Journal of Statistical Software

May 2011, Volume 41, Issue 5. *http://www.jstatsoft.org/*

State Space Modeling Using SsfPack in S+FinMetrics 3.0

Eric W. Zivot
University of Washington

Abstract

This paper presents two illustrations of state space modeling in S-PLUS using the **SsfPack** 3.0 routines implemented in **S+FinMetrics** 3.0. The state space modeling functions in **S+FinMetrics/SsfPack** are extremely flexible and powerful and can be used for a wide variety of linear Gaussian state space models and for some non-linear and non-Gaussian state space models.

Keywords: state space models, software tools, S-PLUS.

1. Introduction

In S-PLUS (TIBCO Software Inc. 2010), state space modeling is implemented in the add-on module **S+FinMetrics** as described in Zivot, Wang, and Koopman (2004), Tsay (2005), and Zivot and Wang (2006). 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, 2008)[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).

In **S+FinMetrics/SsfPack**, the linear Gaussian *state space model* for a multivariate time series $\mathbf{y}_t$ is the system of equations

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

$$\underset{m\times 1}{\alpha_{t+1}} = \underset{m\times 1}{\mathbf{d}_t} + \underset{m\times m}{\mathbf{T}_t} \cdot \underset{m\times 1}{\alpha_t} + \underset{m\times r}{\mathbf{H}_t} \cdot \underset{r\times 1}{\eta_t}, \tag{1}$$

$$\underset{N\times 1}{\theta_t} = \underset{N\times 1}{\mathbf{c_t}} + \underset{N\times m}{\mathbf{Z_t}} \cdot \underset{m\times 1}{\alpha_t}, \tag{2}$$

$$\underset{N\times 1}{\mathbf{y}_t} = \underset{N\times 1}{\theta_t} + \underset{N\times N}{\mathbf{G}_t} \cdot \underset{N\times 1}{\varepsilon_t}, \tag{3}$$

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

$$\alpha_1 \sim N(\mathbf{a}, \mathbf{P}), \tag{4}$$

$$\eta_t \sim \text{iid } N(0, \mathbf{I}_r), \tag{5}$$

$$\varepsilon_t \sim \text{iid } N(\mathbf{0}, \mathbf{I}_N), \tag{6}$$

and it is assumed that

$$E[\varepsilon_t \eta_t^\top] = \mathbf{0}.$$

The *state vector* α_t contains unobserved stochastic processes and unknown fixed effects and the *transition equation* (1) describes the evolution of the state vector over time using a first order Markov structure. The *measurement equation* (3) describes the vector of observations $\mathbf{y}_t$ in terms of the state vector α_t through the signal θ_t and a vector of disturbances ε_t. 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 (1)–(6) may be compactly expressed as

$$\begin{pmatrix} \alpha_{t+1} \\ \mathbf{y}_t \end{pmatrix} = \underset{(m+N)\times 1}{\delta_t} + \underset{(m+N)\times m}{\mathbf{\Phi}_t} \cdot \underset{m\times 1}{\alpha_t} + \underset{(m+N)\times 1}{\mathbf{u}_t}, \tag{7}$$

$$\alpha_1 \sim N(\mathbf{a}, \mathbf{P}), \tag{8}$$

$$\mathbf{u}_t \sim \text{iid } N(\mathbf{0}, \mathbf{\Omega}_t), \tag{9}$$

where

$$\delta_t = \begin{pmatrix} \mathbf{d}_t \\ \mathbf{c}_t \end{pmatrix}, \mathbf{\Phi}_t = \begin{pmatrix} \mathbf{T}_t \\ \mathbf{Z}_t \end{pmatrix}, \mathbf{u}_t = \begin{pmatrix} \mathbf{H}_t \eta_t \\ \mathbf{G}_t \varepsilon_t \end{pmatrix}, \mathbf{\Omega}_t = \begin{pmatrix} \mathbf{H}_t \mathbf{H}_t^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_t \mathbf{G}_t^\top \end{pmatrix}.$$

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

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{P} \\ \mathbf{a}^\top \end{pmatrix}. \tag{10}$$

For multivariate models, i.e., $N > 1$, it is assumed that the $N \times N$ matrix $\mathbf{G}_t \mathbf{G}_t^\top$ is diagonal. State space models in **S+FinMetrics/SsfPack** utilize the compact representation (7) with initial value information (10).

The variance matrix $\mathbf{P}$ of the initial state vector α_1 is assumed to be of the form

$$\mathbf{P} = \mathbf{P}_* + \kappa \mathbf{P}_\infty \tag{11}$$

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 *et al.* (2008).

The remainder of this paper is organized as follows. Section 2 gives an overview of state space modeling using **S+FinMetrics/SsfPack** for the annual flow volume of the river Nile based on the local level model. Section 3 illustrates multivariate state space modeling of affine term structure models, and Section 4 gives some final comments.

2. The local level model for the Nile river data

Consider a univariate time series y_t representing the annual flow volume of the river Nile over the period 1871 to 1970. This data series is included in **S+FinMetrics** in the "`timeSeries`" object `nile.dat`, and is shown in Figure 1. The local level model for y_t has the form

$$\alpha_{t+1} = \alpha_t + \eta_t^*, \ \eta_t^* \sim \text{iid } N(0, \sigma_\eta^2), \tag{12}$$

$$y_t = \alpha_t + \varepsilon_t^*, \ \varepsilon_t^* \sim \text{iid } N(0, \sigma_\varepsilon^2), \tag{13}$$

$$\alpha_1 \sim N(a, P), \tag{14}$$

where it is assumed that $E[\varepsilon_t^* \eta_t^*] = 0$. In the local level model, 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 trend behavior of y_t. The transition equation (12) shows that the trend follows a random walk. The component ε_t^* represents random deviations (noise) from the trend that are assumed to

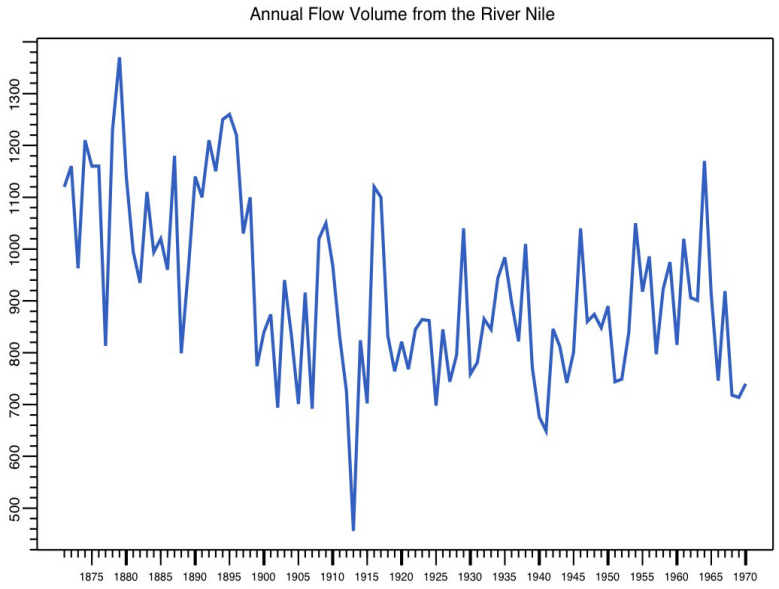

Figure 1: Nile river data from the **S+FinMetrics** "`timeSeries`" object `nile.dat`.

State space parameter	List component name
δ	`mDelta`
$\mathbf{\Phi}$	`mPhi`
$\mathbf{\Omega}$	`mOmega`
$\mathbf{\Sigma}$	`mSigma`

Table 1: State space form list components.

be independent from the innovations to α_t. The strength of the signal in the trend relative to the random deviation is measured by the signal-to-noise ratio of variances $q = \sigma_\eta^2/\sigma_\varepsilon^2$. The state space form (7) of the local level model has time invariant parameters

$$\delta = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Phi = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Omega = \begin{pmatrix} \sigma_\eta^2 & 0 \\ 0 & \sigma_\varepsilon^2 \end{pmatrix}, \tag{15}$$

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(α_1) to some large positive number, e.g., $P = 10^7$, in (14) to reflect that no prior information is available. Using (11), the initial variance is specified with $P_* = 0$ and $P_\infty = 1$. Therefore, the initial state matrix (10) for the local level model has the form

$$\Sigma = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \tag{16}$$

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 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 (15)–(16) calibrated to the Nile river data ($\sigma_\eta^2 =$ 1469.1 and $\sigma_\varepsilon^2 = 15099$, which are the maximum likelihood values determined in Section 2.2.

```
> sigma.e = sqrt(15099)
> sigma.n = sqrt(1469.1)
> 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,] 1469.1     0
[2,]    0.0 15099

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

In the list variable `ssf.ll.list`, the component names match the state space form parameters in (7) and (10). This naming convention, summarized in Table 1, must be used for the specification of any valid state space model.

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:          $mOmega:                $mSigma:       $mDelta:
     [,1]              [,1]  [,2]            [,1]           [,1]
[1,]    1      [1,] 1469.1     0      [1,]   -1      [1,]    0
[2,]    1      [2,]    0.0 15099      [2,]    0      [2,]    0

$mJPhi: $mJOmega: $mJDelta: $mX:   $cT:   $cX:   $cY:   $cSt:
[1] 0   [1] 0     [1] 0     [1] 0  [1] 0  [1] 0  [1] 1  [1] 1
```

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.

2.1. 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.

To generate 100 observations on the state variable α_{t+1} and observations y_t in the local level model (12)–(14) calibrated to the Nile river (with initial value $a_1 = y_0 = 1120$) data use

```
> set.seed(112)
> ssf.ll.list$mSigma[2] = 1120
> ll.sim = SsfSim(ssf.ll.list, n = 100)
```

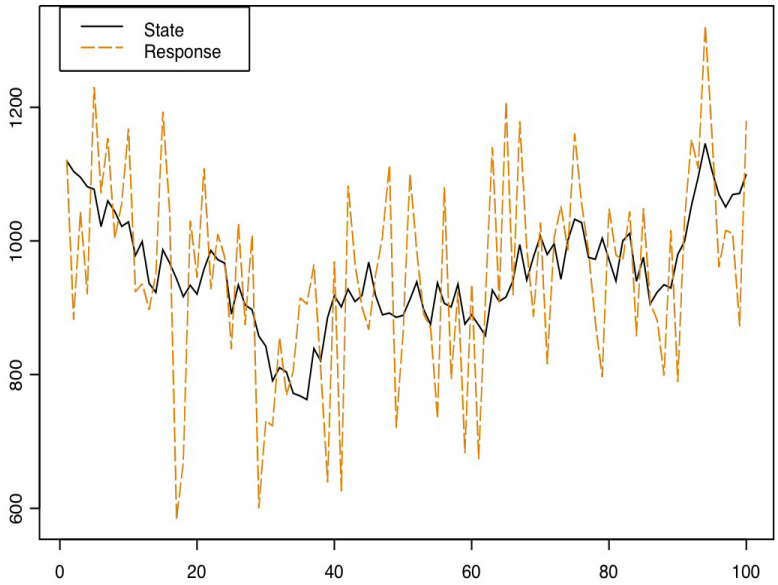

Figure 2: Simulated values from local level model created using the **S+FinMetrics** function `SsfSim`.

```
> 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 2.

2.2. State space modeling algorithms: Filtering, smoothing and forecasting

The **S+FinMetrics/SsfPack** function `KalmanFil` implements the Kalman filter forward recursions in a computationally efficient way, see Koopman *et al.* (2008). The **S+FinMetrics/SsfPack** function `KalmanSmo` computes the Kalman smoother backwards recursions. 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 α_t and $\mathbf{y}_t$, with estimated variances, as well as smoothed estimates of ε_t and η_t, with estimated variances, are computed using the **S+FinMetrics/SsfPack** function `SsfMomentEst`. The function `SsfMomentEst` may also be used to compute out-of-sample forecasts and forecast variances of α_t and $\mathbf{y}_t$. More conveniently, out-of-sample $h-$step ahead forecasts for α_t and $\mathbf{y}_t$, along with forecasts variances, are computed using the **S+FinMetrics/SsfPack** function `SsfForecast`. The next sub-sections illustrate the use of the **S+FinMetrics/SsfPack** functions for implementing the state space algorithms for the local level model (12)–(14) calibrated to the Nile river data.

Kalman filter

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

```
> KalmanFil.ll = KalmanFil(nile.dat, 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"    "mGain"        "loglike"
 [6] "loglike.conc" "dVar"         "mEst"         "mOffP"        "n.predict"
[11] "task"         "err"          "call"         "positions"
```

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: 100 rows, 3 columns.
         [,1]   [,2]       [,3]
 [1,]    0.00 1.0000 0.00000000
 [2,]   40.00 0.5232 0.00003158
 [3,] -177.93 0.3829 0.00004087
...
[100,]  -79.64 0.267 0.00004854
```

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: 100 rows, 4 columns.
        [,1]   [,2]  [,3]  [,4]
 [1,] 1120.0 1120.0 15099 15099
 [2,] 1140.9 1140.9  7900  7900
 [3,] 1072.8 1072.8  5781  5781
...
[100,] 798.4 798.4 4032 4032
```

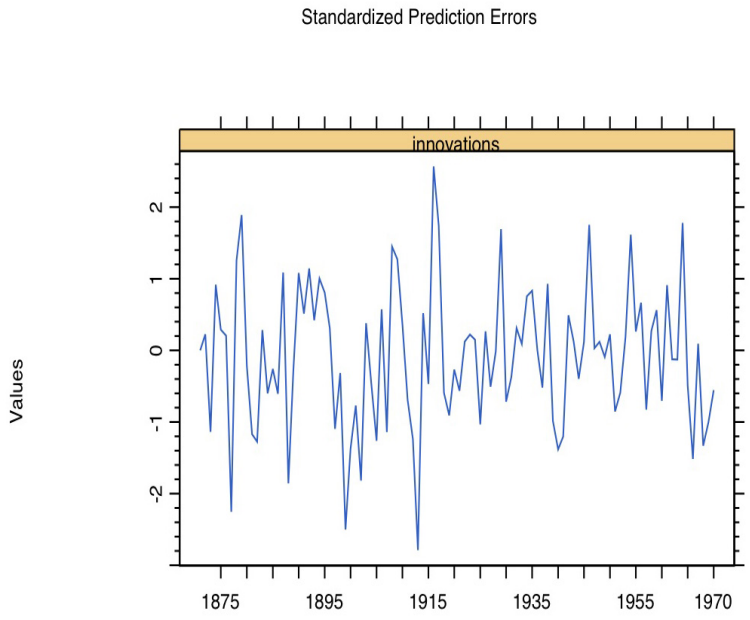

Figure 3: Standardized innovations from Kalman filter applied to local level model calibrated to Nile river data.

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 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)
> class(KalmanSmo.ll)

[1] "KalmanSmo"
```

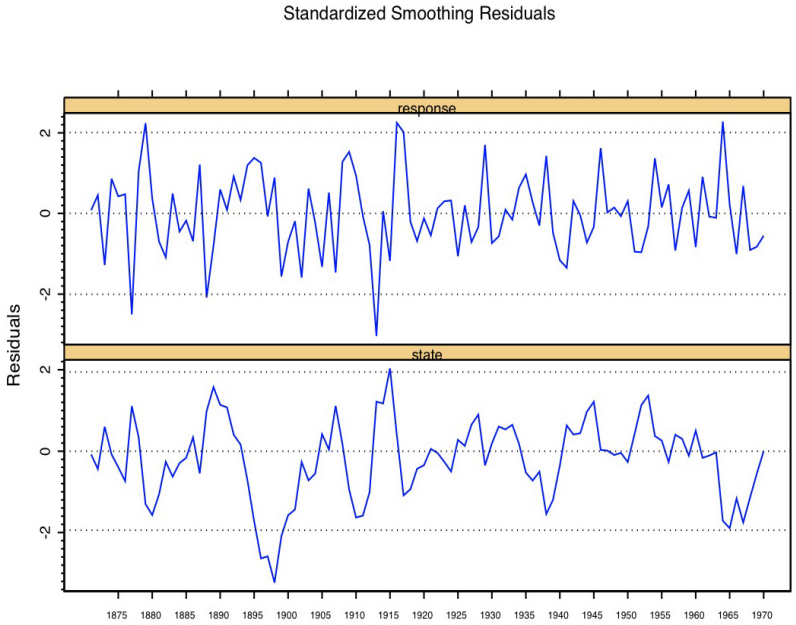

Figure 4: Standardized smoothing residuals from Kalman smoother recursions computed from local level model calibrated to Nile river data.

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" "state.variance"
[4] "response.variance"  "aux.residuals"      "scores"
[7] "positions"          "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 4, is created with the `plot` method.

Filtered and smoothed moment estimates

Filtered and smoothed estimates of α_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(nile.dat, ssf.ll, task = "STFIL")
> class(FilteredEst.ll)
```

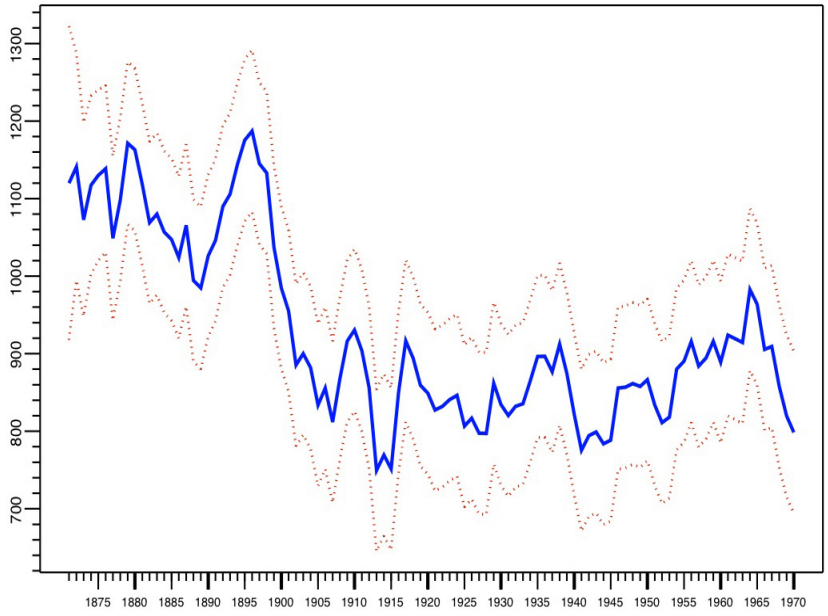

Figure 5: Filtered estimates of α_t and y_t computed from the local level model, calibrated to the Nile river data.

```
[1] "SsfMomentEst"
> names(FilteredEst.ll)
[1] "state.moment"      "state.variance"    "response.moment"
[4] "response.variance" "task"              "positions"
```

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 (13) in the local level model, $a_{t|t} = y_{t|t}$:

```
> FilteredEst.ll$state.moment[1:5]

[1] 1120 1141 1073 1117 1130

> FilteredEst.ll$response.moment[1:5]

[1] 1120 1141 1073 1117 1130
```

The `plot` method creates a multi-panel timeplot of the estimates of α_t and y_t without standard error bars.

A plot of the filtered state estimates with 90% confidence intervals is shown in Figure 5.

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

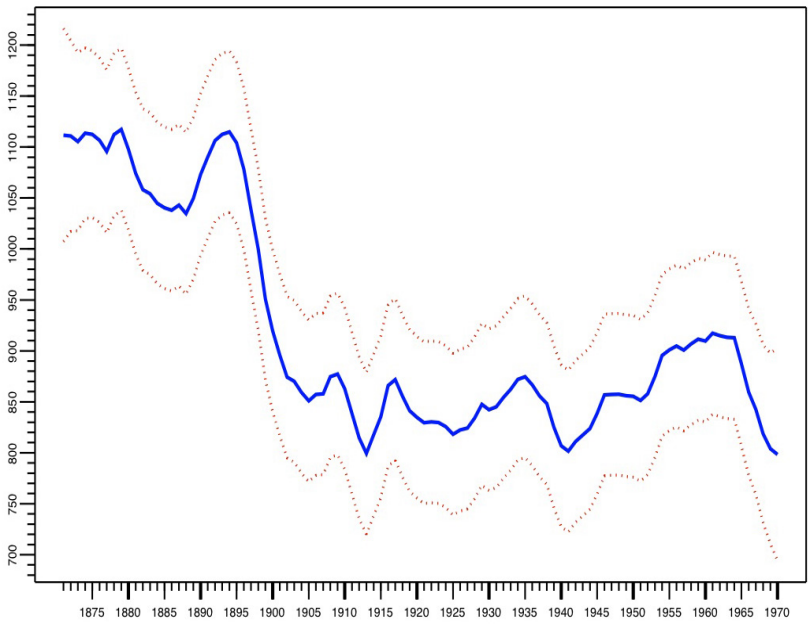

Figure 6: Smoothed estimates of α_t with 90% confidence intervals computed from the local level model calibrated to the Nile river data.

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

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

```
> SmoothedEst.ll$state.moment[1:5]

[1] 1112 1111 1105 1114 1112

> SmoothedEst.ll$response.moment[1:5]

[1] 1112 1111 1105 1114 1112
```

The smoothed state estimates with 90% confidence bands are illustrated in Figure 6. 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(nile.dat, 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 smoothed estimates $\hat{y}_t$ and $\hat{\alpha}_t$ may be visualized using the `plot` method for "`SsfCondDens`" objects.

Forecasting

For a state space model, the Kalman filter prediction equations produces one-step ahead predictions of the state vector, along with prediction variance matrices. You can compute out of sample predictions and associated mean square errors from the Kalman filter prediction equations by extending the sample data set $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ with a set of missing (NA) values. When y_τ is missing, the Kalman filter reduces to the prediction step. As a result, a sequence of m missing values at the end of the sample produces a set of h-step ahead forecasts for $h = 1, \ldots, m$.

To produce out-of-sample h-step ahead forecasts $y_{t+h|t}$ for $h = 1, \ldots, 10$ a sequence of 10 missing values is appended to the end of the Nile river data

```
> td.old = positions(nile.dat)
> td.new = timeCalendar(y = 1971:2000, format = td.old@format)
> td = concat(td.old, td.new)
> nile.dat.new = timeSeries(data = concat(seriesData(nile.dat),
+   rep(NA, 10)), positions = td)
```

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

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

The actual values, forecasts and 50% confidence bands are illustrated in Figure 7.

The **S+FinMetrics** function `SsfForecast` automates the process of forecasting from a state space model. The function takes the following arguments:

```
> args(SsfForecast)

function(mY, ssf, ikf = NULL, newdata = NULL, n.predict = 1)
```

where `mY` denotes the series to be forecast, `ssf` denotes the state space object with response variable `mY`, `ikf` denotes an object returned from a call to `KalmanIni`, `newdata` holds the out-of-sample values of any exogenous variables, and `n.predict` specifies the number of h-step-ahead predictions to compute. The returned value is an **S+FinMetrics** "`forecast`" object for which there are `print`, `summary` and `plot` methods.

To produce 30 h-step-ahead out-of-sample forecasts for the local level model calibrated to the Nile River data, use

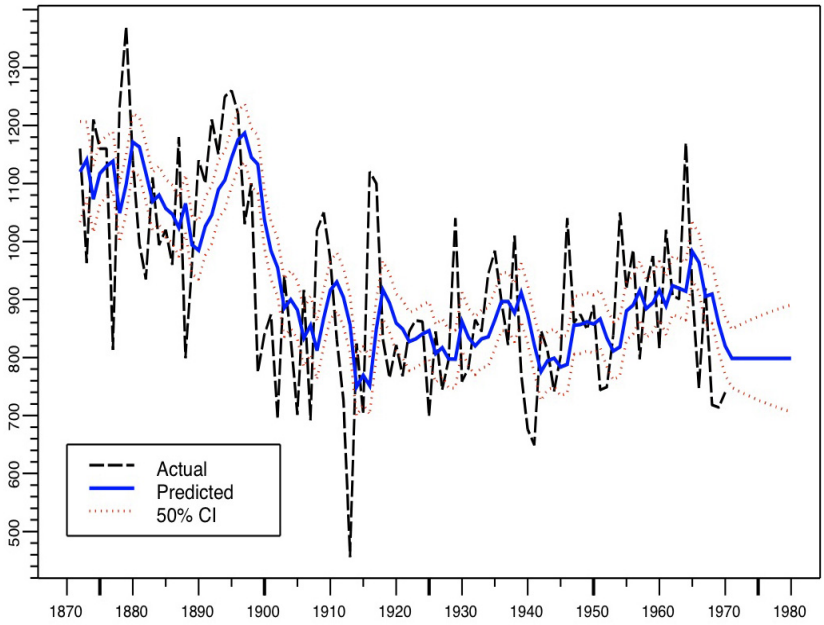

Figure 7: Actual values, h-step forecasts and 50% confidence intervals for y_t from local level model calibrated to Nile river data.

```
> nile.dat.fcst2 = SsfForecast(nile.dat, ssf.ll, n.predict = 10)
> class(nile.dat.fcst2)

[1] "forecast"
```

The returned object is of class "`forecast`". The `summary` method shows the forecasts with standard errors:

```
> summary(nile.dat.fcst2)

Predicted Values with Standard Errors:

              prediction std.err
 1-step-ahead 798.37      74.17
 2-step-ahead 798.37      83.49
 3-step-ahead 798.37      91.87
...
30-step-ahead 798.37     219.33
```

Use the `plot` method to plot the forecasts with standard error bands along with a portion of the original data. The following command plots the forecasts with $\pm$ 0.6745 standard error bands (50% confidence bands) and the last 10 observations of the data:

```
> plot(nile.data.fcst2, xold = nile.dat, width = 0.6745, n.old = 10)
```

Figure 8 shows the resulting plot.

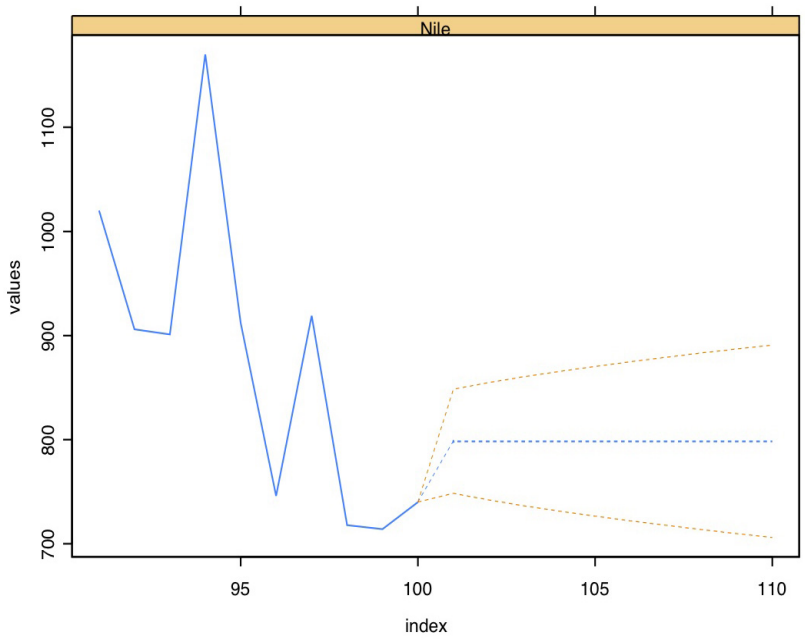

Figure 8: Out-of-sample forecasts computed with the **S+FinMetrics/SsfPack** function `SsfForecast`.

Maximum likelihood estimation of state space models

The *prediction error decomposition* of the log-likelihood function for the unknown parameters φ of a state space model may be conveniently computed using the output of the Kalman filter

$$\begin{aligned} \ln L(\varphi|Y_n) &= \sum_{t=1}^{n} \ln f(\mathbf{y}_t|\mathbf{Y}_{t-1};\varphi) \\ &= -\frac{nN}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{n}\left(\ln|\mathbf{F}_t| + \mathbf{v}_t^\top \mathbf{F}_t^{-1}\mathbf{v}_t\right), \end{aligned} \tag{17}$$

where $f(\mathbf{y}_t|\mathbf{Y}_{t-1};\varphi)$ is a conditional Gaussian density implied by the state space model (1)–(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 φ:

$$\hat{\sigma}^2(\varphi) = \frac{1}{Nn}\sum_{t=1}^{n} \mathbf{v}_t^\top \mathbf{F}_t^{-1}\mathbf{v}_t. \tag{18}$$

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

In some models, e.g., the local level model, it is possible to solve explicitly for one scale factor and concentrate it out of the log-likelihood function (17). The resulting log-likelihood function is called the *concentrated log-likelihood* or *profile log-likelihood* and is denoted $\ln L^c(\varphi|\mathbf{Y}_n)$.

Following Koopman *et al.* (1999), let σ denote such a scale factor, and let

$$\mathbf{y}_t = \theta_t + \mathbf{G}_t^c \varepsilon_t^c,$$

with $\varepsilon_t^c \sim$iid $N(0, \sigma^2\mathbf{I})$ denote the scaled version of the measurement equation (3). The state space form (1)–(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 φ by one. The solution for σ^2 from (17) is given by

$$\tilde{\sigma}^2(\varphi) = \frac{1}{Nn} \sum_{t=1}^{n} \mathbf{v}_t^\top \left(\mathbf{F}_t^c\right)^{-1} \mathbf{v}_t,$$

and the resulting concentrated log-likelihood function is

$$\ln L^c(\varphi|\mathbf{Y}_n) = -\frac{nN}{2}\ln(2\pi) - \frac{nN}{2}\ln\left(\sigma^2(\varphi)+1\right) - \frac{1}{2}\sum_{t=1}^{n}\ln|\mathbf{F}_t^c|. \tag{19}$$

The **S+FinMetrics/SsfPack** function `SsfFit` may be used to compute MLEs of the unknown parameters in the state space model (1)–(6) from the prediction error decomposition of the log-likelihood function (17). 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 φ, `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.

To estimate the unknown parameters $\varphi = (\sigma_\eta^2, \sigma_\varepsilon^2)^\top$ of the local level model, the **S+FinMetrics/SsfPack** function `SsfFit` requires as input a function which takes the parameters φ and produces the state space form for the local level model. One such function is:

```
> ll.mod = function(parm) {
+   parm = sqrt(parm)
+   ssf.mod = GetSsfStsm(irregular = parm[2], level = parm[1])
+   CheckSsf(ssf.mod)
+ }
```

where we note that `parm[1] = sig2.n` (i.e., σ_η^2) and `parm[2] = sig2.e` (i.e., σ_ε^2) upon entry of the function.

In addition, starting values for φ are required. Somewhat arbitrary starting values are $\sigma_\eta^2 = 1000$ and $\sigma_\varepsilon^2 = 10000$. The prediction error decomposition of the log-likelihood function evaluated at the starting values $\varphi = (1000, 10000)^\top$ may be computed using the **S+FinMetrics/ SsfPack** function `KalmanFil` with `task = "KFLIK"`:

```
> ll.start = c(1000, 10000)
> names(ll.start) = c("sig2.n", "sig2.e")
> KalmanFil(nile.dat, ll.mod(ll.start), task = "KFLIK")

Call:
KalmanFil(mY = nile.dat, ssf = ll.mod(ll.start), task = "KFLIK")

Log-likelihood:  -638.2044
Prediction error variance:  1.5036
Sample observations: 100

Standardized Innovations:
    Mean Std.Error
 -0.1012  1.2220
```

Notice that the standardized prediction error variance (18) is 1.5036, slightly above unity, which indicates that the starting value for φ is fairly close to the maximum likelihood estimate (MLE) $\hat{\varphi}_{\text{mle}}$.

The MLEs for $\varphi = (\sigma_\eta^2, \sigma_\varepsilon^2)^\top$ using `SsfFit` are computed as

```
> ll.mle = SsfFit(ll.start, nile.dat, "ll.mod", lower = 0)

Iteration  0 : objective =  638.2
Iteration  1 : objective =  638.2
Iteration  2 : objective =  638.2
...
Iteration  33 : objective =  633.5
RELATIVE FUNCTION CONVERGENCE

> class(ll.mle)

[1] "SsfFit"
```

In the call to `SsfFit`, the non-negative variance conditions $\sigma_\eta^2 \geq 0$ and $\sigma_\varepsilon^2 \geq 0$ are imposed in the estimation using the optional argument `lower = 0`. The result of `SsfFit` is a an object of class "`SsfFit`" with components

```
> names(ll.mle)
```

```
[1]  "parameters" "objective" "message" "grad.norm" "iterations"
[6]  "f.evals"    "g.evals"   "hessian" "scale"     "aux"
[11] "call"       "vcov"
```

The MLEs for σ_η^2 and σ_ε^2 are

```
> ll.mle

Log-likelihood:  -633.5
100 observations
Parameter Estimates:
  sig2.n  sig2.e
 1469.13 15098.6
```

The MLE for the signal-to-noise ratio is $\hat{q}_{mle} = \hat{\sigma}_\eta^2/\hat{\sigma}_\varepsilon^2 = 1469.13/15098.6 = 0.0973$. A summary of the fit, which gives estimated standard errors and t statistics, is given by

```
> summary(ll.mle)

Log-likelihood:  -633.465
100 observations
Parameters:
       Value Std. Error t value
sig2.n  1469       1292   1.137
sig2.e 15100       3146   4.799

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

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

```
> KalmanFil(nile.dat, ll.mod(ll.mle$parameters), task = "KFLIK")

Call:
KalmanFil(mY = nile.dat, ssf = ll.mod(ll.mle$parameters), task = "KFLIK")

Log-likelihood:  -633.4646
Prediction error variance:  1
Sample observations: 100

Standardized Innovations:
    Mean Std.Error
 -0.0832  0.9965
```

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

In the local level model, the variance parameter σ_ε^2 can be analytically concentrated out of the log-likelihood leaving the signal-to-noise ratio $q = \sigma_\eta^2/\sigma_\varepsilon^2$ as the only parameter to be estimated. 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 local level model, as a function of q only, is

```
> ll.modc = function(parm) {
+    parm = exp(2*parm)
+    ssf.llc = GetSsfStsm(irregular = 1, level = sqrt(parm))
+    CheckSsf(ssf.llc)
+ }
```

where we note that `parm = 0.5*log(q) = log(sqrt(q))` (and therefore `q = exp(2*parm)`) upon entry of the function.

By default, the function `GetSsfStsm` sets $\sigma_\varepsilon^2 = 1$ which is required for the computation of the concentrated log-likelihood function from (19). In the function `ll.modc`, to enforce a positive value for q, the concentrated log-likelihood is parameterized using $q = \exp(2 \cdot \psi)$ where ψ is unrestricted. To maximize the concentrated log-likelihood function (19) for the local level model with starting value $\psi = \log(q^{1/2}) = 0$, use `SsfFit` with `ll.modc` and set the optional argument `conc=T` :

```
> llc.start = 0
> names(llc.start) = "0.5*log(q)"
> llc.mle = SsfFit(llc.start, nile.dat, "ssf.ll.modc", conc = T)

Iteration  0 : objective =  637.079
Iteration  1 : objective =  633.52
Iteration  2 : objective =  633.469
Iteration  3 : objective =  633.465
Iteration  4 : objective =  633.465
Iteration  5 : objective =  633.465
RELATIVE FUNCTION CONVERGENCE

> summary(llc.mle)

Log-likelihood:  -633.465
100 observations
Parameters:
            Value Std. Error t value
0.5*log(q) -1.165     0.5061  -2.302

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

Notice that with the concentrated log-likelihood, the optimizer converges in only five iterations (previously it took 33 iterations). The values of the log-likelihood and the MLE for q are the same as found previously. To recover q use

```
> q.mle.c = exp(2*llc.mle$parameters)
> q.mle.c

 0.5*log(q)
 0.097306
```

The MLE for σ_ε^2 may be recovered by running the Kalman filter and computing the variance of the prediction errors from the concentrated log-likelihood:

```
> kf.llc = KalmanFil(nile.dat, ll.modc(llc.mle$parameters), task = "KFLIK")
> kf.ll$dVar
```

```
[1] 15098.5
```

Using $\hat{q}_{\text{mle}} = 0.097306$ and $\hat{\sigma}^2_{\varepsilon,\text{mle}} = 15099$, the MLE for σ_η^2 is given by $\hat{\sigma}^2_{\eta,\text{mle}} = \hat{q}_{\text{mle}} \times \hat{\sigma}^2_{\varepsilon,\text{mle}} = 1469$. One disadvantage of using the concentrated log-likelihood is the lack of an estimated standard error for $\hat{\sigma}^2_{\varepsilon,\text{mle}}$.

3. Affine term structure models

Traditionally the study of the term structure of interest rates focuses on either the cross sectional aspect of the yield curve, or the time series properties of the interest rate. Recently, researchers have utilized state space models and Kalman filtering techniques to estimate affine term structure models by combining both time series and cross sectional data. For simple models, the state space representation is often linear and Gaussian and analysis is straightforward. For more general models, the unobserved state variables generally influence the variance of the transition equation errors making the errors non-Gaussian. In these cases, non-standard state space methods are necessary.

Duffie and Kan (1996) show that many of the theoretical term structure models, such as the Vasicek (1977) model, Cox, Ingersoll, and Ross (1985) square root diffusion model, Longstaff and Schwartz (1992) two-factor model, and Chen (1996) three factor model, are special cases of the class of affine term structure models. The class of affine term structure models is one in which the yields to maturity on default-free pure discount bonds and the instantaneous interest rate are affine (constant plus linear term) functions of m unobservable state variables $\mathbf{X}_t$, which are assumed to follow an affine diffusion process

$$d\mathbf{X}_t = \mathbf{U}(\mathbf{X}_t; \mathbf{\Psi})dt + \mathbf{\Sigma}(\mathbf{X}_t; \mathbf{\Psi})d\mathbf{W}_t, \tag{20}$$

where $\mathbf{W}_t$ is an $m \times 1$ vector of independent Wiener processes; $\mathbf{\Psi}$ is a $p \times 1$ vector of model specific parameters; $\mathbf{U}(\cdot)$ and $\mathbf{\Sigma}(\cdot)$ are affine functions in $\mathbf{X}_t$ such that (20) has a unique solution. In general, the functions $\mathbf{U}(\cdot)$ and $\mathbf{\Sigma}(\cdot)$ can be obtained as the solution to some ordinary differential equations. Only in special cases are closed form solutions available. In this class of models, the price at time t of a default-free pure discount bond with time to maturity τ has the form

$$P_t(\mathbf{X}_t; \mathbf{\Psi}, \tau) = A(\mathbf{\Psi}, \tau) \exp\left\{-\mathbf{B}(\mathbf{\Psi}, \tau)^\top \mathbf{X}_t\right\} \tag{21}$$

where $A(\tau, \mathbf{\Psi})$ is a scalar function and $\mathbf{B}(\tau, \mathbf{\Psi})$ is an $m \times 1$ vector function. The time-t continuously compounded yield-to-maturity on a pure discount bond with time to maturity τ is defined as

$$Y_t(\mathbf{X}_t; \mathbf{\Psi}, \tau) = -\frac{\ln P_t(\mathbf{X}_t; \mathbf{\Psi}, \tau)}{\tau}, \tag{22}$$

which, using (21), has the affine form

$$Y_t(\mathbf{X}_t;\boldsymbol{\Psi},\tau) = -\frac{\ln A(\boldsymbol{\Psi},\tau)}{\tau} + \frac{\mathbf{B}(\boldsymbol{\Psi},\tau)^\top \mathbf{X}_t}{\tau} \tag{23}$$

State space representation

Although (23) dictates an exact relationship between the yield $Y_t(\tau)$ and the state variables $\mathbf{X}_t$, in econometric estimation it is usually treated as an approximation giving rise to the measurement equation

$$Y_t(\tau) = -\frac{\ln A(\boldsymbol{\Psi},\tau)}{\tau} + \frac{\mathbf{B}(\boldsymbol{\Psi},\tau)^\top \mathbf{X}_t}{\tau} + \epsilon_t(\tau), \tag{24}$$

where ϵ_t is a normally distributed measurement error with zero mean and variance σ_τ^2. For any time to maturity τ, the above equation can be naturally treated as the measurement equation of a state space model, with $\mathbf{X}_t$ being the unobserved state variable. To complete the state space representation, the transition equation for $\mathbf{X}_t$ over a discrete time interval h needs to be specified. Defining $\boldsymbol{\Phi}(\mathbf{X}_t;\boldsymbol{\Psi},h) = \text{var}(\mathbf{X}_{t+h}|\mathbf{X}_t)$, Duan and Simonato (1999) show that the transition equation for $\mathbf{X}_t$ has the form

$$\mathbf{X}_{t+h} = \mathbf{a}(\boldsymbol{\Psi},h) + b(\boldsymbol{\Psi},h)\mathbf{X}_t + \boldsymbol{\Phi}(\mathbf{X}_t;\boldsymbol{\Psi},h)^{1/2}\eta_{t+h}. \tag{25}$$

In this transition equation $\eta_t \sim$ iid $N(\mathbf{0},\mathbf{I}_m)$ and $\boldsymbol{\Phi}(\mathbf{X}_t;\boldsymbol{\Psi},h)^{1/2}$ represents the Cholesky factorization of $\boldsymbol{\Phi}(\mathbf{X}_t;\boldsymbol{\Psi},h)$.

In general, the state space model defined by (24) and (25) is non-Gaussian because the conditional variance of $\mathbf{X}_{t+h}$ in (25) depends on $\mathbf{X}_t$. Only for the special case in which $\boldsymbol{\Sigma}(\cdot)$ in (20) is not a function of $\mathbf{X}_t$, is the conditional variance term $\boldsymbol{\Phi}(\mathbf{X}_t;\boldsymbol{\Psi},h)$ also not a function of $\mathbf{X}_t$ and the state space model is Gaussian[2]. See Lund (1997) for a detailed discussion of the econometric issues associated with estimating affine term structure models using the Kalman filter. Although the quasi-maximum likelihood estimator of the model parameters based on the modified Kalman filter is inconsistent, the Monte Carlo results in Duan and Simonato (1999) and de Jong (1985) show that the bias is very small even for the moderately small samples likely to be encountered in practice.

Illustration

The data used for the following example are in the **S+FinMetrics** "`timeSeries`" `fama.bliss`, and consist of four monthly yield series over the period April, 1964 to December, 1997 for the U.S. Treasury debt securities with maturities of 3, 6, 12 and 60 months, respectively. This data was also used by Duan and Simonato (1999). All rates are continuously compounded rates expressed on an annual basis. These rates are displayed in Figure 9.

[2]To estimate the non-Gaussian state space model, Duan and Simonato (1999) modify the Kalman filter recursions to incorporate the presence of $\boldsymbol{\Phi}(\mathbf{X}_t;\boldsymbol{\Psi},h)$ in the conditional variance of η_{t+h}. The **S+FinMetrics/SsfPack** functions `KalmanFil` and `SsfLoglike` can be modified to accommodate this modification.

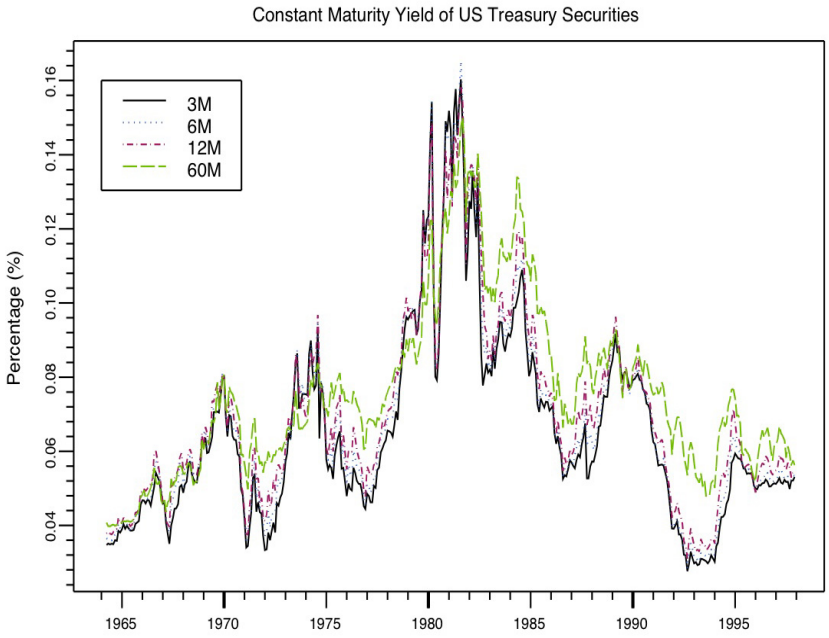

Figure 9: Monthly yields on U.S. treasury debt securities.

Estimation of Vasicek's model

In the Vasicek (1977) model, the state variable driving the term structure is the instantaneous (short) interest rate, r_t, and is assumed to follow the mean-reverting diffusion process

$$dr_t = \kappa(\theta - r_t)dt + \sigma dW_t, \ \kappa \geq 0, \sigma > 0 \tag{26}$$

where W_t is a scalar Wiener process, θ is the long-run average of the short rate, κ is a speed of adjustment parameter, and σ is the volatility of r_t. Duan and Simonato (1999) show that the functions $A(\cdot), B(\cdot), a(\cdot), b(\cdot)$ and $\Phi(\cdot)$ have the form

$$\begin{aligned}
\ln A(\boldsymbol{\Psi}, \tau) &= \gamma(B(\boldsymbol{\Psi}, \tau) - \tau) - \frac{\sigma^2 B^2(\boldsymbol{\Psi}, \tau)}{4\kappa}, \ B(\boldsymbol{\Psi}, \tau) = \frac{1}{\kappa}(1 - \exp(-\kappa\tau)) \\
\gamma &= \theta + \frac{\sigma\lambda}{\kappa} - \frac{\sigma^2}{2\kappa^2} \\
a(\boldsymbol{\Psi}, h) &= \theta(1 - \exp(-\kappa h)), \ b(\boldsymbol{\Psi}, h) = \exp(-\kappa h) \\
\Phi(X_t; \boldsymbol{\Psi}, h) &= \Phi(\boldsymbol{\Psi}, h) = \frac{\sigma^2}{2\kappa}(1 - \exp(-2\kappa h))
\end{aligned}$$

where λ is the risk premium parameter. The model parameters are $\boldsymbol{\Psi} = (\kappa, \theta, \sigma, \lambda)^\top$. Notice that for the Vasicek model, $\Phi(X_t; \boldsymbol{\Psi}, h) = \Phi(\boldsymbol{\Psi}, h)$ so that the state variable r_t does not influence the conditional variance of transition equation errors, the state space model is Gaussian.

The state space representation of the Vasicek model has system matrices

$$\delta = \begin{pmatrix} a(\mathbf{\Psi}, h) \\ -\ln A(\mathbf{\Psi}, \tau_1)/\tau_1 \\ \vdots \\ -\ln A(\mathbf{\Psi}, \tau_4)/\tau_4 \end{pmatrix}, \ \mathbf{\Phi} = \begin{pmatrix} b(\mathbf{\Psi}, h) \\ B(\mathbf{\Psi}, \tau_1)/\tau_1 \\ \vdots \\ B(\mathbf{\Psi}, \tau_4)/\tau_4 \end{pmatrix} \tag{27}$$
$$\mathbf{\Omega} = \text{diag}(\Phi(\mathbf{\Psi}, h), \sigma^2_{\tau_1}, \ldots, \sigma^2_{\tau_4})$$

and initial value matrix

$$\mathbf{\Sigma} = \begin{pmatrix} \theta \\ \sigma^2/2\kappa \end{pmatrix}$$

based on the stationary distribution of the short rate in (26). This a multivariate state space model as the response variable is the 4×1 vector $\mathbf{y}_t = (Y_t(\tau_1), Y_t(\tau_2), Y_t(\tau_3), Y_t(\tau_4))^\top$.

A function to compute the state space form of the Vasicek model for a given set of parameters $\mathbf{\Psi}$, number of yields $\tau_1, \ldots, \tau_N$, and sampling frequency h is

```
> vasicek.ssf = function(param, tau = NULL, freq = 1/52) {
+   if (length(param) < 5) stop("param must have length greater than 4.")
+   N = length(param) - 4
+   if (length(tau) != N) stop("Length of tau is inconsistent with param.")
+
+   Kappa = exp(param[1])
+   Theta = exp(param[2])
+   Sigma = exp(param[3])
+   Lamda = param[4]
+   Var   = exp(param[1:N+4])
+
+   Gamma = Theta + Sigma * Lamda / Kappa - Sigma^2 / (2 * Kappa^2)
+   B = (1 - exp(-Kappa * tau)) / Kappa
+   lnA = Gamma * (B - tau) - Sigma^2 * B^2 / (4 * Kappa)
+
+   a = Theta * (1 - exp(-Kappa * freq))
+   b = exp(-Kappa * freq)
+   Phi = (Sigma^2 / (2 * Kappa)) * (1 - exp(-2 * Kappa * freq))
+
+   mDelta = matrix(c(a, -lnA/tau), ncol = 1)
+   mPhi = matrix(c(b, B/tau), ncol = 1)
+   mOmega = diag(c(Phi, Var^2))
+
+   A0 = Theta
+   P0 = Sigma * Sigma / (2*Kappa)
+   mSigma = matrix(c(P0, A0), ncol=1)
+
+   ssf.mod = list(mDelta = mDelta, mPhi = mPhi, mOmega = mOmega,
+     mSigma = mSigma)
+   CheckSsf(ssf.mod)
+ }
```

The function starts by checking that `param` and `tau` are valid input. Then the parameters are extracted and – where required – non-negativity constraints are imposed, after which `Gamma`, `A`, and `B` are computed. Next, `a`, `b`, and `Phi` are computed, as well as the state space matrices `mDelta`, `mPhi`, and `mOmega`. Finally, the same initialization of `A0` and `P0` is used as in Duan and Simonato (1999) to set up matrix `mSigma`, and all state space matrices are returned.

The exponential transformation is used for those parameters that should be positive, and, since the data in `fama.bliss` are monthly, the default length of the discrete sampling interval, h, is set to 1/12.

Starting values for the parameters and the maturity specification for the yields are

```
> start.vasicek = c(log(0.1), log(0.06), log(0.02), 0.3, log(0.003),
+   log(0.001), log(0.003), log(0.01))
> names(start.vasicek) = c("ln.kappa", "ln.theta", "ln.sigma", "lamda",
+   "ln.sig.3M", "ln.sig.6M", "ln.sig.12M", "ln.sig.60M")
> start.tau = c(0.25, 0.5, 1, 5)
```

The maximum likelihood estimates for the parameters

$$\varphi = (\ln\kappa, \ln\theta, \ln\sigma, \lambda, \ln\sigma_{\tau_1}, \ln\sigma_{\tau_2}, \ln\sigma_{\tau_3}, \ln\sigma_{\tau_4})^\top$$

are obtained using `SsfFit`:

```
> ans.vasicek = SsfFit(start.vasicek, fama.bliss, vasicek.ssf,
+ tau = start.tau, freq = 1/12, trace = T,
+ control = nlminb.control(abs.tol = 1e-6, rel.tol = 1e-6,
+ x.tol = 1e-6, eval.max = 1000, iter.max = 500))

Iteration  0 : objective =  -6347.453
...
Iteration  37 : objective =  -6378.45
RELATIVE FUNCTION CONVERGENCE

> ssf.fit = vasicek.ssf(ans.vasicek$parameters, tau = start.tau, freq = 1/12)
```

The maximum likelihood estimates and asymptotic standard errors for the model parameters

$$\theta = (\kappa, \theta, \sigma, \lambda, \sigma_{\tau_1}, \sigma_{\tau_2}, \sigma_{\tau_3}, \sigma_{\tau_4})^\top$$

are computed using the delta method:

```
> ans.vasicek$parameters[-4] = exp(ans.vasicek$parameters[-4])
> names(ans.vasicek$parameters) = c("kappa", "theta", "sigma", "lamda",
+   "sig.3M", "sig.6M", "sig.12M", "sig.60M")
> dg = ans.vasicek$parameters; dg[4] = 1
> ans.vasicek$vcov = diag(dg) %*% ans.vasicek$vcov %*% diag(dg)
> summary(ans.vasicek)
```

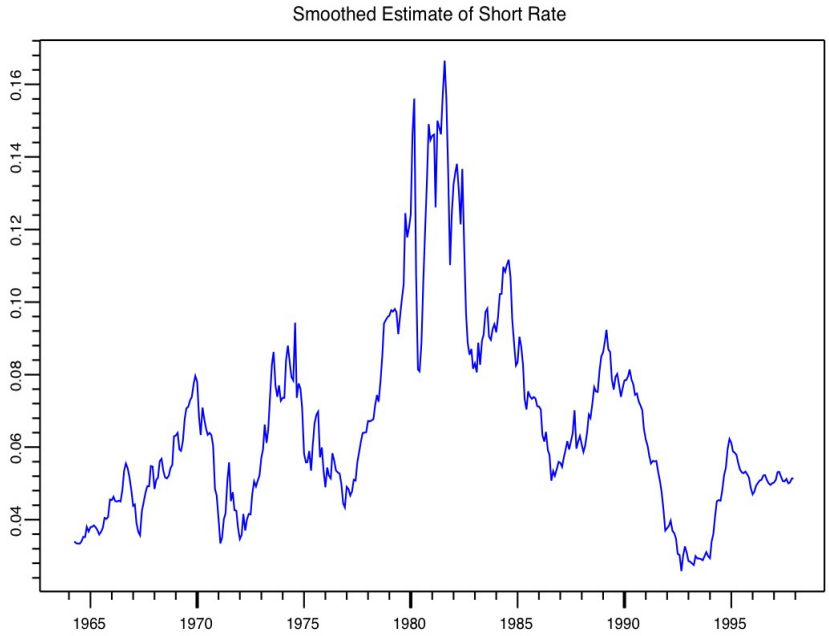

Figure 10: Smoothed estimate of short rate r_t from (26).

```
Log-likelihood:  6378.45
405 observations
Parameters:
             Value Std. Error t value
  kappa 0.11880000  0.0135800  8.7500
  theta 0.05739000  0.0267800  2.1430
  sigma 0.02138000  0.0007906 27.0500
  lamda 0.34770000  0.1493000  2.3280
 sig.3M 0.00283500  0.0001011 28.0500
 sig.6M 0.00002155  0.0005905  0.0365
sig.12M 0.00301600  0.0001083 27.8400
sig.60M 0.00990000  0.0003718 26.6300

Convergence:  RELATIVE FUNCTION CONVERGENCE
```

These results are almost identical to those reported by Duan and Simonato (1999). All parameters are significant at the 5% level except the measurement equation standard deviation for the six month maturity yield. The largest measurement equation error standard deviation is for the sixty month yield, indicating that the model has the poorest fit for this yield. The short rate is mean reverting since $\hat{\kappa} > 0$, and the long-run average short rate is $\hat{\theta} = 5.74\%$ per year. The estimated risk premium parameter, $\hat{\lambda} = 0.3477$, is positive indicating a positive risk premium for bond prices.

The smoothed estimates of the short-rate and the yields are computed using `SsfCondDens` with `task = "STSMO"`

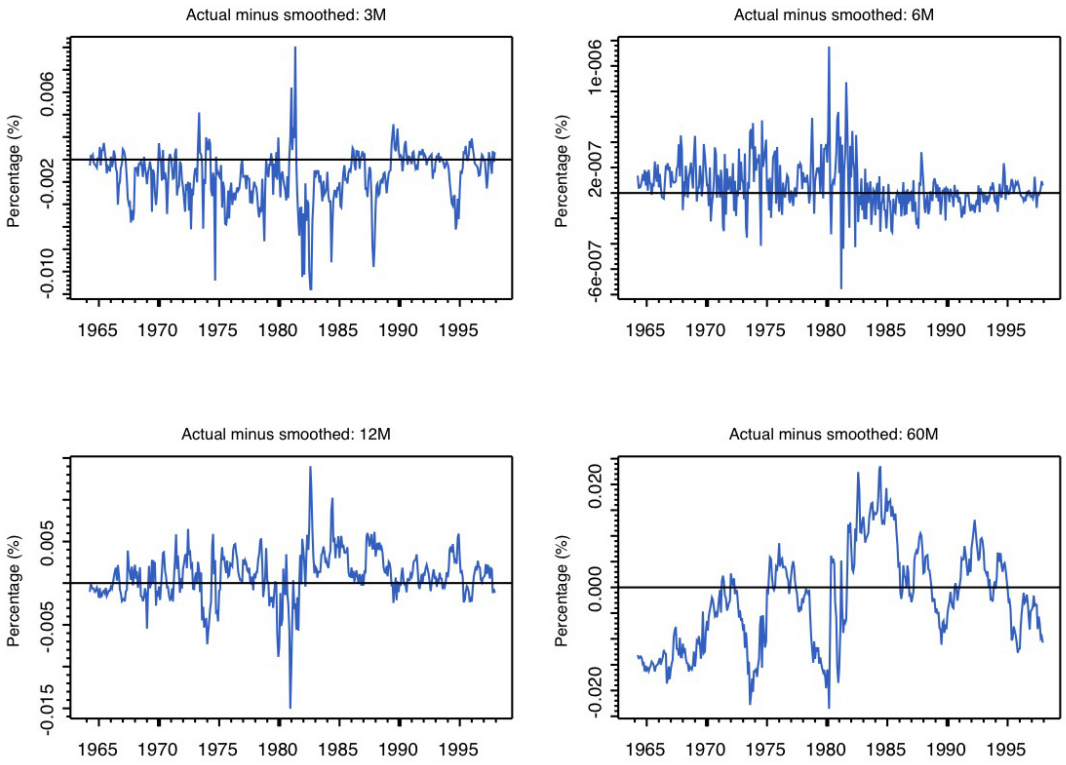

Figure 11: Smoothed estimates of bond yields from Vasicek term structure model.

```
> m.s = SsfCondDens(fama.bliss, ssf.fit)
> r.s = timeSeries(data = m.s$state, pos = m.s$positions)
> y.s = timeSeries(data = m.s$response, pos = m.s$positions)
```

Figure 10 gives the smoothed estimate of the instantaneous short rate r_t from (26). The differences between the actual and smoothed yield estimates are displayed in Figure 11. The model fits well on the short end of the yield curve but poorly on the long end.

As another check on the fit of the model, the presence of serial correlation in the standardized innovations is tested using the Box-Ljung modified Q-statistic (computed using the **S+FinMetrics** function `autocorTest`).

```
> autocorTest(KalmanFil(fama.bliss, ssf.fit)$std.innov)

Test for Autocorrelation: Ljung-Box

Null Hypothesis: no autocorrelation

Test Statistics:
                3M        6M        12M       60M
Test Stat   80.9471  282.4316  756.3304 3911.7736
  p.value    0.0000    0.0000    0.0000    0.0000
```

```
Dist. under Null: chi-square with 26 degrees of freedom
   Total Observ.: 405
```

The null of no serial correlation is easily rejected for the standardized innovations of all yields.

4. Conclusion

This paper presented two examples to illustrate the basics of state space modeling using **S+FinMetrics/SsfPack**. Full details of the use of **S+FinMetrics/SsfPack** are given in Zivot and Wang (2006), and several examples in macroeconomics and finance are provided in Zivot *et al.* (2004) and Tsay (2005). The state space modeling functions in **S+FinMetrics/SsfPack** are extremely flexible and powerful and can be used for a wide variety of univariate and multivariate linear Gaussian state space models. Users can build custom state space representations from scratch or they can use **S+FinMetrics/SsfPack** functions for specifying common state space models including ARIMA, seasonal ARIMA, regression, spline, and structural time series models. The state space representation in **S+FinMetrics/SsfPack** also allows for Markov switching in the system matrices as described in Kim (1994). Model fitting via the prediction error decomposition can be done using maximum likelihood or quasi-maximum likelihood, the latter being useful for linear but non-Gaussian state space models such as the log-normal stochastic volatility model. Because the state space algorithms are implemented in C, they are very fast. However, the internals of the algorithms are hidden from the user and cannot be modified which is a drawback to the user.

References

Chen L (1996). "Stochastic Mean and Stochastic Volatility - A Three Factor Model of the Term Structure of Interest Rates and Its Application to the Pricing of Interest Rate Derivatives." *Financial Markets, Institutions and Instruments*, **5**, 1–88.

Cox JC, Ingersoll JE, Ross SA (1985). "A Theory of the Term Structure of Interest Rates." *Econometrica*, **53**, 385–407.

de Jong F (1985). "Time Series and Cross-Section Information in Affine Term-Structure Models." *Journal of Business and Economic Statistics*, **18**, 300–314.

Duan JC, Simonato JG (1999). "Estimating Exponential-Affine Term Structure Models by Kalman Filter." *Review of Quantitative Finance and Accounting*, **13**, 111–135.

Duffie D, Kan R (1996). "A Yield-Factor Model of Interest Rates." *Mathematical Finance*, **6**, 379–406.

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

Kim C (1994). "Dynamic Linear Models with Markov-Switching." *Journal of Econometric*, **60**, 1–22.

Koopman SJ, Shephard N, Doornik JA (1999). "Statistical Algorithms for Models in State Space Using **SsfPack** 2.2." *Econometrics Journal*, **2**, 113–166.

Koopman SJ, Shephard N, Doornik JA (2008). ***SsfPack** 3.0: Statistical Algorithms for Models in State Space Form*. Timberlake Consultants, London.

Longstaff F, Schwartz E (1992). "Interest Rate Volatility and the Term Structure: A Two-Factor General Equilibrium Model." *Journal of Finance*, **47**, 1259–1282.

Lund J (1997). *Non-Linear Kalman Filtering Techniques for Term-Structure Models*. The Aarhus School of Business, Denmark.

TIBCO Software Inc (2010). *TIBCO Spotfire S+ Version 8.2*. Palo Alto, CA. URL http://spotfire.tibco.com/.

Tsay R (2005). *Analysis of Financial Time Series*. 2nd edition. John Wiley & Sons, New Jersey.

Vasicek O (1977). "An Equilibrium Characterization of the Term Structure." *Journal of Financial Economics*, **5**, 177–188.

Zivot E, Wang J (2006). *Modelling Financial Time Series with S-PLUS*. 2nd edition. Springer-Verlag, New York.

Zivot E, Wang J, Koopman SJ (2004). "State Space Models in Economics and Finance Using **SsfPack** in **S+FinMetrics**." In AC Harvey, SJ Koopman, N Shephard (eds.), *State Space and Unobserved Components Models*. Cambridge University Press, Cambridge.

Affiliation:

Eric W. Zivot
Department of Economics
University of Washington
Seattle, United States of America
E-mail: ezviot@u.washington.edu
URL: http://faculty.washington.edu/ezivot

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 41, Issue 5 *Submitted:* 2010-01-25
May 2011 *Accepted:* 2010-11-18