form (14.7) of (14.17) - (14.19) is

$$\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} = \begin{pmatrix} \mathbf{I}_2 \\ \mathbf{x}_t' \end{pmatrix} \boldsymbol{\alpha}_t + \begin{pmatrix} \mathbf{H}\boldsymbol{\eta}_t \\ G\varepsilon_t \end{pmatrix}$$

and has parameters

$$\boldsymbol{\Phi}_t = \begin{pmatrix} \mathbf{I}_2 \\ \mathbf{x}_t' \end{pmatrix},\ \boldsymbol{\Omega} = \begin{pmatrix} \sigma_\xi^2 & 0 & 0 \\ 0 & \sigma_\varsigma^2 & 0 \\ 0 & 0 & \sigma_\nu^2 \end{pmatrix} \quad (14.20)$$

Since $\boldsymbol{\alpha}_t$ is $I(1)$ the initial state vector $\boldsymbol{\alpha}_1$ doesn't have finite variance so it is customary to set $\mathbf{a} = \mathbf{0}$ and $\mathbf{P} = \kappa \mathbf{I}_2$ where κ is large. Using (14.11), the initial variance is specified with $\mathbf{P}_* = \mathbf{0}$ and $\mathbf{P}_\infty = \mathbf{I}_2$. Therefore, the initial state matrix (14.10) for the time varying CAPM has the form

$$\boldsymbol{\Sigma} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}$$

The state space parameter matrix $\mathbf{\Phi}_t$ in (14.20) has a time varying system element $\mathbf{Z}_t = \mathbf{x}_t'$. In `S+FinMetrics/SsfPack`, the specification of this time varying element in $\mathbf{\Phi}_t$ requires an index matrix $\mathbf{J}_\Phi$ and a data matrix $\mathbf{X}$ to which the indices in $\mathbf{J}_\Phi$ refer. The index matrix $\mathbf{J}_\Phi$ must have the same dimension as $\mathbf{\Phi}_t$. The elements of $\mathbf{J}_\Phi$ are all set to -1 except the elements for which the corresponding elements of $\mathbf{\Phi}_t$ are time varying. The non-negative index value indicates the column of the data matrix $\mathbf{X}$ which contains the time varying values.

The specification of the state space form for the time varying CAPM requires values for the variances σ_ξ^2, σ_ς^2, and σ_ν^2 as well as a data matrix $\mathbf{X}$ whose rows correspond with $\mathbf{Z}_t = \mathbf{x}_t' = (1, r_{M,t})$. For example, let $\sigma_\xi^2 = (0.01)^2$, $\sigma_\varsigma^2 = (0.05)^2$ and $\sigma_\nu^2 = (0.1)^2$ and construct the data matrix $\mathbf{X}$ using the excess return data in the `S+FinMetrics` "`timeSeries`" `excessReturns.ts`

```
> X.mat = cbind(1,
+ as.matrix(seriesData(excessReturns.ts[,"SP500"]))
```

The state space form may be created using

```
> Phi.t = rbind(diag(2),rep(0,2))
> Omega = diag(c((.01)^2,(.05)^2,(.1)^2))
> J.Phi = matrix(-1,3,2)
> J.Phi[3,1] = 1
> J.Phi[3,2] = 2
> Sigma = -Phi.t
> ssf.tvp.capm = list(mPhi=Phi.t,
+ mOmega=Omega,
+ mJPhi=J.Phi,
+ mSigma=Sigma,
+ mX=X.mat)
> ssf.tvp.capm
$mPhi:
     [,1] [,2]
[1,]    1    0
[2,]    0    1
[3,]    0    0

$mOmega:
       [,1]   [,2] [,3]
[1,] 0.0001 0.0000 0.00
[2,] 0.0000 0.0025 0.00
[3,] 0.0000 0.0000 0.01

$mJPhi:
     [,1] [,2]
```

Parameter index matrix	List component name
$\mathbf{J}_\delta$	mJDelta
$\mathbf{J}_\Phi$	mJPhi
$\mathbf{J}_\Omega$	mJOmega
Time varying component data matrix	**List component name**
$\mathbf{X}$	mX

TABLE 14.2. S+FinMetrics time varying state space components

```
[1,]   -1   -1
[2,]   -1   -1
[3,]    1    2

$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mX:
numeric matrix: 131 rows, 2 columns.
         SP500
 1 1  0.002803
 2 1  0.017566
...
131 1 -0.0007548
```

Notice in the specification of $\mathbf{\Phi}_t$ the values associated with $\mathbf{x}_t'$ in the third row are set to zero. In the index matrix $\mathbf{J}_\Phi$, the (3,1) element is 1 and the (3,2) element is 2 indicating that the data for the first and second columns of $\mathbf{x}_t'$ come from the first and second columns of the component mX, respectively.

In the general state space model (14.7), it is possible that all of the system matrices $\boldsymbol{\delta}_t$, $\mathbf{\Phi}_t$ and $\mathbf{\Omega}_t$ have time varying elements. The corresponding index matrices $\mathbf{J}_\delta$, $\mathbf{J}_\Phi$ and $\mathbf{J}_\Omega$ indicate which elements of the matrices $\boldsymbol{\delta}_t$, $\mathbf{\Phi}_t$ and $\mathbf{\Omega}_t$ are time varying and the data matrix $\mathbf{X}$ contains the time varying components. The naming convention for these components is summarized in Table 14.2.

14.2.3 Missing Values

The S+FinMetrics/SsfPack state space modeling functions can handle missing values in the vector of response variables $\mathbf{y}_t$ in (14.3). Missing values are not allowed in the state space system matrices $\mathbf{\Phi}_t$, $\mathbf{\Omega}_t$, $\mathbf{\Sigma}$ and $\boldsymbol{\delta}_t$. Missing values are represented by NA in S-PLUS.

In the S+FinMetrics/SsfPack state space functions, the observation vector $\mathbf{y}_t$ with missing values will be be reduced to the vector $\mathbf{y}_t^\dagger$ without missing values and the measurement equation will be adjusted accordingly. For example, the measurement equation $\mathbf{y}_t = \mathbf{c}_t + \mathbf{Z}_t\boldsymbol{\alpha}_t + \mathbf{G}_t\boldsymbol{\varepsilon}_t$ with

$$\mathbf{y}_t = \begin{pmatrix} 5 \\ NA \\ 3 \\ NA \end{pmatrix}, \ \mathbf{c}_t = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}, \ \mathbf{Z}_t = \begin{pmatrix} Z_{1,t} \\ Z_{2,t} \\ Z_{3,t} \\ Z_{4,t} \end{pmatrix}, \ \mathbf{G}_t = \begin{pmatrix} G_{1,t} \\ G_{2,t} \\ G_{3,t} \\ G_{4,t} \end{pmatrix}$$

reduces to the measurement equation $\mathbf{y}_t^\dagger = \mathbf{c}_t^\dagger + \mathbf{Z}_t^\dagger\boldsymbol{\alpha}_t + \mathbf{G}_t^\dagger\boldsymbol{\varepsilon}_t$ with

$$\mathbf{y}_t^\dagger = \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \ \mathbf{c}_t^\dagger = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \ \mathbf{Z}_t^\dagger = \begin{pmatrix} Z_{1,t} \\ Z_{3,t} \end{pmatrix}, \ \mathbf{G}_t^\dagger = \begin{pmatrix} G_{1,t} \\ G_{3,t} \end{pmatrix}$$

The *SsfPack* algorithms in S+FinMetrics automatically replace the observation vector $\mathbf{y}_t$ with $\mathbf{y}_t^\dagger$ when missing values are encountered and the system matrices are adjusted accordingly.

14.2.4 S+FinMetrics/SsfPack Functions for Specifying the State Space Form for Some Common Time Series Models

S+FinMetrics/SsfPack has functions for the creation of the state space representation of some common time series models. These functions and models are described in the following sub-sections.

ARMA Models

Following Harvey (1993, Section 4.4), the ARMA(p, q) model with zero mean[3]

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \xi_t + \theta_1 \xi_{t-1} + \cdots + \theta_q \xi_{t-q} \tag{14.21}$$

may be put in state space form with transition and measurement equations

$$\begin{aligned} \boldsymbol{\alpha}_{t+1} &= \mathbf{T}\boldsymbol{\alpha}_t + \mathbf{H}\xi_t, \ \xi_t \sim N(0, \sigma_\varepsilon^2) \\ y_t &= \mathbf{Z}\boldsymbol{\alpha}_t \end{aligned}$$

[3]Note that the MA coefficients are specified with positive signs, which is the opposite of how the MA coefficients are specified for models estimated by the S-PLUS function arima.mle.

and time invariant system matrices

$$\begin{aligned}
\mathbf{T} &= \begin{pmatrix} \phi_1 & 1 & 0 & \cdots & 0 \\ \phi_2 & 0 & 1 & & 0 \\ \vdots & & & \ddots & \vdots \\ \phi_{m-1} & 0 & 0 & & 1 \\ \phi_m & 0 & 0 & \cdots & \end{pmatrix}, \ \mathbf{H} = \begin{pmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_{m-1} \\ \theta_m \end{pmatrix}, \quad (14.22) \\
\mathbf{Z} &= \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \end{pmatrix}
\end{aligned}$$

where $\mathbf{d}, \mathbf{c}$ and $\mathbf{G}$ of the state space form (14.1)-(14.3) are all zero and $m = \max(p, q+1)$. The state vector $\boldsymbol{\alpha}_t$ has the form

$$\boldsymbol{\alpha}_t = \begin{pmatrix} y_t \\ \phi_2 y_{t-1} + \cdots + \phi_p y_{t-m+1} + \theta_1 \xi_t + \cdots + \theta_{m-1} \xi_{t-m+2} \\ \phi_3 y_{t-1} + \cdots + \phi_p y_{t-m+2} + \theta_2 \xi_t + \cdots + \theta_{m-1} \xi_{t-m+3} \\ \vdots \\ \phi_m y_{t-1} + \theta_{m-1} \xi_t \end{pmatrix} \quad (14.23)$$

In compact state space form (14.7), the model is

$$\begin{aligned}
\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} &= \begin{pmatrix} \mathbf{T} \\ \mathbf{Z} \end{pmatrix} \boldsymbol{\alpha}_t + \begin{pmatrix} \mathbf{H} \\ 0 \end{pmatrix} \xi_t \\
&= \boldsymbol{\Phi} \boldsymbol{\alpha}_t + \mathbf{u}_t
\end{aligned}$$

and

$$\boldsymbol{\Omega} = \begin{pmatrix} \sigma_\xi^2 \mathbf{H}\mathbf{H}' & 0 \\ 0 & 0 \end{pmatrix}$$

If y_t is stationary then $\boldsymbol{\alpha}_t \sim N(0, \mathbf{V})$ is the unconditional distribution of the state vector, and the covariance matrix $\mathbf{V}$ satisfies $\mathbf{V} = \mathbf{T}\mathbf{V}\mathbf{T}' + \boldsymbol{\sigma}_\xi^2 \mathbf{H}\mathbf{H}'$, which can be solved for the elements of $\mathbf{V}$. The initial value parameters are then

$$\boldsymbol{\Sigma} = \begin{pmatrix} \mathbf{V} \\ \mathbf{0}' \end{pmatrix}$$

The `S+FinMetrics/SsfPack` function `GetSsfArma` creates the state space system matrices for any univariate stationary and invertible ARMA model. The arguments expected by `GetSsfArma` are

```
> args(GetSsfArma)
function(ar = NULL, ma = NULL, sigma = 1, model = NULL)
```

where `ar` is the vector of p AR coefficients, `ma` is the vector of q MA coefficients, `sigma` is the innovation standard deviation σ_ξ, and `model` is a list with components giving the AR, MA and innovation standard deviation. If the arguments `ar`, `ma`, and `sigma` are specified, then `model` is ignored. The function `GetSsfArma` returns a list containing the system matrices $\boldsymbol{\Phi}$, $\boldsymbol{\Omega}$ and the initial value parameters $\boldsymbol{\Sigma}$.

Example 94 *AR(1) and ARMA(2,1)*

Consider the AR(1) model

$$y_t = 0.75y_{t-1} + \xi_t,\ \xi_t \sim GWN(0, (0.5)^2)$$

The state space form may be computed using

```
> ssf.ar1 = GetSsfArma(ar=0.75,sigma=.5)
> ssf.ar1
$mPhi:
     [,1]
[1,] 0.75
[2,] 1.00

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    0

$mSigma:
       [,1]
[1,] 0.5714
[2,] 0.0000
```

In the component `mSigma`, the unconditional variance of the initial state α_1 is computed as $var(\alpha_1) = (0.5)^2/(1 - 0.75^2) = 0.5714$.

Next, consider the ARMA(2,1) model

$$y_t = 0.6y_{t-1} + 0.2y_{t-2} + \varepsilon_t - 0.2\varepsilon_{t-1},\ \varepsilon_t \sim GWN(0, 0.9)$$

The state space system matrices may be computed using

```
> arma21.mod = list(ar=c(0.6,0.2),ma=c(-0.2),sigma=sqrt(0.9))
> ssf.arma21 = GetSsfArma(model=arma21.mod)
> ssf.arma21
$mPhi:
     [,1] [,2]
[1,]  0.6    1
[2,]  0.2    0
[3,]  1.0    0

$mOmega:
      [,1]   [,2] [,3]
[1,]  0.90 -0.180    0
[2,] -0.18  0.036    0
[3,]  0.00  0.000    0
```

```
$mSigma:
        [,1]    [,2]
[1,] 1.58571 0.01286
[2,] 0.01286 0.09943
[3,] 0.00000 0.00000
```

The unconditional variance of the initial state vector $\boldsymbol{\alpha}_1 = (\alpha_{11}, \alpha_{12})'$ is in the top 2×2 block of `mSigma` and is

$$var(\boldsymbol{\alpha}_1) = \begin{pmatrix} 1.586 & 0.013 \\ 0.013 & 0.099 \end{pmatrix}.$$

Structural Time Series Models

The basic univariate unobserved components *structural time series model* (STSM) for a time series y_t has the form

$$y_t = \mu_t + \gamma_t + \psi_t + \xi_t \tag{14.24}$$

where μ_t represents the unobserved *trend* component, γ_t represents the unobserved *seasonal* component, ψ_t represents the unobserved *cycle* component, and ξ_t represents the unobserved *irregular* component.

The nonstationary trend component μ_t has the form of a *local linear trend*:

$$\begin{aligned} \mu_{t+1} &= \mu_t + \beta_t + \eta_t,\ \eta_t \sim GWN(0, \sigma_\eta^2) && (14.25) \\ \beta_{t+1} &= \beta_t + \varsigma_t,\ \varsigma_t \sim GWN(0, \sigma_\varsigma^2) && (14.26) \end{aligned}$$

with $\mu_1 \sim N(0, \kappa)$ and $\beta_1 \sim N(0, \kappa)$ where κ is a large number, e.g. $\kappa = 10^6$. If $\sigma_\varsigma^2 = 0$ then μ_t follows a random walk with drift β_1. If both $\sigma_\varsigma^2 = 0$ and $\sigma_\eta^2 = 0$ then μ_t follows a linear deterministic trend.

The stochastic seasonal component γ_t has the form

$$\begin{aligned} S(L)\gamma_t &= \omega_t,\ \omega_t \sim GWN(0, \sigma_\omega^2) && (14.27) \\ S(L) &= 1 + L + \cdots + L^{s-1} \end{aligned}$$

where s gives the number of seasons. When $\sigma_\omega^2 = 0$, the seasonal component becomes fixed.

The stochastic cycle component ψ_t is specified as

$$\begin{aligned} \begin{pmatrix} \psi_{t+1} \\ \psi_{t+1}^* \end{pmatrix} &= \rho \begin{pmatrix} \cos\lambda_c & \sin\lambda_c \\ -\sin\lambda_c & \cos\lambda_c \end{pmatrix} \begin{pmatrix} \psi_t \\ \psi_t^* \end{pmatrix} + \begin{pmatrix} \chi_t \\ \chi_t^* \end{pmatrix}, && (14.28) \\ \begin{pmatrix} \chi_t \\ \chi_t^* \end{pmatrix} &\sim GWN\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma_\psi^2(1-\rho^2)\mathbf{I}_2 \right) \end{aligned}$$

where $\psi_0 \sim N(0, \sigma_\psi^2)$, $\psi_0^* \sim N(0, \sigma_\psi^2)$ and $cov(\psi_0, \psi_0^*) = 0$. The parameter $\rho \in (0, 1]$ is interpreted as a damping factor. The frequency of the cycle

Argument	STSM parameter
irregular	σ_η
level	σ_ξ
slope	σ_ς
seasonalDummy	s
seasonalTrig	σ_ω, s
seasonalHS	σ_ω, s
cycle0	$\sigma_\psi, \lambda_c, \rho$
⋮	⋮
cycle9	$\sigma_\psi, \lambda_c, \rho$

TABLE 14.3. Arguments to the S+FinMetrics function GetSsfStsm

is $\lambda_c = 2\pi/c$ and c is the period. When $\rho = 1$ the cycle reduces to a deterministic sine-cosine wave.

The S+FinMetrics/SsfPack function GetSsfStsm creates the state space system matrices for the univariate structural time series model (14.24). The arguments expected by GetSsfStsm are

```
> args(GetSsfStsm)
function(irregular = 1, level = 0.1, slope = NULL,
   seasonalDummy = NULL, seasonalTrig = NULL, seasonalHS
   = NULL, cycle0 = NULL, cycle1 = NULL, cycle2 = NULL,
   cycle3 = NULL, cycle4 = NULL, cycle5 = NULL, cycle6 =
   NULL, cycle7 = NULL, cycle8 = NULL, cycle9 = NULL)
```

These arguments are explained in Table 14.3.

Example 95 *Local level model*

The state space for the local level model (14.12)-(14.14) may be constructed using

```
> ssf.stsm = GetSsfStsm(irregular=1, level=0.5)
> class(ssf.stsm)
[1] "list"
> ssf.stsm
$mPhi:
     [,1]
[1,]    1
[2,]    1

$mOmega:
     [,1] [,2]
[1,] 0.25    0
[2,] 0.00    1
```

```
$mSigma:
     [,1]
[1,]   -1
[2,]    0
```

The arguments `irregular=1` and `level=0.5` specify $\sigma_\varepsilon = 1$ and $\sigma_\eta = 1$ in (14.13) and (14.14), respectively.

Regression Models

The linear regression model

$$y_t = \mathbf{x}_t'\boldsymbol{\beta} + \xi_t,\ \xi_t \sim GWN(0, \sigma_\xi^2),$$

where $\mathbf{x}_t$ is a $k \times 1$ data matrix and $\boldsymbol{\beta}$ is a $k \times 1$ fixed parameter vector, may be put in the state space

$$\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} = \begin{pmatrix} \mathbf{I}_k \\ \mathbf{x}_t' \end{pmatrix} \boldsymbol{\alpha}_t + \begin{pmatrix} \mathbf{0} \\ \sigma_\xi \varepsilon_t \end{pmatrix} \tag{14.29}$$

The state vector satisfies $\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t = \boldsymbol{\beta}$. The state space system matrices are $\mathbf{T}_t = \mathbf{I}_k$, $\mathbf{Z}_t = \mathbf{x}_t'$, $\mathbf{G}_t = \sigma_\xi$ and $\mathbf{H}_t = 0$. The coefficient vector $\boldsymbol{\beta}$ is fixed and unknown so that the initial conditions are $\boldsymbol{\alpha}_1 \sim N(\mathbf{0}, \kappa \mathbf{I}_k)$ where κ is large. An advantage of analyzing the linear regression model in state space form is that recursive least squares estimates of the regression coefficient vector $\boldsymbol{\beta}$ are readily computed. Another advantage is that it is straightforward to allow some or all of the regression coefficients to be time varying.

The linear regression model with time varying parameters may be introduced by setting $\mathbf{H}_t$ not equal to zero in (14.29). For example, to allow all regressors to evolve as random walks set

$$\mathbf{H}_t = \begin{pmatrix} \sigma_{\beta_1} \\ \vdots \\ \sigma_{\beta_k} \end{pmatrix}$$

so that the state equation becomes

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \mathbf{H}_t \boldsymbol{\eta}_t \tag{14.30}$$

More explicitly, since $\alpha_{i,t+1} = \alpha_{i,t} = \beta_{i,t}$ the state equation (14.30) implies

$$\beta_{i,t+1} = \beta_{i,t} + \sigma_{\beta_i} \cdot \eta_{i,t},\ i = 1, \ldots, k$$

If $\sigma_{\beta_i} = 0$ then β_i is constant.

The `S+FinMetrics/SsfPack` function `GetSsfReg` creates the state space system matrices for the linear regression model. The arguments expected by `GetSsfReg` are

```
> args(GetSsfReg)
function(mX)
```

where `mX` is a rectangular data object which represents the regressors in the model. The function `GetSsfReg` returns a list with components describing the minimal state space representation of the linear regression model.

Example 96 *Time trend regression model*

Consider the time trend regression model

$$y_t = \mu + \delta t + \xi_t,\ \xi_t \sim GWN(0, \sigma_\xi^2)$$

The state space form for a sample of size $n = 10$ is computed using

```
> ssf.reg = GetSsfReg(cbind(1,1:10))
> class(ssf.reg)
[1] "list"
> names(ssf.reg)
[1] "mPhi"   "mOmega" "mSigma" "mJPhi"  "mX"
> ssf.reg
$mPhi:
     [,1] [,2]
[1,]    1    0
[2,]    0    1
[3,]    0    0

$mOmega:
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    1

$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mJPhi:
     [,1] [,2]
[1,]   -1   -1
[2,]   -1   -1
[3,]    1    2

$mX:
      [,1] [,2]
```

```
 [1,]    1    1
 [2,]    1    2
 [3,]    1    3
 [4,]    1    4
 [5,]    1    5
 [6,]    1    6
 [7,]    1    7
 [8,]    1    8
 [9,]    1    9
[10,]    1   10
```

Since the system matrix $\mathbf{Z}_t = \mathbf{x}_t'$, the parameter $\mathbf{\Phi}_t$ is time varying and the index matrix $\mathbf{J}_\Phi$, represented by the component `mJPhi`, determines the association between the time varying data in $\mathbf{Z}_t$ and the data supplied in the component `mX`.

To specify a time trend regression with a time varying slope of the form

$$\delta_{t+1} = \delta_t + \zeta_t, \ \zeta_t \sim GWN(0, \sigma_\zeta^2) \tag{14.31}$$

one needs to specify a non-zero value for σ_ζ^2 in $\mathbf{\Omega}_t$. For example, if $\sigma_\zeta^2 = 0.5$ then

```
> ssf.reg.tvp = ssf.reg
> ssf.reg.tvp$mOmega[2,2] = 0.5
> ssf.reg.tvp$mOmega
     [,1] [,2] [,3]
[1,]    0  0.0    0
[2,]    0  0.5    0
[3,]    0  0.0    1
```

modifies the state space form for the time trend regression to allow a time varying slope of the form (14.31).

Regression Models with ARMA Errors

The ARMA(p, q) models created with `GetSsfArma` do not have deterministic terms (e.g., constant, trend, seasonal dummies) or exogenous regressors and are therefore limited. The general ARMA(p, q) model with exogenous regressors has the form

$$y_t = \mathbf{x}_t'\boldsymbol{\beta} + \xi_t$$

where ξ_t follows an ARMA(p, q) process of the form (14.21). Let $\boldsymbol{\alpha}_t$ be defined as in (14.23) and let

$$\boldsymbol{\alpha}_t^* = \begin{pmatrix} \boldsymbol{\alpha}_t \\ \boldsymbol{\beta}_t \end{pmatrix} \tag{14.32}$$

where $\boldsymbol{\beta}_t = \boldsymbol{\beta}$. Writing the state equation implied by (14.32) as $\boldsymbol{\alpha}^*_{t+1} = \mathbf{T}^*\boldsymbol{\alpha}^*_t + \mathbf{H}^*\boldsymbol{\eta}_t$ and let

$$\begin{aligned} \mathbf{T}^* &= \begin{bmatrix} \mathbf{T} & 0 \\ 0 & \mathbf{I}_k \end{bmatrix}, \ \mathbf{H}^* = \begin{bmatrix} \mathbf{H} \\ \mathbf{0} \end{bmatrix}, \\ \mathbf{Z}^*_t &= \begin{pmatrix} 1 & 0 & \dots & 0 & \mathbf{x}'_t \end{pmatrix} \end{aligned}$$

where $\mathbf{T}$ and $\mathbf{H}$ are defined in (14.23). Then the state space form of the regression model with ARMA errors is

$$\begin{pmatrix} \boldsymbol{\alpha}_{t+1} \\ y_t \end{pmatrix} = \begin{pmatrix} \mathbf{T}^* \\ \mathbf{Z}^*_t \end{pmatrix} \boldsymbol{\alpha}^*_t + \begin{pmatrix} \mathbf{H}^*\boldsymbol{\eta}_t \\ 0 \end{pmatrix}$$

The `S+FinMetrics/SsfPack` function `GetSsfRegArma` creates the state space system matrices for the linear regression model with ARMA errors. The arguments expected by `GetSsfRegArma` are

```
> args(GetSsfRegArma)
function(mX, ar = NULL, ma = NULL, sigma = 1, model = NULL)
```

where `mX` is a rectangular data object which represents the regressors in the model, and the remaining arguments are the same as those for `GetSsfArma`. The function `GetSsfRegArma` returns a list with components describing the minimal state space representation of the linear regression model.

Example 97 *Time trend regression with AR(2) errors*

The state space form of the time trend regression with AR(2) errors

$$\begin{aligned} y_t &= \mu + \delta t + \xi_t, \\ \xi_t &= \phi_1\xi_{t-1} + \phi_2\xi_{t-2} + \nu_t, \ \nu_t \sim GWN(0, \sigma^2_\xi) \end{aligned}$$

may be computed using

```
> ssf.reg.ar2 = GetSsfRegArma(cbind(1,1:10),
+ ar=c(1.25,-0.5))
> ssf.reg.ar2
$mPhi:
      [,1] [,2] [,3] [,4]
[1,]  1.25    1    0    0
[2,] -0.50    0    0    0
[3,]  0.00    0    1    0
[4,]  0.00    0    0    1
[5,]  1.00    0    0    0

$mOmega:
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
```

```
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0

$mSigma:
       [,1]   [,2] [,3] [,4]
[1,]  4.364 -1.818    0    0
[2,] -1.818  1.091    0    0
[3,]  0.000  0.000   -1    0
[4,]  0.000  0.000    0   -1
[5,]  0.000  0.000    0    0

$mJPhi:
     [,1] [,2] [,3] [,4]
[1,]   -1   -1   -1   -1
[2,]   -1   -1   -1   -1
[3,]   -1   -1   -1   -1
[4,]   -1   -1   -1   -1
[5,]   -1   -1    1    2

$mX:
      [,1] [,2]
 [1,]    1    1
 [2,]    1    2
...
[10,]    1   10
```

Nonparametric Cubic Spline Model

Suppose the continuous time process $y(t)$ is observed at discrete time points $t_1, \ldots, t_n$. Define $\delta_i = t_i - t_{i-1} \geq 0$ as the time duration between observations. The goal of the *nonparametric cubic spline model* is to estimate a smooth signal $\mu(t)$ from $y(t)$ using

$$y(t) = \mu(t) + \varepsilon(t)$$

where $\varepsilon(t)$ is a stationary error. The signal $\mu(t)$ is extracted by minimizing

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

where the second term is a penalty term that forces $\mu(t)$ to be "smooth"[4], and q may be interpreted as a signal-to-noise ratio. The resulting function $\mu(t)$ may be expressed as the structural time series model

$$\begin{aligned} \mu(t_{i+1}) &= \mu(t_i) + \delta_i \beta(t_i) + \eta(t_i) \\ \beta(t_{i+1}) &= \beta(t_i) + \varsigma(t_i) \end{aligned} \tag{14.33}$$

where

$$\begin{pmatrix} \eta(t_i) \\ \varsigma(t_i) \end{pmatrix} \tilde{} N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma_\varsigma^2 \delta_i \begin{pmatrix} \frac{1}{3}\delta_i^2 & \frac{1}{2}\delta_i \\ \frac{1}{2}\delta_i & 1 \end{pmatrix} \right]$$

Combining (14.33) with the measurement equation

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

where $\varepsilon(t_i) \sim N(0, \sigma_\varepsilon^2)$ and is independent of $\eta(t_i)$ and $\varsigma(t_i)$, gives the state space form for the nonparametric cubic spline model. The state space system matrices are

$$\Phi_t = \begin{pmatrix} 1 & \delta_i \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \Omega_t = \begin{pmatrix} \frac{q\delta_t^2}{3} & \frac{q\delta_t^2}{2} & 0 \\ \frac{q\delta_t^2}{2} & q\delta_t & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

When the observations are equally spaced, δ_i is constant and the above system matrices are time invariant.

The `S+FinMetrics/SsfPack` function `GetSsfSpline` creates the state space system matrices for the nonparametric cubic spline model. The arguments expected by `GetSsfSpline` are

```
> args(GetSsfSpline)
function(snr = 1, delta = 0)
```

where `snr` is the signal-to-noise ratio q, and `delta` is a numeric vector containing the time durations δ_i $(i = 1, \ldots, n)$. If `delta=0` then δ_i is assumed to be equal to unity and the time invariant form of the model is created. The function `GetSsfSpline` returns a list with components describing the minimal state space representation of the nonparametric cubic spline model.

Example 98 *State space form of non-parametric cubic spline model*

The default non-parametric cubic spline model with $\delta_t = 1$ is created using

[4]This process can be interpreted as an interpolation technique and is similar to the technique used in the `S+FinMetrics` functions `interpNA` and `hpfilter`. See also the smoothing spline method described in chapter sixteen.

```
> GetSsfSpline()
$mPhi:
     [,1] [,2]
[1,]    1    1
[2,]    0    1
[3,]    1    0

$mOmega:
       [,1] [,2] [,3]
[1,] 0.3333  0.5    0
[2,] 0.5000  1.0    0
[3,] 0.0000  0.0    1

$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mJPhi:
NULL

$mJOmega:
NULL

$mX:
NULL
```

For unequally spaced observations

```
> t.vals = c(2,3,5,9,12,17,20,23,25)
> delta.t = diff(t.vals)
```

and $q = 0.2$, the state space form is

```
> GetSsfSpline(snr=0.2,delta=delta.t)
$mPhi:
     [,1] [,2]
[1,]    1    1
[2,]    0    1
[3,]    1    0

$mOmega:
        [,1] [,2] [,3]
[1,] 0.06667  0.1    0
[2,] 0.10000  0.2    0
[3,] 0.00000  0.0    1
```

```
$mSigma:
     [,1] [,2]
[1,]   -1    0
[2,]    0   -1
[3,]    0    0

$mJPhi:
     [,1] [,2]
[1,]   -1    1
[2,]   -1   -1
[3,]   -1   -1

$mJOmega:
     [,1] [,2] [,3]
[1,]    4    3   -1
[2,]    3    2   -1
[3,]   -1   -1   -1

$mX:
        [,1]   [,2]  [,3] [,4]  [,5] [,6] [,7]   [,8]
[1,] 1.00000 2.0000 4.000  3.0 5.000  3.0  3.0 2.0000
[2,] 0.20000 0.4000 0.800  0.6 1.000  0.6  0.6 0.4000
[3,] 0.10000 0.4000 1.600  0.9 2.500  0.9  0.9 0.4000
[4,] 0.06667 0.5333 4.267  1.8 8.333  1.8  1.8 0.5333
```

14.2.5 Simulating Observations from the State Space Model

Once a state space model has been specified, it is often interesting to draw simulated values from the model. The `S+FinMetrics/SsfPack` function `SsfSim` may be used for such a purpose. The arguments expected from `SsfSim` are

```
> args(SsfSim)
function(ssf, n = 100, mRan = NULL, a1 = NULL)
```

where `ssf` represents either a list with components giving a minimal state space form or a valid "`ssf`" object, `n` is the number of simulated observations, `mRan` is user-specified matrix of disturbances, and `a1` is the initial state vector. The use of `SsfSim` is illustrated with the following examples.

Example 99 *Simulating observations from the local level model*

To generate 250 observations on the state variable α_{t+1} and observations y_t in the local level model (14.12) - (14.14) use

```
> set.seed(112)
```

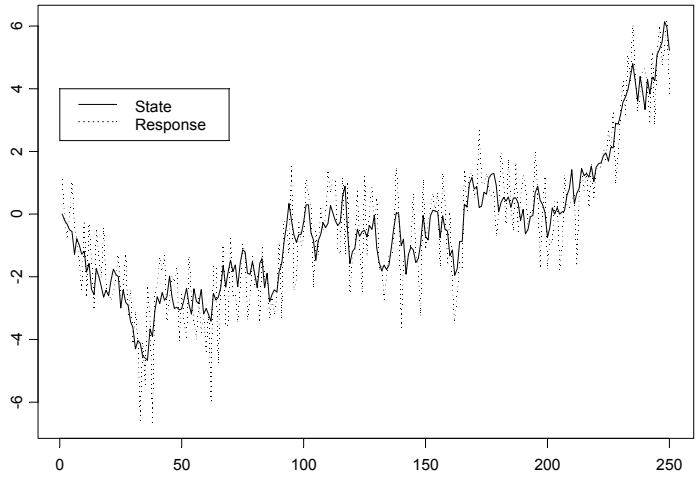

FIGURE 14.1. Simulated values from local level model created using the `S+FinMetrics` function `SsfSim`.

```
> ll.sim = SsfSim(ssf.ll.list,n=250)
> class(ll.sim)
[1] "matrix"
> colIds(ll.sim)
[1] "state"    "response"
```

The function `SsfSim` returns a matrix containing the simulated state variables α_{t+1} and observations y_t. These values are illustrated in Figure 14.1 created using

```
> tsplot(ll.sim)
> legend(0,4,legend=c("State","Response"),lty=1:2)
```

Example 100 *Simulating observations from CAPM with time varying parameters*

When simulating observations from a state space form with a data matrix component `mX` using the function `SsfSim`, the number of simulated values must match the number of rows of `mX`. The state space form for the CAPM with time varying parameters created earlier uses a data matrix `mX` with $n = 131$ observations

```
> nrow(ssf.tvp.capm$mX)
[1] 131
```

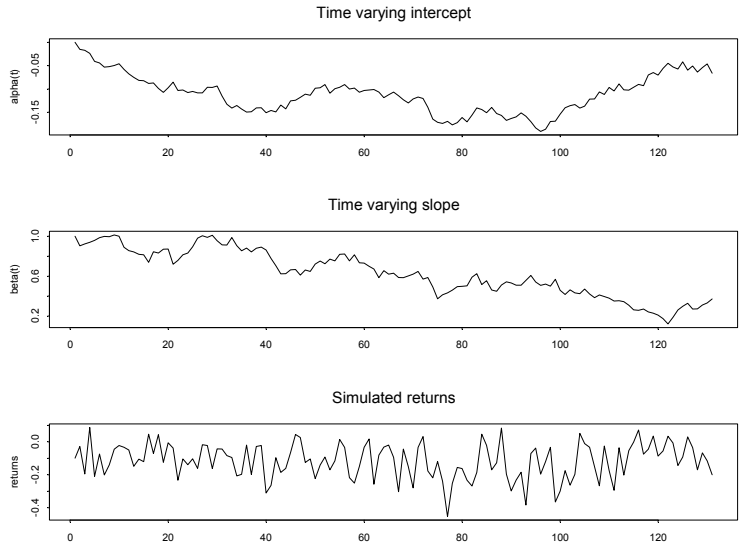

FIGURE 14.2. Simulated state and response values from the CAPM with time varying parameters state space form `ssf.tvp`.

The state variables are the time varying intercept α_{t+1} and the time varying β_{t+1}. Natural initial values for these parameters are $\alpha_1 = 0$ and $\beta_1 = 1$. Using these initial values, $n = 131$ observations are generated from the CAPM with time varying parameters using

```
> set.seed(444)
> tvp.sim = SsfSim(ssf.tvp.capm,n=nrow(X.mat),a1=c(0,1))
> colIds(tvp.sim)
[1] "state.1"  "state.2"  "response"
```

The simulated state and response variables are illustrated in Figure 14.2 created using

```
> par(mfrow=c(3,1))
> tsplot(tvp.sim[,"state.1"],main="Time varying intercept",
+ ylab="alpha(t)")
> tsplot(tvp.sim[,"state.2"],main="Time varying slope",
+ ylab="beta(t)")
> tsplot(tvp.sim[,"response"],main="Simulated returns",
+ ylab="returns")
```

14.3 Algorithms

14.3.1 Kalman Filter

The Kalman filter is a recursive algorithm for the evaluation of moments of the normally distributed state vector $\boldsymbol{\alpha}_{t+1}$ conditional on the observed data $\mathbf{Y}_t = (y_1, \ldots, y_t)$. To describe the algorithm, let $\mathbf{a}_t = E[\boldsymbol{\alpha}_t|\mathbf{Y}_{t-1}]$ denote the conditional mean of $\boldsymbol{\alpha}_t$ based on information available at time $t-1$ and let $\mathbf{P}_t = var(\boldsymbol{\alpha}_t|\mathbf{Y}_{t-1})$ denote the conditional variance of α_t.

The *filtering* or *updating* equations of the Kalman filter compute $\mathbf{a}_{t|t} = E[\boldsymbol{\alpha}_t|\mathbf{Y}_t]$ and $\mathbf{P}_{t|t} = var(\boldsymbol{\alpha}_t|\mathbf{Y}_t)$ using

$$\begin{aligned} \mathbf{a}_{t|t} &= \mathbf{a}_t + \mathbf{K}_t\mathbf{v}_t & (14.34)\\ \mathbf{P}_{t|t} &= \mathbf{P}_t - \mathbf{P_t}\mathbf{Z}_\mathbf{t}'\mathbf{K}_t' & (14.35) \end{aligned}$$

where

$$\begin{aligned} \mathbf{v}_t &= \mathbf{y}_t - \mathbf{c}_t - \mathbf{Z}_t\mathbf{a}_t & (14.36)\\ \mathbf{F}_t &= \mathbf{Z}_t\mathbf{P}_t\mathbf{Z}_t' + \mathbf{G}_t\mathbf{G}_t' & (14.37)\\ \mathbf{K}_t &= \mathbf{P}_t\mathbf{Z}_t'\mathbf{F}_t^{-1} & (14.38) \end{aligned}$$

The variable $\mathbf{v}_t$ is the *measurement equation innovation* or prediction error, $\mathbf{F}_t = var(\mathbf{v}_t)$ and $\mathbf{K}_t$ is the *Kalman gain* matrix.

The *prediction* equations of the Kalman filter compute $\mathbf{a}_{t+1}$ and $\mathbf{P}_{t+1}$ using

$$\begin{aligned} \mathbf{a}_{t+1} &= \mathbf{T}_t\mathbf{a}_{t|t} & (14.39)\\ \mathbf{P}_{t+1} &= \mathbf{T}_t\mathbf{P}_{t|t}\mathbf{T}_t' + \mathbf{H}_t\mathbf{H}_t' & (14.40) \end{aligned}$$

In the Kalman filter recursions, if there are missing values in $\mathbf{y}_t$ then $\mathbf{v}_t = \mathbf{0}$, $\mathbf{F}_t^{-1} = \mathbf{0}$ and $\mathbf{K}_t = \mathbf{0}$. This allows out-of-sample forecasts of $\boldsymbol{\alpha}_t$ and $\mathbf{y}_t$ to be computed from the updating and prediction equations.

14.3.2 Kalman Smoother

The Kalman filtering algorithm is a forward recursion which computes one-step ahead estimates $\mathbf{a}_{t+1}$ and $\mathbf{P}_{t+1}$ based on $\mathbf{Y}_t$ for $t = 1, \ldots, n$. The *Kalman smoothing* algorithm is a backward recursion which computes the mean and variance of specific conditional distributions based on the full data set $\mathbf{Y}_n = (y_1, \ldots, y_n)$. The *smoothing equations* are

$$\begin{aligned} &\mathbf{r}_t^* = \mathbf{T}_t'\mathbf{r}_t,\ \mathbf{N}_t^* = \mathbf{T}_t\mathbf{N}_t\mathbf{T}_t',\ \mathbf{K}_t^* = \mathbf{N}_t^*\mathbf{K}_t & (14.41)\\ &\mathbf{e}_t = \mathbf{F}_t^{-1}\mathbf{v}_t - \mathbf{K}_t'\mathbf{r}_t^*,\ \mathbf{D}_t = \mathbf{F}_t^{-1} + \mathbf{K}_t\mathbf{K}_t^{*\prime} \end{aligned}$$

and the *backwards updating equations* are

$$\mathbf{r}_{t-1} = \mathbf{Z}_t'\mathbf{e}_t + \mathbf{r}_t^*,\ \mathbf{N}_{t-1} = \mathbf{Z}_t'\mathbf{D}_t\mathbf{Z}_t - < \mathbf{K}_t^*\mathbf{Z}_t > + \mathbf{N}_t^* \qquad (14.42)$$

for $t = n, \ldots, 1$ with initializations $\mathbf{r}_n = 0$ and $\mathbf{N}_n = 0$. For any square matrix $\mathbf{A}$, the operator $< \mathbf{A} >= \mathbf{A} + \mathbf{A}'$. The values $\mathbf{r}_t$ are called *state smoothing residuals* and the values $\mathbf{e}_t$ are called *response smoothing residuals*. The recursions (14.41) and (14.42) are somewhat non-standard. Durbin and Koopman (2001) show how they may be re-expressed in more standard form.

14.3.3 Smoothed State and Response Estimates

The smoothed estimates of the state vector $\boldsymbol{\alpha}_t$ and its variance matrix are denoted $\hat{\boldsymbol{\alpha}}_t = E[\boldsymbol{\alpha}_t|\mathbf{Y}_n]$ and $var(\hat{\boldsymbol{\alpha}}_t|\mathbf{Y}_n)$, respectively. The smoothed estimate $\hat{\boldsymbol{\alpha}}_t$ is the optimal estimate of $\boldsymbol{\alpha}_t$ using all available information $\mathbf{Y}_n$, whereas the filtered estimate $\hat{\mathbf{a}}_{t|t}$ is the optimal estimate only using information available at time $t, \mathbf{Y}_t$. The computation of $\hat{\boldsymbol{\alpha}}_t$ and its variance from the Kalman smoother algorithm is described in Durbin and Koopman (2001).

The smoothed estimate of the response $\mathbf{y}_t$ and its variance are computed using

$$\begin{aligned} \hat{\mathbf{y}}_t &= \hat{\boldsymbol{\theta}}_t = E[\boldsymbol{\theta}_t|\mathbf{Y}_n] = \mathbf{c}_t + \mathbf{Z}_t\hat{\boldsymbol{\alpha}}_t \\ var(\hat{\mathbf{y}}_t|\mathbf{Y}_n) &= \mathbf{Z}_t var(\hat{\boldsymbol{\alpha}}_t|\mathbf{Y}_n)\mathbf{Z}_t' \end{aligned}$$

14.3.4 Smoothed Disturbance Estimates

The smoothed disturbance estimates are the estimates of the measurement equations innovations $\boldsymbol{\varepsilon}_t$ and transition equation innovations $\boldsymbol{\eta}_t$ based on all available information $\mathbf{Y}_n$, and are denoted $\hat{\boldsymbol{\varepsilon}}_t = E[\boldsymbol{\varepsilon}_t|\mathbf{Y}_n]$ and $\hat{\boldsymbol{\eta}}_t = E[\boldsymbol{\eta}_t|\mathbf{Y}_n]$, respectively. The computation of $\hat{\boldsymbol{\varepsilon}}_t$ and $\hat{\boldsymbol{\eta}}_t$ from the Kalman smoother algorithm is described in Durbin and Koopman (2001). These smoothed disturbance estimates are useful for parameter estimation by maximum likelihood and for diagnostic checking. See chapter seven in Durbin and Koopman (2001) for details.

14.3.5 Forecasting

The Kalman filter prediction equations (14.39) - (14.40) produces one-step ahead predictions of the state vector, $\mathbf{a}_{t+1} = E[\boldsymbol{\alpha}_{t+1}|\mathbf{Y}_t]$, along with prediction variance matrices $\mathbf{P}_{t+1}$. Out of sample predictions, together with associated mean square errors, can be computed from the Kalman filter prediction equations by extending the data set $\mathbf{y}_1, \ldots, \mathbf{y}_n$ with a set of missing values. When y_τ is missing, the Kalman filter reduces to the prediction step described above. As a result, a sequence of m missing values at the end of the sample will produce a set of h-step ahead forecasts for $h = 1, \ldots, m$.

14.3.6 S+FinMetrics/SsfPack Implementation of State Space Modeling Algorithms

The S+FinMetrics/SsfPack function KalmanFil implements the Kalman filter forward recursions in a computationally efficient way, see Koopman, Shephard and Doornik (2001). It produces an object of class "KalmanFil" for which there are print and plot methods. The S+FinMetrics/SsfPack function KalmanSmo computes the Kalman smoother backwards recursions, and produces an object of class "KalmanSmo" for which there are print and plot methods. The functions KalmanFil and KalmanSmo are primarily used by other S+FinMetrics/SsfPack state space functions that require the output from the Kalman filter and Kalman smoother.

Filtered and smoothed estimates of $\boldsymbol{\alpha}_t$ and $\mathbf{y}_t$, with estimated variances, as well as smoothed estimates of $\boldsymbol{\varepsilon}_t$ and $\boldsymbol{\eta}_t$, with estimated variances, are computed using the S+FinMetrics/SsfPack function SsfMomentEst. The result of SsfMomentEst is an object of class "SsfMomentEst" for which there is only a plot method. The function SsfMomentEst may also be used to compute out-of-sample forecasts and forecast variances of $\boldsymbol{\alpha}_t$ and y_t.

The use of the S+FinMetrics/SsfPack functions for implementing the state space algorithms are illustrated with the following examples.

Example 101 *State space algorithms applied to local level model*

Consider the simulated data for the local level model (14.12) - (14.14) in the object ll.sim computed earlier. The response variable y_t is extracted using

```
> y.ll = ll.sim[,"response"]
```

Kalman Filter

The Kalman filter recursions for the simulated data from the local level model are obtained using the S+FinMetrics/SsfPack function KalmanFil with the optional argument task="STFIL" (which stands for state filtering)

```
> KalmanFil.ll = KalmanFil(y.ll,ssf.ll,task="STFIL")
> class(KalmanFil.ll)
[1] "KalmanFil"
```

The function KalmanFil takes as input a vector of response data and either a list describing the minimal state space form or a valid "ssf" object. The result of KalmanFil is an object of class "KalmanFil" with components

```
> names(KalmanFil.ll)
 [1] "mOut"         "innov"        "std.innov"
 [4] "mGain"        "loglike"      "loglike.conc"
 [7] "dVar"         "mEst"         "mOffP"
[10] "task"         "err"          "call"
```

A complete explanation of the components of a "KalmanFil" object is given in the online help for KalmanFil. These components are mainly used by other S+FinMetrics/SsfPack functions and are only briefly discussed here. The component mOut contains the basic Kalman filter output.

```
> KalmanFil.ll$mOut
numeric matrix: 250 rows, 3 columns.
          [,1]   [,2]   [,3]
 [1,]  0.00000 1.0000 0.0000
 [2,] -1.28697 0.5556 0.4444
...
[250,] -1.6371 0.3904 0.6096
```

The first column of mOut contains the prediction errors v_t, the second column contains the Kalman gains, K_t, and the last column contains the inverses of the prediction error variances, F_t^{-1}. Since task="STFIL" the filtered estimates $a_{t|t}$ and $y_{t|t} = Z_t a_{t|t}$ are in the component mEst

```
> KalmanFil.ll$mEst
numeric matrix: 250 rows, 4 columns.
          [,1]     [,2]   [,3]   [,4]
 [1,]  1.10889  1.10889 1.0000 1.0000
 [2,]  0.39390  0.39390 0.5556 0.5556
...
[250,] 4.839 4.839 0.3904 0.3904
```

The plot method allows for a graphical analysis of the Kalman filter output

```
> plot(KalmanFil.ll)

Make a plot selection (or 0 to exit):

1: plot: all
2: plot: innovations
3: plot: standardized innovations
4: plot: innovation histogram
5: plot: normal QQ-plot of innovations
6: plot: innovation ACF
Selection:
```

The standardized innovations v_t/F_t are illustrated in Figure 14.3.

Kalman Smoother

The Kalman smoother backwards recursions for the simulated data from the local level model are obtained using the S+FinMetrics/SsfPack function KalmanSmo

```
> KalmanSmo.ll = KalmanSmo(KalmanFil.ll,ssf.ll)
```

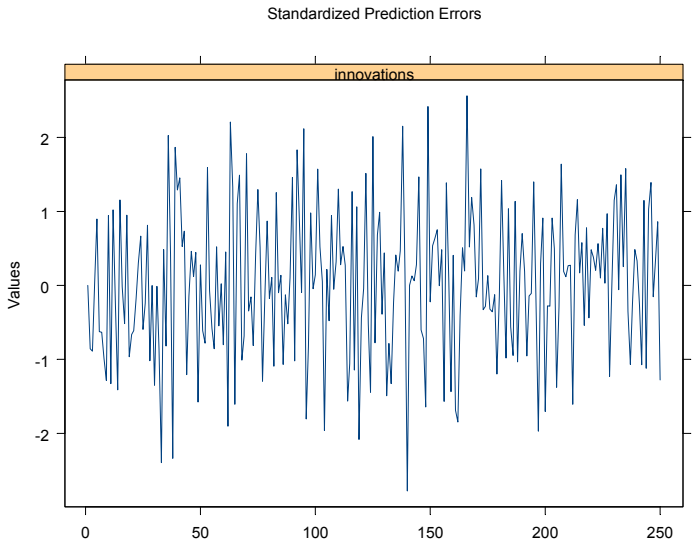

FIGURE 14.3. Standardized innovations from Kalman filter applied to simulated data from local level model.

```
> class(KalmanSmo.ll)
[1] "KalmanSmo"
```

The function `KalmanSmo` takes as input an object of class "`KalmanFil`" and an associated list variable containing the state space form used to produce the "`KalmanFil`" object. The result of `KalmanSmo` is an object of class "`KalmanSmo`" with components

```
> names(KalmanSmo.ll)
[1] "state.residuals"    "response.residuals"
[3] "state.variance"     "response.variance"
[5] "aux.residuals"      "scores"
[7] "call"
```

The component `state.residuals` contains the smoothing residuals from the state equation, `response.residuals` contains the smoothing residuals from the measurement equation. The corresponding variances of these residuals are in the components `state.variance` and `response.variance`. A multi-panel timeplot of the standardized residuals in the component `aux.residuals`, illustrated in Figure 14.4, is created with the `plot` method

```
> plot(KalmanSmo.ll,layout=c(1,2))
```

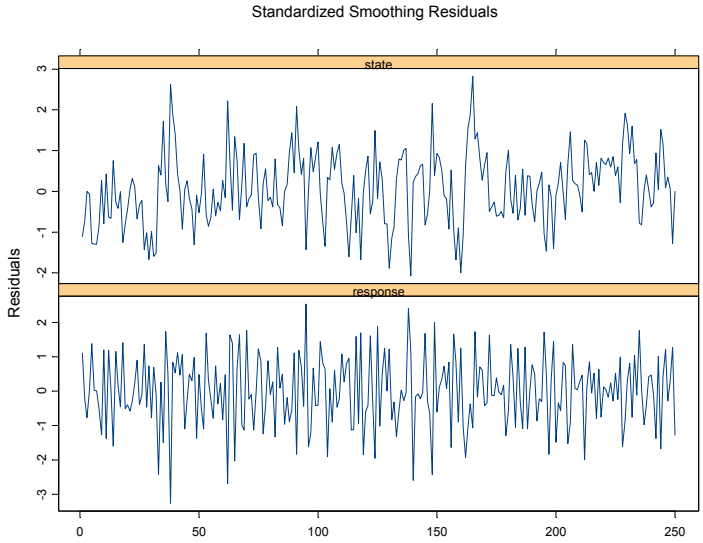

FIGURE 14.4. Standardized smoothing residuals from Kalman smoother recursions computed from simulated data from local level model.

Filtered and Smoothed Moment Estimates

Filtered and smoothed estimates of $\boldsymbol{\alpha}_t$ and y_t with corresponding estimates of variances may be computed using the `S+FinMetrics/SsfPack` function `SsfMomentEst`. To compute filtered estimates, call `SsfMomentEst` with the argument `task="STFIL"` (which stands for state filtering)

```
> FilteredEst.ll = SsfMomentEst(y.ll,ssf.ll,task="STFIL")
> class(FilteredEst.ll)
[1] "SsfMomentEst"
> names(FilteredEst.ll)
[1] "state.moment"      "state.variance"
[3] "response.moment"   "response.variance"
[5] "task"
```

The function `SsfMomentEst` takes as input a vector of response data and either a list describing the minimal state space form or a valid "`ssf`" object. The result of `SsfMomentEst` is an object of class "`SsfMomentEst`" for which there is only a `plot` method. The filtered estimates $a_{t|t}$ and $y_{t|t} = c_t + Z_t a_{t|t}$ are in the components `state.moment` and `response.moment`, respectively, and the corresponding filtered variance estimates are in the components `state.variance` and `response.variance`. From the measurement equation (14.13) in the local level model, $a_{t|t} = y_{t|t}$

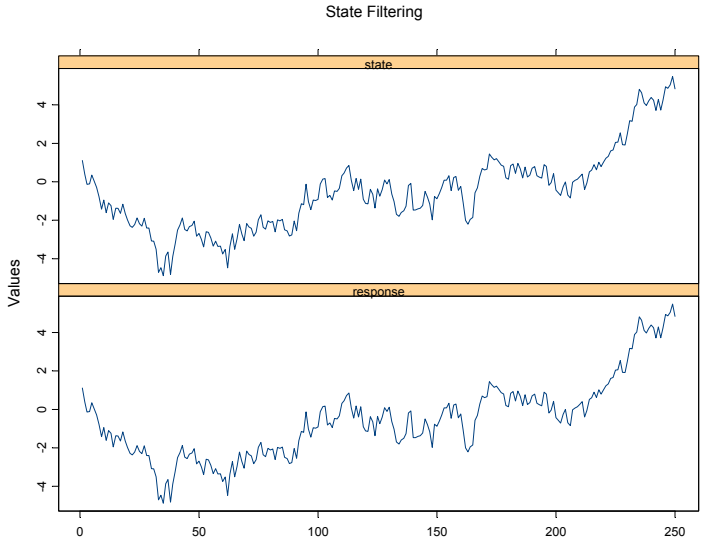

FIGURE 14.5. Filtered estimates of α_t and y_t computed from simulated data from local level model.

```
> FilteredEst.ll$state.moment[1:5]
[1]  1.1089  0.3939 -0.1389 -0.1141  0.3461
> FilteredEst.ll$response.moment[1:5]
[1]  1.1089  0.3939 -0.1389 -0.1141  0.3461
```

The `plot` method creates a multi-panel timeplot, illustrated in Figure 14.5, of the estimates of α_t and y_t

```
> plot(FilteredEst.ll,layout=c(1,2))
```

A plot of the filtered state estimates with 95% confidence intervals may be created using

```
> upper.state = FilteredEst.ll$state.moment +
+ 2*sqrt(FilteredEst.ll$state.variance)
> lower.state = FilteredEst.ll$state.moment -
+ 2*sqrt(FilteredEst.ll$state.variance)
> tsplot(FilteredEst.ll$state.moment,upper.state,lower.state,
+ lty=c(1,2,2),ylab="filtered state")
```

and is shown in Figure 14.6.

The smoothed estimates $\hat{\alpha}_t$ and $\hat{y}_t$ along with estimated variances may be computed using `SsfMomentEst` with `task="STSMO"` (state smoothing)

```
> SmoothedEst.ll = SsfMomentEst(y.ll,ssf.ll.list,task="STSMO")
```

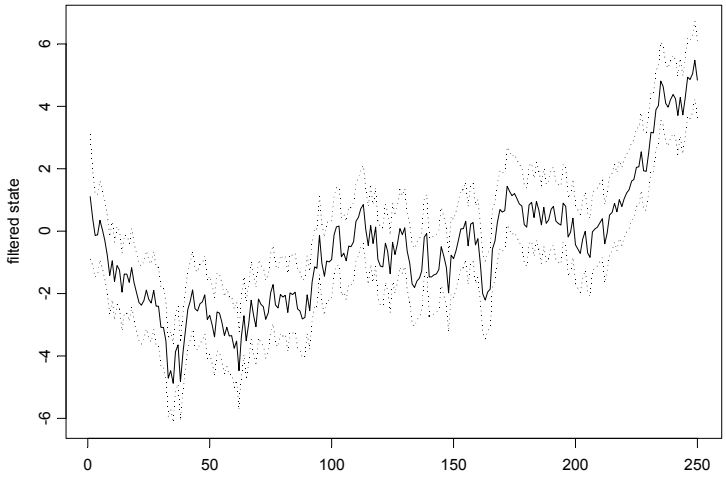

FIGURE 14.6. Filtered estimates of α_t with 95% confidence intervals computed from simulated values from local level model.

In the local level model, $\hat{\alpha}_t = \hat{y}_t$

```
> SmoothedEst.ll$state.moment[1:5]
[1]  0.24281  0.02629 -0.13914 -0.13925 -0.15455
> SmoothedEst.ll$response.moment[1:5]
[1]  0.24281  0.02629 -0.13914 -0.13925 -0.15455
```

The smoothed state estimates with 95% confidence bands are illustrated in Figure 14.7. Compared to the filtered state estimates, the smoothed estimates are "smoother" and the confidence bands are slightly smaller.

Smoothed estimates of α_t and y_t without estimated variances may be obtained using the `S+FinMetrics/SsfPack` function `SsfCondDens` with the argument `task="STSMO"` (which stands for state smoothing)

```
> smoothedEst.ll = SsfCondDens(y.ll,ssf.ll.list,task="STSMO")
> class(smoothedEst.ll)
[1] "SsfCondDens"
> names(smoothedEst.ll)
[1] "state"    "response" "task"
```

The object `smoothedEst.ll` is of class "`SsfCondDens`" with components `state`, giving the smoothed state estimates $\hat{\alpha}_t$, `response`, which gives the smoothed response estimates $\hat{y}_t$, and `task`, naming the task performed. The

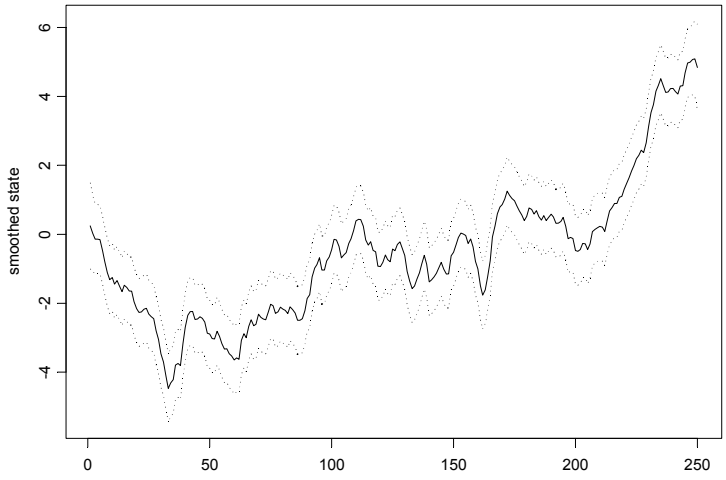

FIGURE 14.7. Smoothed estimates of α_t with 95% confidence intervals computed from simulated values from local level model.

smoothed estimates $\hat{y}_t$ and $\hat{\alpha}_t$ may be visualized using the `plot` method for "`SsfCondDens`" objects

```
> plot(smoothedEst.ll)
```

The resulting plot has the same format at the plot shown in Figure 14.5.

Smoothed Disturbance Estimates

The smoothed disturbance estimates $\hat{\varepsilon}_t$ and $\hat{\eta}_t$ may be computed using `SsfMomentEst` with the optional argument `task="DSSMO"` (which stands for disturbance smoothing)

```
> disturbEst.ll = SsfMomentEst(y.ll,ssf.ll,task="DSSMO")
> names(disturbEst.ll)
[1] "state.moment"      "state.variance"
[3] "response.moment"   "response.variance"
[5] "task"
```

The estimates $\hat{\eta}_t$ are in the component `state.moment`, and the estimates $\hat{\varepsilon}_t$ are in the component `response.moment`. These estimates may be visualized using the `plot` method.

Koopman, Shephard and Doornik (1999) point out that, in the local level model, the standardized smoothed disturbances

$$\frac{\hat{\eta}_t}{\sqrt{\widehat{var}(\hat{\eta}_t)}}, \quad \frac{\hat{\varepsilon}_t}{\sqrt{\widehat{var}(\hat{\varepsilon}_t)}} \tag{14.43}$$

may be interpreted as t-statistics for impulse intervention variables in the transition and measurement equations, respectively. Consequently, large values of (14.43) indicate outliers and/or structural breaks in the local level model.

Forecasting

To produce out-of-sample h-step ahead forecasts $y_{t+h|t}$ for $h = 1, \ldots, 5$ a sequence of 5 missing values is appended to the end of the response vector `y.ll`

```
> y.ll.new = c(y.ll,rep(NA,5))
```

The forecast values and mean squared errors are computed using the function `SsfMomentEst` with the argument `task="STPRED"`

```
> PredictedEst.ll = SsfMomentEst(y.ll.new,ssf.ll,task="STPRED")
> y.ll.fcst = PredictedEst.ll$response.moment
> fcst.var = PredictedEst.ll$response.variance
```

The actual values, forecasts and 95% confidence bands are illustrated in Figure 14.8 created by

```
> upper = y.ll.fcst + 2*sqrt(fcst.var)
> lower = y.ll.fcst - 2*sqrt(fcst.var)
> upper[1:250] = lower[1:250] = NA
> tsplot(y.ll.new[240:255],y.ll.fcst[240:255],
+ upper[240:255],lower[240:255],lty=c(1,2,2,2))
```

14.4 Estimation of State Space Models

14.4.1 Prediction Error Decomposition of Log-Likelihood

The *prediction error decomposition* of the log-likelihood function for the unknown parameters $\boldsymbol{\varphi}$ of a state space model may be conveniently computed using the output of the Kalman filter

$$\begin{aligned} \ln L(\boldsymbol{\varphi}|Y_n) &= \sum_{t=1}^{n} \ln f(\mathbf{y}_t|\mathbf{Y}_{t-1};\boldsymbol{\varphi}) \\ &= -\frac{nN}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{n}\left(\ln|\mathbf{F}_t| + \mathbf{v}_t'\mathbf{F}_t^{-1}\mathbf{v}_t\right) \end{aligned} \tag{14.44}$$

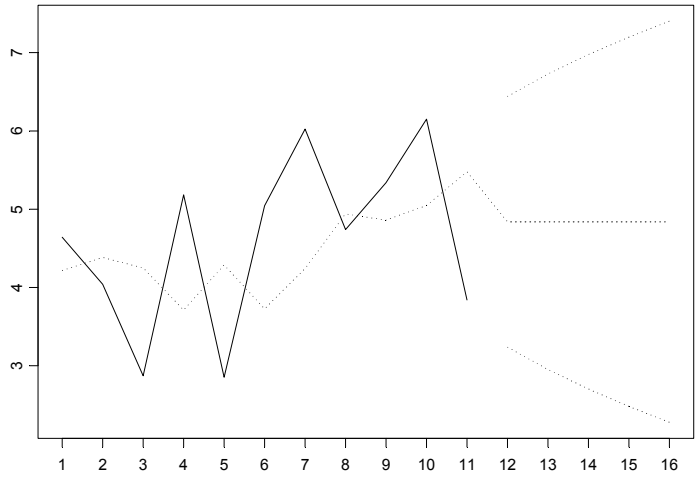

FIGURE 14.8. Actual values, h-step forecasts and 95% confidence intervals for y_t from local level model.

where $f(\mathbf{y}_t|\mathbf{Y}_{t-1};\boldsymbol{\varphi})$ is a conditional Gaussian density implied by the state space model (14.1) - (14.6). The vector of prediction errors $\mathbf{v}_t$ and prediction error variance matrices $\mathbf{F}_t$ are computed from the Kalman filter recursions.

A useful diagnostic is the estimated variance of the standardized prediction errors for a given value of $\boldsymbol{\varphi}$:

$$\hat{\sigma}^2(\boldsymbol{\varphi}) = \frac{1}{Nn}\sum_{t=1}^{n}\mathbf{v}_t'\mathbf{F}_t^{-1}\mathbf{v}_t \tag{14.45}$$

As mentioned by Koopman, Shephard and Doornik (1999), it is helpful to choose starting values for $\boldsymbol{\varphi}$ such that $\hat{\sigma}^2(\boldsymbol{\varphi}_{start}) \approx 1$. For well specified models, $\hat{\sigma}^2(\hat{\boldsymbol{\varphi}}_{mle})$ should be very close to unity.

Concentrated Log-likelihood

In some models, e.g. ARMA models, it is possible to solve explicitly for one scale factor and concentrate it out of the log-likelihood function (14.44). The resulting log-likelihood function is called the *concentrated log-likelihood* or *profile log-likelihood* and is denoted $\ln L^c(\boldsymbol{\varphi}|\mathbf{Y}_n)$. Following Koopman, Shephard and Doornik (1999), let σ denote such a scale factor, and let

$$\mathbf{y}_t = \boldsymbol{\theta}_t + \mathbf{G}_t^c\boldsymbol{\varepsilon}_t^c$$

with $\boldsymbol{\varepsilon}_t^c \sim iid\ N(0, \sigma^2\mathbf{I})$ denote the scaled version of the measurement equation (14.3). The state space form (14.1) - (14.3) applies but with $\mathbf{G}_t = \sigma\mathbf{G}_t^c$ and $\mathbf{H}_t = \sigma\mathbf{H}_t^c$. This formulation implies that one non-zero element of $\sigma\mathbf{G}_t^c$ or $\sigma\mathbf{H}_t^c$ is kept fixed, usually at unity, which reduces the dimension of the parameter vector $\boldsymbol{\varphi}$ by one. The solution for σ^2 from (14.44) is given by

$$\tilde{\sigma}^2(\boldsymbol{\varphi}) = \frac{1}{Nn}\sum_{t=1}^{n} \mathbf{v}_t' \left(\mathbf{F}_t^c\right)^{-1} \mathbf{v}_t$$

and the resulting concentrated log-likelihood function is

$$\ln L^c(\boldsymbol{\varphi}|\mathbf{Y}_n) = -\frac{nN}{2}\ln(2\pi) - \frac{nN}{2}\ln\left(\sigma^2(\boldsymbol{\varphi}) + 1\right) - \frac{1}{2}\sum_{t=1}^{n}\ln|\mathbf{F}_t^c| \quad (14.46)$$

14.4.2 Fitting State Space Models Using the S+FinMetrics/SsfPack Function SsfFit

The `S+FinMetrics/SsfPack` function `SsfFit` may be used to compute MLEs of the unknown parameters in the state space model (14.1)-(14.6) from the prediction error decomposition of the log-likelihood function (14.44). The arguments expected by `SsfFit` are

```
> args(SsfFit)
function(parm, data, FUN, conc = F, scale = 1, gradient =
NULL, hessian = NULL, lower =  - Inf, upper = Inf,
trace = T, control = NULL, ...)
```

where `parm` is a vector containing the starting values of the unknown parameters $\boldsymbol{\varphi}$, `data` is a rectangular object containing the response variables $\mathbf{y}_t$, and `FUN` is a character string giving the name of the function which takes `parm` together with the optional arguments in `...` and produces an "`ssf`" object representing the state space form. The remaining arguments control aspects of the `S-PLUS` optimization algorithm `nlminb`. An advantage of using `nlminb` is that box constraints may be imposed on the parameters of the log-likelihood function using the optional arguments `lower` and `upper`. See the online help for `nlminb` for details. A disadvantage of using `nlminb` is that the value of the Hessian evaluated at the MLEs is returned only if an analytic formula is supplied to compute the Hessian. The use of `SsfFit` is illustrated with the following examples.

Example 102 *Exact maximum likelihood estimation of AR(1) model*

Consider estimating by exact maximum likelihood the AR(1) model discussed earlier. First, $n = 250$ observations are simulated from the model

```
> ssf.ar1 = GetSsfArma(ar=0.75,sigma=.5)
> set.seed(598)
```

```
> sim.ssf.ar1 = SsfSim(ssf.ar1,n=250)
> y.ar1 = sim.ssf.ar1[,"response"]
```

Least squares estimation of the AR(1) model, which is equivalent to MLE conditional on the first observation, gives

```
> OLS(y.ar1~tslag(y.ar1)-1)

Call:
OLS(formula = y.ar1 ~tslag(y.ar1) - 1)

Coefficients:
 tslag(y.ar1)
 0.7739

Degrees of freedom: 249 total; 248 residual
Residual standard error: 0.4921
```

The `S+FinMetrics/SsfPack` function `SsfFit` requires as input a function which takes the unknown parameters $\boldsymbol{\varphi} = (\phi, \sigma^2)'$ and produces the state space form for the AR(1). One such function is

```
> ar1.mod = function(parm) {
+   phi = parm[1]
+   sigma2 = parm[2]
+   ssf.mod = GetSsfArma(ar=phi,sigma=sqrt(sigma2))
+   CheckSsf(ssf.mod)
+ }
```

In addition, starting values for $\boldsymbol{\varphi}$ are required. Somewhat arbitrary starting values are

```
> ar1.start = c(0.5,1)
> names(ar1.start) = c("phi","sigma2")
```

The prediction error decomposition of the log-likelihood function evaluated at the starting values $\boldsymbol{\varphi} = (0.5, 1)'$ may be computed using the `S+FinMetrics/SsfPack` function `KalmanFil` with `task="KFLIK"`

```
> KalmanFil(y.ar1,ar1.mod(ar1.start),task="KFLIK")

Call:
KalmanFil(mY = y.ar1, ssf = ar1.mod(ar1.start), task =
"KFLIK")

Log-likelihood:  -265.5222
Prediction error variance:  0.2851
Sample observations: 250
```

```
Standardized Innovations:
    Mean Std.Error
 -0.0238  0.5345
```

Notice that the standardized prediction error variance (14.45) is 0.285, far below unity, which indicates that the starting values are not very good.

The MLEs for $\boldsymbol{\varphi} = (\phi, \sigma^2)'$ using `SsfFit` are computed as

```
> ar1.mle = SsfFit(ar1.start,y.ar1,"ar1.mod",
+ lower=c(-.999,0),upper=c(0.999,Inf))
Iteration  0 : objective =  265.5
Iteration  1 : objective =  282.9
...
Iteration  18 : objective =  176.9
RELATIVE FUNCTION CONVERGENCE
```

In the call to `SsfFit`, the stationarity condition $-1 < \phi < 1$ and the positive variance condition $\sigma^2 > 0$ is imposed in the estimation. The result of `SsfFit` is a an object of class "`SsfFit`" with components

```
> names(ar1.mle)
 [1] "parameters" "objective"  "message"    "grad.norm"  "iterations"
 [6] "f.evals"    "g.evals"    "hessian"    "scale"      "aux"
[11] "call"       "vcov"
```

The exact MLEs for $\boldsymbol{\varphi} = (\phi, \sigma^2)'$ are

```
> ar1.mle
Log-likelihood:  -176.9
250 observations
Parameter Estimates:
     phi    sigma2
 0.77081 0.2402688
```

and the MLE for σ is

```
> sqrt(ar1.mle$parameters["sigma2"])
 sigma2
 0.4902
```

These values are very close to the least squares estimates. Standard errors and t-statistics are displayed using

```
> summary(ar1.mle)
Log-likelihood:  -176.936
250 observations
Parameters:
        Value Std. Error t value
   phi 0.7708    0.03977   19.38
sigma2 0.2403    0.02149   11.18
```

```
Convergence:  RELATIVE FUNCTION CONVERGENCE
```

A summary of the log-likelihood evaluated at the MLEs is

```
> KalmanFil(y.ar1,ar1.mod(ar1.mle$parameters),
+ task="KFLIK")

Call:
KalmanFil(mY = y.ar1, ssf = ar1.mod(ar1.mle$parameters),
task = "KFLIK")

Log-likelihood:  -176.9359
Prediction error variance:  1
Sample observations: 250

Standardized Innovations:
    Mean Std.Error
 -0.0213  1.0018
```

Notice that the estimated variance of the standardized prediction errors is equal to 1.

Example 103 *Exact maximum likelihood estimation of AR(1) model using re-parameterization*

An alternative function to compute the state space form of the AR(1) is

```
ar1.mod2 = function(parm) {
     phi = exp(parm[1])/(1 + exp(parm[1]))
     sigma2 = exp(parm[2])
     ssf.mod = GetSsfArma(ar=phi,sigma=sqrt(sigma2))
     CheckSsf(ssf.mod)
}
```

In the above model, a stationary value for ϕ between 0 and 1 and a positive value for σ^2 is guaranteed by parameterizing the log-likelihood in terms of $\gamma_0 = \ln(\phi/(1-\phi))$ and $\gamma_1 = \ln(\sigma^2)$ instead of ϕ and σ^2. By the invariance property of maximum likelihood estimation, $\hat{\phi}_{mle} = \exp(\hat{\gamma}_{0,mle})/(1+\exp(\hat{\gamma}_{0,mle}))$ and $\hat{\sigma}^2_{mle} = \exp(\hat{\gamma}_{1,mle})$ are the MLEs for ϕ and σ^2. The MLEs for $\boldsymbol{\varphi} = (\gamma_0, \gamma_1)'$ computed using `SsfFit` are

```
> ar1.start = c(0,0)
> names(ar1.start) = c("ln(phi/(1-phi))","ln(sigma2)")
> ar1.mle = SsfFit(ar1.start,y.ar1,"ar1.mod2")
Iteration  0 : objective =  265.5222
...
Iteration  10 : objective =  176.9359
```

```
RELATIVE FUNCTION CONVERGENCE
> summary(ar1.mle)
Log-likelihood:  -176.936
250 observations
Parameters:
                 Value Std. Error t value
ln(phi/(1-phi))  1.213    0.22510   5.388
     ln(sigma2) -1.426    0.08945 -15.940

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

The MLEs for ϕ and σ^2, and estimated standard errors computed using the delta method, are[5]

```
> ar1.mle2 = ar1.mle
> tmp = coef(ar1.mle)
> ar1.mle2$parameters[1] = exp(tmp[1])/(1 + exp(tmp[1]))
> ar1.mle2$parameters[2] = exp(tmp[2])
> dg1 = exp(-tmp[1])/(1 + exp(-tmp[1]))^2
> dg2 = exp(tmp[2])
> dg = diag(c(dg1,dg2))
> ar1.mle2$vcov = dg%*%ar1.mle2$vcov%*%dg
> summary(ar1.mle2)
Log-likelihood:  -176.936
250 observations
Parameters:
                 Value Std. Error t value
ln(phi/(1-phi)) 0.7708    0.03977   19.38
     ln(sigma2) 0.2403    0.02149   11.18

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

and exactly match the previous MLEs.

Example 104 *Exact maximum likelihood estimation of AR(1) model using concentrated log-likelihood*

In the AR(1) model, the variance parameter σ^2 can be analytically concentrated out of the log-likelihood. The advantages of concentrating the log-likelihood are to reduce the number of parameters to estimate, and to improve the numerical stability of the optimization. A function to compute the state space form for the AR(1) model, as a function of ϕ only, is

```
ar1.modc = function(parm) {
```

[5] Recall, if $\sqrt{n}(\hat{\theta}-\theta) \xrightarrow{d} N(\theta, V)$ and g is a continuous function then $\sqrt{n}(g(\hat{\theta})-g(\theta)) \xrightarrow{d} N(0, \frac{\partial g}{\partial \theta'} V \frac{\partial g}{\partial \theta'}')$.

```
     phi = parm[1]
     ssf.mod = GetSsfArma(ar=phi)
     CheckSsf(ssf.mod)
}
```

By default, the function `GetSsfArma` sets $\sigma^2 = 1$ which is required for the computation of the concentrated log-likelihood function from (14.46). To maximize the concentrated log-likelihood function (14.46) for the AR(1) model, use `SsfFit` with `ar1.modc` and set the optional argument `conc=T` :

```
> ar1.start = c(0.7)
> names(ar1.start) = c("phi")
> ar1.cmle = SsfFit(ar1.start,y.ar1,"ar1.modc",conc=T,
+ lower=0,upper=0.999)
Iteration  0 : objective =  178.506
...
Iteration  4 : objective =  176.9359
RELATIVE FUNCTION CONVERGENCE
> summary(ar1.cmle)
Log-likelihood:  -176.936
250 observations
Parameters:
     Value Std. Error t value
phi 0.7708    0.03977   19.38

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

Notice that with the concentrated log-likelihood, the optimizer converges in only four interations. The values of the log-likelihood and the MLE for ϕ are the same as found previously. The MLE for σ^2 may be recovered by running the Kalman filter and computing the variance of the prediction errors:

```
> ar1.KF = KalmanFil(y.ar1,ar1.mod(ar1.cmle$parameters),
+ task="KFLIK")
> ar1.KF$dVar
[1] 0.2403
```

One disadvantage of using the concentrated log-likelihood is the lack of an estimated standard error for $\hat{\sigma}^2$.

Example 105 *Maximum likelihood estimation of CAPM with time varying parameters*

Consider estimating the CAPM with time varying coefficients (14.17) - (14.19) using monthly data on Microsoft and the S&P 500 index over the period February, 1990 through December, 2000 contained in the `S+FinMetrics` "`timeSeries`" `excessReturns.ts`. The parameters of the model are the

variances of the innovations to the transition and measurement equations; $\sigma_\xi^2, \sigma_\varsigma^2$ and σ_ν^2. Since these variances must be positive the log-likelihood is parameterized using $\boldsymbol{\varphi} = (\ln(\sigma_\xi^2), \ln(\sigma_\varsigma^2), \ln(\sigma_\nu^2))'$. Since the state space form for the CAPM with time varying coefficients requires a data matrix $\mathbf{X}$ containing the excess returns on the S&P 500 index, the function `SsfFit` requires as input a function which takes both $\boldsymbol{\varphi}$ and $\mathbf{X}$ and returns the appropriate state space form. One such function is

```
tvp.mod = function(parm,mX=NULL) {
   parm = exp(parm)   # 3 x 1 vector containing log variances
   ssf.tvp = GetSsfReg(mX=mX)
   diag(ssf.tvp$mOmega) = parm
   CheckSsf(ssf.tvp)
}
```

Starting values for $\boldsymbol{\varphi}$ are specified as

```
> tvp.start = c(0,0,0)
> names(tvp.start) = c("ln(s2.alpha)","ln(s2.beta)","ln(s2.y)")
```

The maximum likelihood estimates for $\boldsymbol{\varphi}$ based on `SsfFit` are computed using

```
> tvp.mle = SsfFit(tvp.start,msft.ret,"tvp.mod",mX=X.mat)
Iteration  0 : objective =  183.2
...
Iteration  22 : objective =  -123
RELATIVE FUNCTION CONVERGENCE
```

The MLEs for $\boldsymbol{\varphi} = (\ln(\sigma_\xi^2), \ln(\sigma_\varsigma^2), \ln(\sigma_\nu^2))'$ are

```
> summary(tvp.mle)
Log-likelihood:  122.979
131 observations
Parameters:
               Value Std. Error t value
ln(s2.alpha) -12.480     2.8030  -4.452
 ln(s2.beta)  -5.900     3.0890  -1.910
    ln(s2.y)  -4.817     0.1285 -37.480

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

The estimates for the standard deviations σ_ξ, σ_ς and σ_ν as well as estimated standard errors, from the delta method, are:

```
> tvp2.mle = tvp.mle
> tvp2.mle$parameters = exp(tvp.mle$parameters/2)\qquad \qquad
> names(tvp2.mle$parameters) = c("s.alpha","s.beta","s.y")
> dg = diag(tvp2.mle$parameters/2)
```

```
> tvp2.mle$vcov = dg%*%tvp.mle$vcov%*%dg
> summary(tvp2.mle)
Log-likelihood:  122.979
131 observations
Parameters:
           Value Std. Error t value
s.alpha 0.001951   0.002734  0.7135
 s.beta 0.052350   0.080860  0.6474
    s.y 0.089970   0.005781 15.5600

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

It appears that the estimated standard deviations for the time varying parameter CAPM are close to zero, suggesting a constant parameter model.

The smoothed estimates of the time varying parameters α_t and β_t as well as the expected returns may be extracted and plotted using

```
> smoothedEst.tvp = SsfCondDens(y.capm,
+ tvp.mod(tvp.mle$parameters,mX=X.mat),
+ task="STSMO")
> plot(smoothedEst.tvp,strip.text=c("alpha(t)",
+ "beta(t)","Expected returns"),main="Smoothed Estimates",scales="free")
```

These estimates are illustrated in Figure 14.9. Notice the increase in $\hat{\beta}_t$ and decrease in $\hat{\alpha}_t$ over the sample.

14.4.3 Quasi-Maximum Likelihood Estimation

The `S+FinMetrics` function `SsfFitFast` is a fast version of `SsfFit` that also computes the sandwhich covariance matrix estimate which is appropriate for QMLE. This matrix is given in the `r.vcov` component of the "`SsfFit`" object returned by `SsfFitFast`.

Example 106 *Quasi-maximum likelihood estimation of a stochastic volatility model*

Let r_t denote the continuously compounded return on an asset between times $t-1$ and t. Following Harvey, Ruiz and Shephard (1994), hereafter HRS, a simple discrete-time stochastic volatility (SV) model has the form

$$\begin{aligned} r_t &= \sigma_t \varepsilon_t, \qquad \varepsilon_t \sim \text{iid } N(0,1) \\ h_t &= \ln \sigma_t^2 = \gamma + \phi h_{t-1} + \eta_t, \qquad \eta_t \sim \text{iid } N(0, \sigma_\eta^2) \\ E[\varepsilon_t \eta_t] &= 0 \end{aligned} \tag{14.47}$$

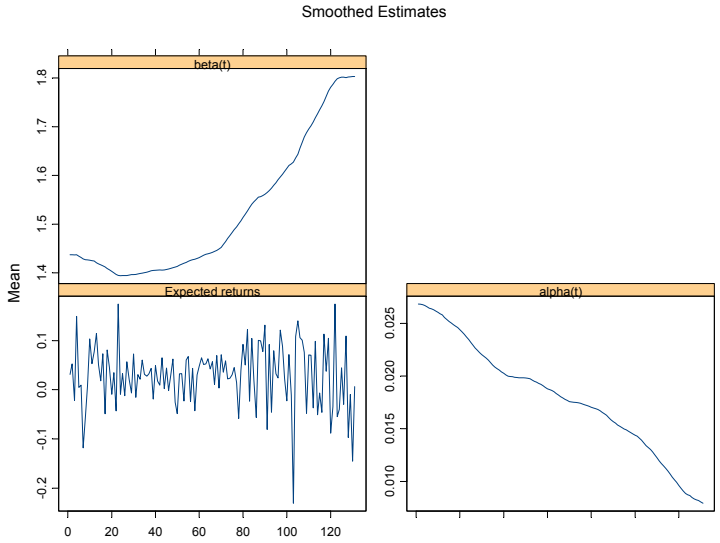

FIGURE 14.9. Smoothed estimates of α_t and β_t from CAPM with time varying parameter fit to the monthly excess returns on Microsoft.

Defining $y_t = \ln r_t^2$, and noting that $E[\ln \varepsilon_t^2] = -1.27$ and $\text{var}(\ln \varepsilon_t^2) = \pi^2/2$ an unobserved components state space representation for y_t has the form

$$\begin{aligned} y_t &= -1.27 + h_t + \xi_t, \qquad \xi_t \sim \text{iid } (0, \pi^2/2) \\ h_t &= \gamma + \phi h_{t-1} + \eta_t, \qquad \eta_t \sim \text{iid } N(0, \sigma_\eta^2) \\ E[\xi_t \eta_t] &= 0 \end{aligned}$$

If ξ_t were iid Gaussian then the parameters $\boldsymbol{\varphi} = (\gamma, \sigma_\eta^2, \phi)'$ of the SV model could be efficiently estimated by maximizing the prediction error decomposition of the log-likelihood function constructed from the Kalman filter recursions. However, since $\xi_t = \ln \varepsilon_t^2$ is not normally distributed the Kalman filter only provides minimum mean squared error linear estimators of the state and future observations. Nonetheless, HRS point out that even though the exact log-likelihood cannot be computed from the prediction error decomposition based on the Kalman filter, consistent estimates of $\boldsymbol{\varphi} = (\gamma, \sigma_\eta^2, \phi)'$ can still be obtained by treating ξ_t as though it were iid $N(0, \pi^2/2)$ and maximizing the quasi log-likelihood function constructed from the prediction error decomposition.

The state space representation of the SV model has system matrices

$$\boldsymbol{\delta} = \begin{pmatrix} \gamma \\ -1.27 \end{pmatrix}, \qquad \boldsymbol{\Phi} = \begin{pmatrix} \phi \\ 1 \end{pmatrix}, \qquad \boldsymbol{\Omega} = \begin{pmatrix} \sigma_\eta^2 & 0 \\ 0 & \pi^2/2 \end{pmatrix}$$

Assuming that $|\phi| < 1$, the initial value matrix has the form

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_\eta^2/(1-\phi^2) \\ \gamma/(1-\phi) \end{pmatrix}$$

If $\phi = 1$ then use

$$\mathbf{\Sigma} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

A function to compute the state space form of the SV model given a vector of parameters, assuming $|\phi| < 1$, is

```
sv.mod = function(parm) {
     g = parm[1]
     sigma2.n = exp(parm[2])
     phi = parm[3]
     ssf.mod = list(mDelta=c(g,-1.27),
     mPhi=as.matrix(c(phi,1)),
     mOmega=matrix(c(sigma2.n,0,0,0.5*pi^2),2,2),
     mSigma=as.matrix(c((sigma2.n/(1-phi^2)),g/(1-phi))))
     CheckSsf(ssf.mod)
}
```

Notice that an exponential transformation is utilized to ensure a positive value for σ_η^2. A sample of $T = 1000$ observations are simulated from the SV model using the parameters $\gamma = -0.3556$, $\sigma_\eta^2 = 0.0312$ and $\phi = 0.9646$:

```
> parm.hrs = c(-0.3556,log(0.0312),0.9646)
> nobs = 1000
> set.seed(179)
> e = rnorm(nobs)
> xi = log(e^2)+1.27
> eta = rnorm(nobs,sd=sqrt(0.0312))
> sv.sim = SsfSim(sv.mod(parm.hrs),
+ mRan=cbind(eta,xi),a1=(-0.3556/(1-0.9646)))
```

The first 250 simulated squared returns, r_t^2, and latent squared volatilities, σ_t^2, are shown in Figure 14.10.

Starting values for the estimation of $\boldsymbol{\varphi} = (\gamma, \ln \sigma_\eta^2, \phi)'$ are values close to the true values:

```
> sv.start = c(-0.3,log(0.03),0.9)
> names(sv.start) = c("g","ln.sigma2","phi")
```

Using `SsfFitFast`, the quasi-maximum likelihood (QML) estimates are

```
> low.vals = c(-Inf,-Inf,-0.999)
> up.vals = c(Inf,Inf,0.999)
> sv.mle = SsfFitFast(sv.start,sv.sim[,2],"sv.mod",
```

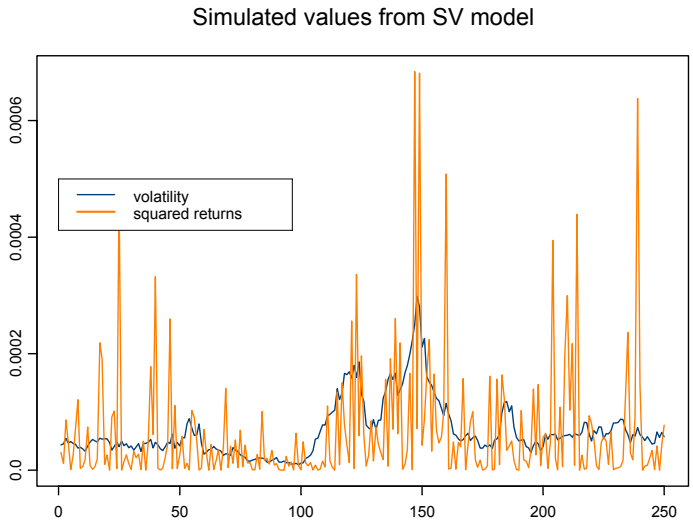

FIGURE 14.10. Simulated values from SV model.

```
+ lower=low.vals,upper=up.vals)
Iteration  0 : objective =  5.147579
...
Iteration  15 : objective =  2.21826
RELATIVE FUNCTION CONVERGENCE
```

To show the estimates with the QMLE standard errors use `summary` with the optional argument `method=\qmle"`

```
> summary(sv.mle,method="qmle")
Log-likelihood:  -2218.26
1000 observations
Parameters:
            Value QMLE Std. Error t value
        g -0.4810         0.18190  -2.644
ln.sigma2 -3.5630         0.53020  -6.721
      phi  0.9509         0.01838  51.740

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

Using the delta method, the QMLE and estimated standard error for σ_η^2 are 0.02834 and 0.01503, respectively.

The filtered and smoothed estimates of log-volatility are computed using

```
> ssf.sv = sv.mod(sv.mle2$parameters)
```

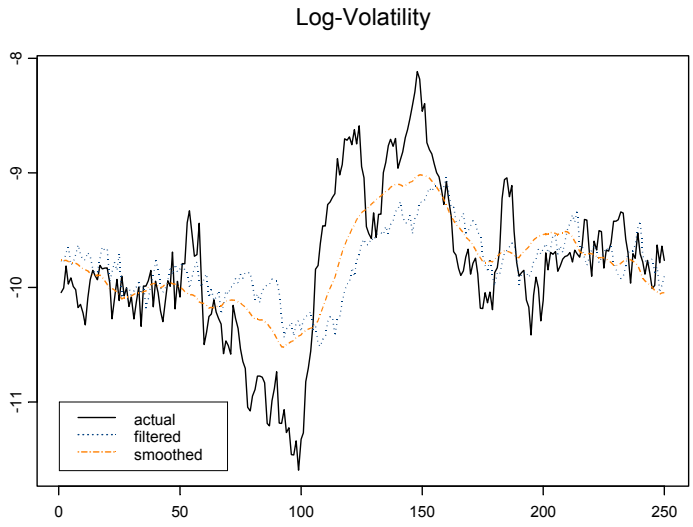

FIGURE 14.11. Log-volatility along with filtered and smoothed estimates from SV model.

```
> filteredEst.sv = SsfMomentEst(sv.sim[,2],ssf.sv,task="STFIL")
> smoothedEst.sv = SsfCondDens(sv.sim[,2],ssf.sv,task="STSMO")
```

The first 250 estimates along with actual log-volatility are illustrated in Figure 14.11

14.5 Simulation Smoothing

The simulation of state and response vectors $\boldsymbol{\alpha}_t$ and $\mathbf{y}_t$ or disturbance vectors $\boldsymbol{\eta}_t$ and $\boldsymbol{\varepsilon}_t$ conditional on the observations $\mathbf{Y}_n$ is called *simulation smoothing*. Simulation smoothing is useful for evaluating the appropriateness of a proposed state space model and for the Bayesian analysis of state space models using Markov chain Monte Carlo (MCMC) techniques. The `S+FinMetrics/SsfPack` function `SimSmoDraw` generates random draws from the distributions of the state and response variables or from the distributions of the state and response disturbances. The arguments expected by `SimSmoDraw` are

```
> args(SimSmoDraw)
function(kf, ssf, task = "DSSIM", mRan = NULL, a1 = NULL)
```

where `kf` is a "`KalmanFil`" object, `ssf` is a list which either contains the minimal necessary components for a state space form or is a valid "`ssf`" object and `task` determines whether the state smoothing ("`STSIM`") or disturbance smoothing ("`DSSIM`") is performed.

Example 107 *Simulation smoothing from the local level model*

Simulated state and response values from the local level model may be generated using

```
> KalmanFil.ll = KalmanFil(y.ll,ssf.ll,task="STSIM")
> ll.state.sim = SimSmoDraw(KalmanFil.ll,ssf.ll,
+ task="STSIM")
> class(ll.state.sim)
[1] "SimSmoDraw"
> names(ll.state.sim)
[1] "state"    "response" "task"
```

The resulting simulated values may be visualized using

```
> plot(ll.state.sim,layout=c(1,2))
```

To simulate disturbances from the state and response equations, set `task="DSSIM"` in the calls to `KalmanFil` and `SimSmoDraw`.

14.6 References

[1] Bomhoff, E. J. (1994). *Financial Forecasting for Business and Economics*. Academic Press, San Diego.

[2] Carmona, R. (2001). *Statistical Analysis of Financial Data, with an implementation in Splus*. Textbook under review.

[3] Chan, N.H. (2002). *Time Series: Applicatios to Finance*. John Wiley & Sons, New York.

[4] Durbin, J. and S.J. Koopman (2001). *Time Series Analysis by State Space Methods*. Oxford University Press, Oxford.

[5] Duan, J.-C. and J.-G. Simonato (1999). "Estimating Exponential-Affine Term Structure Models by Kalman Filter," Review of Quantitative Finance and Accounting, 13, 111-135.

[6] Hamilton, J.D. (1994). *Time Series Analysis*. Princeton University Press, Princeton.

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

[8] Harvey, A.C. (1993). *Time Series Models*, 2nd edition. MIT Press, Cambridge.

[9] Harvey, A.C., E. Ruiz and N. Shephard (1994). "Multivariate Stochastic Variance Models," *Review of Economic Studies*, 61, 247-264.

[10] Kim, C.-J., and C.R. Nelson (1999). *State-Space Models with Regime Switching*. MIT Press, Cambridge.

[11] Koopman, S.J., N. Shephard, and J.A. Doornik (1999). "Statistical Algorithms for State Space Models Using SsfPack 2.2," *Econometrics Journal*, 2, 113-166.

[12] Koopman, S.J., N. Shephard, and J.A. Doornik (2001). "SsfPack 3.0beta: Statistical Algorithms for Models in State Space," unpublished manuscript, Free University, Amsterdam.

[13] Engle, R.F. and M.W. Watson (1987). "The Kalman Filter: Applications to Forecasting and Rational Expectations Models," in T.F. Bewley (ed.) *Advances in Econometrics: Fifth World Congress, Volume I*. Cambridge University Press, Cambridge.

[14] West, M. and J. Harrison (1997). *Bayesian Forecasting and Dynamic Models*, 2nd edition. Springer-Verlag, New York.

[15] Zivot, E., Wang, J. and S.J. Koopman (2004). "State Space Models in Economics and Finance Using SsfPack in S+FinMetrics," chapter 14 in *Unobserved Components Models* (A. Harvey, S.J. Koopman, and N. Shephard eds.), Cambridge University Press.