State Space Models in R

Article *in* Journal of Statistical Software · May 2011

DOI: 10.18637/jss.v041.i04 · Source: RePEc

2 authors:

Sonia Petrone
Università commerciale Luigi Bocconi
39 PUBLICATIONS **1,402** CITATIONS
SEE PROFILE

Giovanni Petris
University of Arkansas
22 PUBLICATIONS **862** CITATIONS
SEE PROFILE

Journal of Statistical Software

May 2011, Volume 41, Issue 4. *http://www.jstatsoft.org/*

State Space Models in R

Giovanni Petris
University of Arkansas

Sonia Petrone
Università Bocconi

Abstract

We give an overview of some of the software tools available in R, either as built-in functions or contributed packages, for the analysis of state space models. Several illustrative examples are included, covering constant and time-varying models for both univariate and multivariate time series. Maximum likelihood and Bayesian methods to obtain parameter estimates are considered.

Keywords: Kalman filter, state space models, unobserved components, software tools, R.

1. Introduction

There are currently more than 2000 contributed packages available on CRAN, the Comprehensive R Archive Network (R Development Core Team 2011). The requirements for a package to be made available on CRAN are minimal, essentially boiling down to the syntactic correctness of the code included in the package and the formal correctness in the documentation of the functions and data sets provided by the package. This has the positive effect of encouraging communication and sharing of ideas within the community of R users. On the other hand, the decentralized structure of the network of contributors makes it easy to end up with several unrelated packages from different authors, all meant to solve the same problem, to different degrees and using different, incompatible, user interfaces. As a consequence, it may be hard for the common user to navigate all the repository to find the right package for the analysis one is planning to carry out. State space models and Kalman filtering provide a typical illustration of this state of affairs. Functions like `RSiteSearch` or `findFn` in package **sos** (Graves, Dorai-Raj, and Francois 2010) can be used to navigate the many packages dealing to some extent with state space models. In this paper we will illustrate some of the R packages that a user can employ to analyze time series data by state space methods. As a rough guideline, we have opted to focus on packages that offer a complete suite of functions to perform filtering, smoothing, and forecasting for univariate and multivariate linear

This volume	West & Harrison
α_t	θ_t
Z_t	$F_t^\top$
ε_t	ν_t
H_t	V_t
T_t	G_{t+1}
$R_t\eta_t$	ω_{t+1}
R_t, Q_t	$W_{t+1}(= R_tQ_tR_t^\top)$
a_t	a_t
P_t	R_t
$a_{t\|t}$	m_t
$P_{t\|t}$	C_t

Table 1: Notation for state space models.

Gaussian state space models. To the best of our knowledge, the packages that satisfy this requirement at the time of writing are three: **dse** (Gilbert 2009), **dlm** (Petris 2010a), and **KFAS** (Helske 2010). We will treat **dlm** and **KFAS** in some detail in Section 2. Another package worth mentioning is **sspir** (Dethlefsen, Lundbye-Christensen, and Luther Christensen 2009). Package **sspir** includes functions for exponential family state space models and a nice formula interface to specify common models. On the other hand, its main limitations are that multivariate models are not covered and that, being written entirely in R, execution speed can be an issue, especially with long time series. The reader interested in a full description of package **sspir** can consult Dethlefsen and Lundbye-Christensen (2006).

The present paper is not meant to be an exhaustive review of all, or even some, of the R tools available for state space analysis. Rather, its purpose is to bring to the attention of the reader the most interesting among such tools by means of simple or moderately complex examples. The interested user can then learn more details by consulting the documentation of each package or other publications related to it. An interesting comparative review of the tools available in R for state space analysis is Tusell (2011).

There is no universally accepted notation for state space models, and this is unfortunately reflected in different names being used for the same quantities by the different contributed packages. However, two are the most commonly used notations in statistical and econometric applications. One, used in this collection, is that used in earlier works by Harvey (1989) and more recently revived in Durbin and Koopman (2001); this is the most common notation in the econometric literature. The other originated in the work by Harrison and Stevens (1976), was further popularized by West and Harrison (1997), and it is the one commonly adopted in the statistical (and Bayesian in particular) literature. The notation is also different for filtered and smoothed estimates and their variances. Table 1 shows the correspondence between the two notations.

In terms of approach and notation, package **dlm** falls in the tradition of West and Harrison (1997). It was developed to accompany Petris, Petrone, and Campagnoli (2009); see also Petris (2010b). The main focus is on Bayesian analysis, but maximum likelihood estimation of unknown parameters is also supported. Time-varying linear Gaussian state space models, both univariate and multivariate, can be analyzed using the functions provided in the package.

Task	Function	
	dlm	**KFAS**
Kalman filter	`dlmFilter`	`kf`
Kalman smoother	`dlmSmooth`	`ks`
Forecasts	`dlmForecast`	`forecast`
Likelihood	`dlmLL`	`kf`
ML estimates	`dlmMLE`	–

Table 2: Main functions in R packages **dlm** and **KFAS**.

The user interface is fairly well developed, providing for example extractor and replacement functions for any of the matrices defining the model. Furthermore, standard models can be easily set up using a suite of especially designed functions, and models can be combined by adding them together (e.g, local level plus trend) as well as by taking outer sums (e.g, producing a multivariate local level model from several univariate ones). Some of the most important functions are listed in Table 2. For more details the reader can consult Petris *et al.* (2009) and the package documentation, including the vignette[1].

Package **KFAS** falls, as far as notation and algorithms are concerned, in the tradition of Durbin and Koopman (2001). In addition to time-varying linear Gaussian state space models, both univariate and multivariate, univariate exponential family (Poisson and Binomial) linear state space models can be analyzed. Simulation-based Bayesian inference, via Markov chain Monte Carlo methods, although not explicity supported, can be implemented using the functions provided for the simulation smoother. On the negative side, the user interface is very basic and the documentation could be improved. Some of the most important functions of **KFAS** are listed in Table 2.

In what follows, up to Section 4, we use the term state space model to denote a linear Gaussian state space model. The paper is organized as follows. Section 2 introduces the main tools for the analysis of state space models in R, in the simple case of a local level model for the Nile data. Besides maximum likelihood estimation, Bayesian inference is illustrated too. Section 3 provides additional univariate and multivariate examples. Extensions to non-linear and non-Gaussian state space models are briefly discussed in the final Section 4.

2. The local level model

Probably the simplest nontrivial state space model is the local level model (Commandeur, Koopman, and Ooms 2011, Section 2.1). In this Section we illustrate how to work with this specific state space model in R. We first demonstrate the tools that are included in the standard distribution of R, and then move on to contributed packages **dlm** and **KFAS**.

2.1. The local level model in R

Structural time series models (Commandeur *et al.* 2011, Section 2.3) can be easily implemented in R through the function `StructTS` by B.D. Ripley, included in the base package

[1] A vignette is a PDF document included in a package, providing additional documentation for that package. Vignettes can be opened from R with the function `vignette`, as in `vignette("dlm", package = "dlm")`.

stats. This function, together with other tools for time series analysis included in base R, is described in detail in Ripley (2002). `StructTS` has the advantage of being of simple usage and quite reliable. It gives the main tools for fitting a structural model for a time series by maximum likelihood; the options `"level"`, `"trend"`, `"BSM"` are used to fit a local level model, a local linear trend, or a local trend with an additional seasonal component ("basic structural model"). The Nile river data are included in any standard distribution of R as a time series object (i.e., a vector containing the data together with information about start/end time and sampling frequency); a detailed description of the data is given in the help file, `?Nile`. Thus, there is no need to read them in and one can proceed to fit the model.

```
R> fitNile <- StructTS(Nile, "level")
R> fitNile

Call:
StructTS(x = Nile, type = "level")

Variances:
  level  epsilon
   1469    15099
```

The maximum likelihood estimates (MLEs) of the level and observation error variances, 1469 and 15099, respectively, are included in the output, `fitNile` as `fitNile$coef`. However, asymptotic standard errors are not provided. `StructTS` is quite reliable; yet, in the help file (`?StructTS`), Ripley underlines that "optimization of structural models is a lot harder than many of the references admit." The time series of the filtered estimates of the level, $a_{t|t} = \mathrm{E}(\alpha_t|y_{1:t})$, $t = 1, \ldots, n$, is obtained by applying the `fitted` method function[2] to the object `fitNile` (of class StructTS). For the local level model, these coincide with the one-step-ahead forecasts $\hat{y}_{t+1} = \mathrm{E}(y_{t+1}|y_{1:t})$. Similarly, smoothed estimates are obtained by applying the method function `tsSmooth`; however, standard errors of the estimates, and auxiliary residuals, are not provided. Filtered and smoothed estimates are shown in Figure 1.

```
R> plot(Nile, type = "o")
R> lines(fitted(fitNile), lty = "dashed", lwd = 2)
R> lines(tsSmooth(fitNile), lty = "dotted", lwd = 2)
```

The function `tsdiag` can be called on an object of class StructTS to obtain diagnostic plots based on the standardized one-step-ahead forecast errors.

Forecasts for structural time series, as objects of class StructTS, can be obtained by either the method function `predict` or `forecast` in package **forecast** (Hyndman 2011; Hyndman and Khandakar 2008). This package provides also a convenient plot method function for the resulting object of class forecast. Figure 2, obtained with the code below, shows the forecasted Nile river data until 1980, togeher with 50% and 90% probability intervals.

```
R> plot(forecast(fitNile, level = c(50, 90), h = 10), xlim = c(1950, 1980))
```

[2]We recall that objects in R may have a *class* attribute. Functions may have different methods to handle objects of different classes. A *method function* is a function designed to work on objects of a specific class.

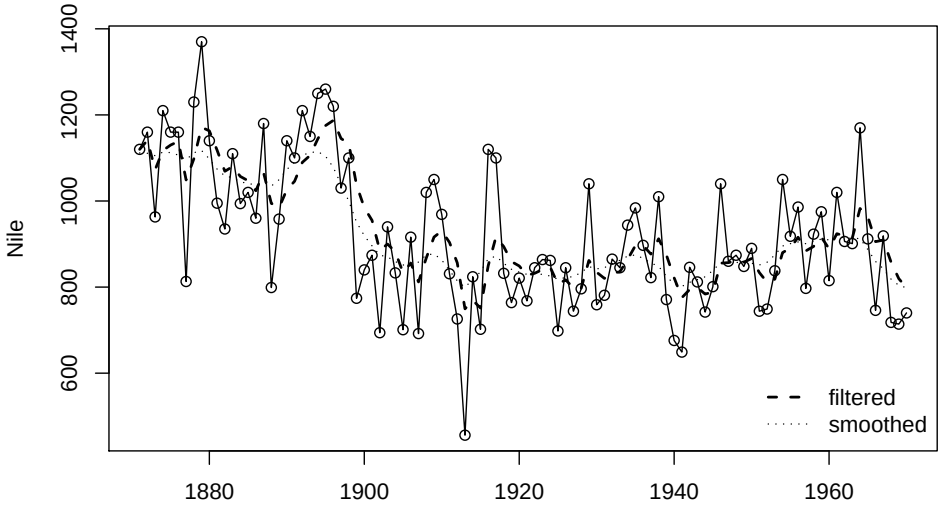

Figure 1: Nile river data, with filtered and smoothed estimates, obtained with `StructTS`.

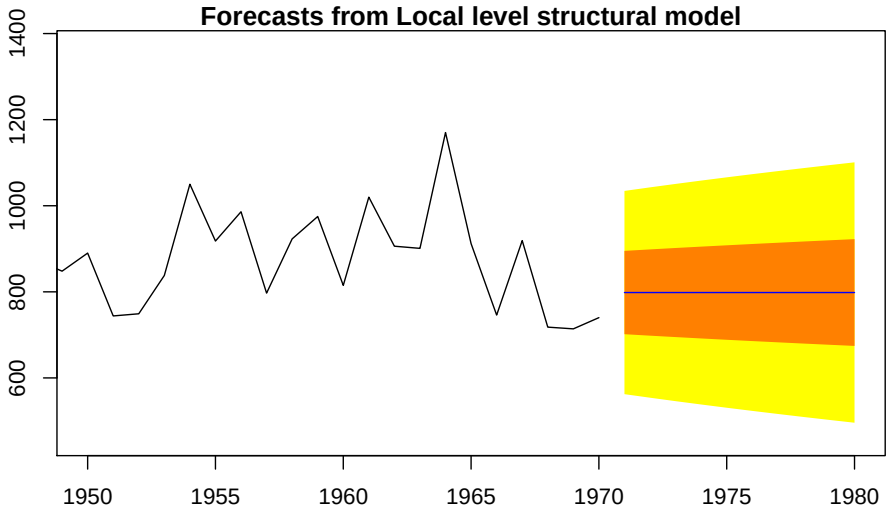

Figure 2: Forecasted Nile river level, with 50 and 90% probability intervals, obtained with `StructTS` and `forecast`.

2.2. The local level model with package dlm

A polynomial DLM (a local level model is a polynomial DLM of order 1, a local linear trend is a polynomial DLM of order 2), is easily defined in **dlm** through the function `dlmModPoly`. For example, a local level model, with known variances equal to 0.3 and 0.01, say, is obtained as

```
R> mod <- dlmModPoly(1, dV = 0.3, dW = 0.01)
```

The arguments `dV` and `dW` are used to specify the *diagonal* of the observation and evolution covariance matrices respectively (both 1×1 in this case). As it will be discussed in the next

section, further functions are provided in package **dlm** to create DLMs of standard types, such as, besides the basic structural components, a dynamic regression model or an ARMA model.

MLEs are computed in package **dlm** by the function `dlmMLE`, which is a wrapper to `optim`, the general-purpose optimizer included in package **stats**. The main arguments are the data, a starting value for the unknown (vector) parameter, and a function that sets up a dlm object using the unknown parameter. This is internally combined with a call to `dlmLL`, which evaluates the negative log likelihood, and passed to `optim`, which performs the actual optimization. For a local level model with unknown variances a build function can be defined as

```
R> buildNile <- function(theta) {
+    dlmModPoly(order = 1, dV = theta[1], dW = theta[2])
+  }
```

The MLEs are then obtained by calling `dlmMLE` as follows:

```
R> fit <- dlmMLE(Nile, parm = c(100, 2), buildNile, lower = rep(1e-4, 2))
```

Note the argument `lower`, used to specify a lower bound for the possible value of the parameters, which is passed directly to `optim`. The output `fit` is simply the one produced by `optim`, so the user familiar with that function can easily interpret the different components of the returned list. In particular, the component `par` contains the value of the (vector) argument at which the optimum is achieved, while the component `convergence` is set to zero if and only if the algorithm converged successfully. As a minimal check, we encourage the user to make sure that `convergence` is actually zero in the output from `dlmMLE`. If this is the case, the user can proceed to set up the estimated model and to examine the MLE of the parameter. To set up the model it is advisable to use the very same build function used in the call to `dlmMLE`, as shown in the code below.

```
R> modNile <- buildNile(fit$par)
R> drop(V(modNile))

[1] 15099.79

R> drop(W(modNile))

[1] 1468.438
```

The inverse of the Hessian matrix of the negative loglikelihood function evaluated at the MLEs is, by standard maximum likelihood theory, an estimate of the asymptotic variance matrix of the maximum likelihood estimators. The default optimization algorithm used by `optim` when called inside `dlmMLE` is BFGS (for details, see Monahan 2001, Section 8.7), which can also return an estimated Hessian at the optimum. However, such an estimate is generally unreliable, resulting sometimes in a matrix which is not even positive definite. In order to obtain a numerically accurate evaluation of the Hessian at the MLEs, it is better to use the function `hessian` in package **numDeriv** (Gilbert 2011). From the Hessian, standard errors can then be estimated in the usual way.

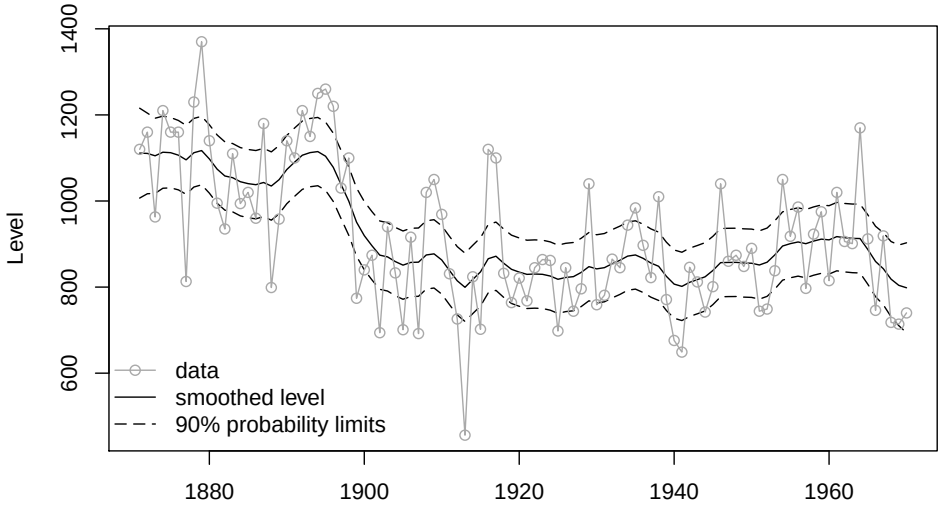

Figure 3: Smoothed Nile river level, with 90% probability limits.

```
R> hs <- hessian(function(x) dlmLL(Nile, buildNile(x)), fit$par)
R> all(eigen(hs, only.values = TRUE)$values > 0)

[1] TRUE

R> aVar <- solve(hs)
R> sqrt(diag(aVar))

[1] 3146.003 1280.180
```

The reader will certainly notice the large uncertainty associated with the MLEs, summarized by the two standard errors. The asymptotic correlation between the estimates of the two variances is -0.61. We can conclude that the likelihood function does not allow to accurately estimate the two variances.

The Kalman smoother can be run on the estimated model using the function `dlmSmooth`.

```
R> smoothNile <- dlmSmooth(Nile, modNile)
```

The output consists of a list with three components: the first, named `s`, is the time series of smoothed estimates of the state, from time $t = 0$ to time $t = n$, the second and third, `U.S` and `D.S`, contain the singular value decomposition of the corresponding smoothing variances. The variances themselves can be obtained using the utility function `dlmSvd2var` and employed, for example, to evaluate probability limits for the state vector, as illustrated in the code below.

```
R> hwidth <- qnorm(0.05, lower = FALSE) *
+    sqrt(unlist(dlmSvd2var(smoothNile$U.S, smoothNile$D.S)))
R> sm <- cbind(smoothNile$s, as.vector(smoothNile$s) + hwidth %o% c(-1, 1))
```

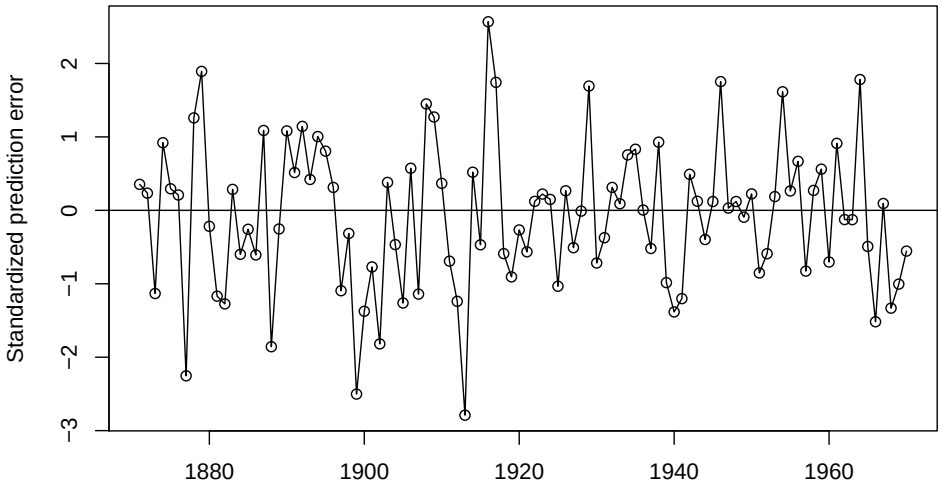

Figure 4: Standardized one-step prediction errors for the Nile river level.

The object `sm` is now a three-variate time series whose components are the smoothing state estimates, lower 90% probability limits, and upper 90% probability limits. A plot of the smoothing estimates is provided in Figure 3.

Kalman filter can be run by calling the function `dlmFilter`. This produces an object of class dlmFiltered, a list containing, most notably, predicted states and filtered estimates of state vectors together with their variances. Method functions `residuals` and `tsdiag` can be used to compute the one-step forecast errors and plot several diagnostics based on them. For example, Figure 4 was obtained with the following code.

```
R> filterNile <- dlmFilter(Nile, modNile)
R> plot(residuals(filterNile, sd = FALSE), type = "o",
+    ylab = "Standardized prediction error")
R> abline(h = 0)
```

Smoothed observation and state disturbances can be easily computed from the output of `dlmSmooth`. However, in package **dlm** there is no easy way to compute their variances, which makes it hard to obtain auxiliary residuals. Simulating from the model (using `dlmForecast`) to compute a Monte Carlo estimate of the smoothed disturbance variances is an option.

Forecasting future state vectors or observations is easily done in **dlm** using the function `dlmForecast`. For example, the code below was used to produce Figure 5.

```
R> foreNile <- dlmForecast(filterNile, nAhead = 10)
R> attach(foreNile)
R> hwidth <- qnorm(0.25, lower = FALSE) * sqrt(unlist(Q))
R> fore <- cbind(f, as.vector(f) + hwidth %o% c(-1, 1))
R> rg <- range(c(fore, window(Nile, start = c(1951, 1))))
R> plot(fore, type = "o", pch = 16, plot.type = "s", lty = c(1, 3, 3),
+    ylab = "Nile level", xlab = "", xlim = c(1951, 1980), ylim = rg)
R> lines(window(Nile, start = c(1951, 1)), type = 'o')
```

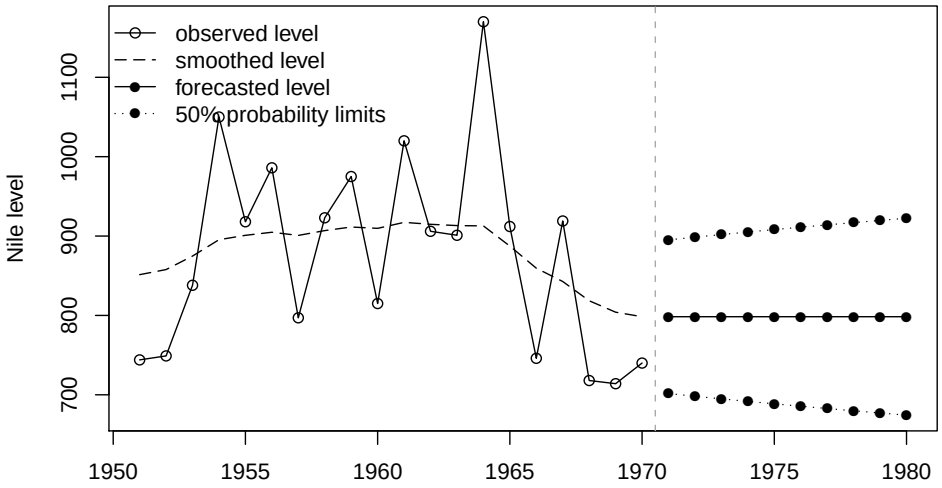

Figure 5: Nile river level: forecasts and 50% probability limits.

```
R> lines(window(smoothNile$s, start = c(1951,1)), lty = 5)
R> abline(v = mean(c(time(f)[1], tail(time(Nile), 1))),
+    lty = "dashed", col = "darkgrey")
R> legend("topleft", lty = c(1, 5, 1, 3), pch = c(1, NA, 16, 16), bty = "n",
+    legend = c("observed level", "smoothed level", "forecasted level",
+      "50% probability limits"))
R> detach(foreNile)
```

In addition to point forecasts and forecast variances, `dlmForecast` can provide a sample from the joint forecast distribution of future states and observations. This feature can also be used to simulate data from a specific model, by calling `dlmForecast` on an object of class dlm instead of one of class dlmFiltered.

2.3. The local level model with package KFAS

The basic function for Kalman filtering, `kf`, returns, among other things, the value of the log likelihood of a specified model, for a particular data set. This makes it easy to write a function that returns the negative log likelihood and pass it to `optim`.

```
R> logLik <- function(theta) {
+    lik <- kf(yt = Nile, Zt = 1, Tt = 1, Rt = 1, Ht = theta[1],
+      Qt = theta[2], a1 = 0, P1 = 1e7)
+    return(-lik$lik)
+  }
R> fit <- optim(par = c(100, 2), fn = logLik, lower = rep(1e-4, 2))
```

To allow comparisons with the other packages, we set the initial variance to a finite, large value, even if `kf` can deal with diffuse initial conditions. The Hessian matrix can be evaluated using `numDeriv::hessian`, as we did in Section 2.2.

Smoothed values of the level, together with their variances, can be obtained calling the two functions `kf` (Kalman filter) and `ks` (Kalman smoother) in this order. The plot shown in Figure 3 can be obtained using **KFAS** with the code below.

```
R> filterNile <- kf(yt = Nile, Zt = 1, Tt = 1, Rt = 1, Ht = fit$par[1],
+    Qt = fit$par[2], a1 = 0, P1 = 1e7)
R> smoothNile <- ks(filterNile)
R> attach(smoothNile)
R> hwidth <- qnorm(0.05, lower = FALSE) * sqrt(drop(Vt))
R> sm <- cbind(drop(ahat), as.vector(ahat) + hwidth %o% c(-1, 1))
R> sm <- ts(sm, start = start(Nile))
R> plot(sm, plot.type = "s", type = "l", lty = c(1, 5, 5),
+    ylab = "Level", xlab = "", ylim = range(Nile))
R> lines(Nile, type = "o", col = "darkgrey")
R> legend("bottomleft", col = c("darkgrey", rep("black", 2)),
+    lty = c(1, 1, 5), pch = c(1, NA, NA), bty = "n", legend =
+    c("data", "smoothed level", "90% probability limits"))
R> detach(smoothNile)
```

There is no predefined function in package **KFAS** to compute standardized prediction errors, and one-step-ahead forecasts for the observations are not returned by `kf`: in order to compute standardized prediction errors the user needs to evaluate them based on the arrays containing the a_t, P_t, and H_t. In the univariate case only, the v_t and their variances F_t happen to coincide with the components `vtuni` and `Ftuni` of the list returned by `kf`. Therefore, for the Nile level one can easily compute standardized prediction errors as follows.

```
R> residNile <- drop(filterNile$vtuni / sqrt(filterNile$Ftuni))
```

A time series plot of `residNile` would look like Figure 4 (except for the time labeling, as time series attributes are lost in the output of `kf`).

Forecasts of the Nile level up to and including 1980, together with 50% probability limits can be computed with package **KFAS** as follows.

```
R> foreNile <- forecast(filterNile, fc = 9)
R> attach(foreNile)
R> hwidth <- qnorm(0.25, lower = FALSE) * sqrt(drop(Pt.fc))
R> fore <- ts(cbind(drop(at.fc), drop(at.fc) + hwidth %o% c(-1, 1)),
+    start = 1 + end(Nile)[1])
R> rg <- range(c(fore, window(Nile, start = c(1951, 1))))
R> detach(foreNile)
```

2.4. Bayesian inference for the local level model

The standard practice of fitting a state-space model by maximum likelihood, plugging the MLEs into the model, and considering state estimates and forecasts, together with their variances, obtained from the Kalman filter does not account in any way for the uncertainty about the true values of the model parameters. This may be an issue, since precise parameter

estimates in state-space models are the exception rather than the rule, even in simple models like the local level model – recall the large standard errors of the variances obtained in Section 2.2 for the Nile river data. Moreover, maximum likelihood estimation in state space models can be numerically hard. The Bayesian approach offers a natural way of dealing with parameter uncertainty in a state space model. Given the observations $y_{1:n}$, Bayesian inference on the unknown parameter vector, say ψ, and on the states history $\alpha_{0:n}$, is simply solved, in principle, by computing their posterior distribution, $\pi(\psi, \alpha_{0:n}|y_{1:n})$. However, except for a few particular cases with conjugate priors, this distribution is not computable in closed form, and one has to resort to approximation techniques, the most popular being Markov chain Monte Carlo (MCMC). The difficulty in implementing a user friendly software for Bayesian analysis in state space models is that MCMC is prior- and model- specific, so that it is not possible to provide an algorithm that works in all cases. However, auxiliary tools can be provided. A Gibbs sampler for $\pi(\psi, \alpha_{0:n}|y_{1:n})$ can be obtained by iteratively sampling from the full conditional distribution of the states given the parameters and the data, and from the full conditional of the parameters, given the states and the data. Although the latter is problem specific, sampling from $\pi(\alpha_{0:n}|\psi, y_{1:n})$ can be generally implemented by the Forward-Filtering Backward-Sampling algorithm (FFBS), see Früwirth-Schnatter (1994); Carter and Kohn (1994); Shephard (1994), or the simulation smoother, see de Jong and Shephard (1995); Durbin and Koopman (2002).

Several tools for Bayesian smoothing in state space models are provided in the package **dlm**. The function `dlmBSample` implements FFBS (the simulation smoother is not available in **dlm**). The complete Gibbs sampling is provided for the basic case of a univariate state space model with unknown observation variance and unknown, diagonal, evolution covariance matrix, with independent Inverse-Gamma priors (function `dlmGibbsDIG`). Tools for the analysis of MCMC output are also provided.

Let us illustrate Bayesian estimation and smoothing with package **dlm** for the local level model, applied to the Nile data. Here the unknown parameters are the error variances $\psi = (\sigma^2_\epsilon, \sigma^2_\xi)$. We assume independent Inverse-Gamma prior distributions, i.e., $(\sigma^2_\epsilon)^{-1} \sim \text{Gamma}(\alpha_\epsilon, \beta_\epsilon)$ and $(\sigma^2_\xi)^{-1} \sim \text{Gamma}(\alpha_\xi, \beta_\xi)$, where $\alpha.$ and $\beta.$ are the shape and the rate parameters of the gamma distribution. A Gibbs sampling from the joint posterior $\pi(\sigma^2_\epsilon, \sigma^2_\xi, \alpha_{0:n}|y_{1:n})$ (for more details, see Petris *et al.* 2009, Section 4.5.1) is implemented by the function `dlmGibbsDIG`. The user must be aware that the function may be slow, and has been included in the package mostly for didactical purposes.

```
R> set.seed(123)
R> gibbsOut <- dlmGibbsDIG(Nile, mod = dlmModPoly(1), shape.y = 0.1,
+    rate.y = 0.1, shape.theta = 0.1, rate.theta = 0.1, n.sample = 10000,
+    thin = 9)
```

The code above runs 100000 MCMC iterations, saving the results of one out of every 10 iterations (`thin=9`) in order to reduce the autocorrelation in the saved MCMC samples. Package **dlm** includes some tools to facilitate basic convergence diagnostics of the MCMC output. Figure 6 shows running sample means and empirical autocorrelation functions for the MCMC samples of the variances, after discarding the first 1000 draws as burn-in; the plots are obtained with the code below.

```
R> burn <- 1:1000
```

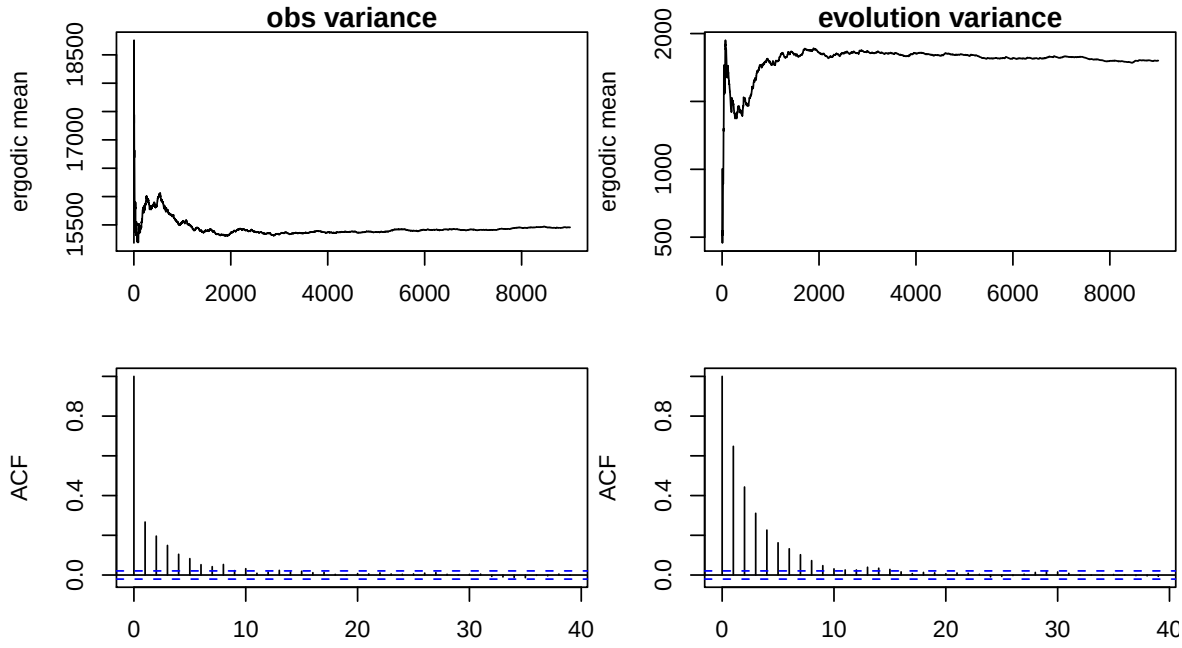

Figure 6: Nile data. Diagnostic plots for MCMC output: Running sample means and autocorrelation function for the MCMC samples of σ_ϵ^2 (first column) and of σ_ξ^2.

```
R> attach(gibbsOut)
R> ts.plot(ergMean(dV[-burn]), ylab = "sample mean", xlab = "iterations",
+    main = "obs variance")
R> ts.plot(ergMean(dW[-burn]), ylab = "sample mean", xlab = "iterations",
+    main = "evolution variance")
R> acf(dV[-burn])
R> acf(dW[-burn])
```

Figure 7 shows the MCMC estimates of the posterior densities of the observation variance and of the evolution variance, respectively. The last panel plots the MCMC samples from their joint posterior density; a high correlation is evident, reflecting a rather slow mixing of the Gibbs sampler.

```
R> plot(density(dV[-burn]), xlim = c(2000, 34000), ylab = "", main = "")
R> hist(dV[-burn], prob = TRUE, add = TRUE)
R> curve(dgamma(1/x, shape = 0.1, rate = 0.1) / x^2, lty = "dashed",
+    add = TRUE)
R> plot(density(dW[-burn]), ylab = "", xlim = c(0, 16000),  main = "")
R> hist(dW[-burn], prob = TRUE, add = TRUE)
R> curve(dgamma(1/x, shape = 0.1, rate = 0.1) / x^2, lty = "dashed",
+    add = TRUE)
R> plot(dV[-burn], dW[-burn], pch = ".", cex = 1.5, ylab = "")
```

The Bayesian estimates of the unknown variances, with respect to quadratic loss, are given by their posterior expectations, whose MCMC estimate, together with Monte Carlo standard

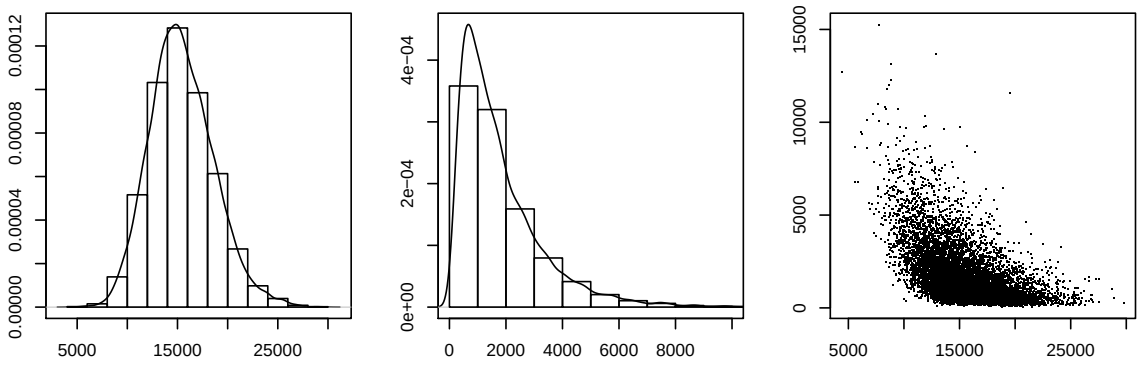

Figure 7: Nile data. Posterior density of the evolution variance and of the observation variance (MCMC approximation). The dashed line in the first two panels is the prior density. The third panel shows MCMC samples from the joint posterior of σ_ϵ^2 and σ_ξ^2.

errors, are given below. The code also shows how to obtain 95% posterior probability intervals for the unknown parameters.

```
R> mcmcMean(dV[-burn])

  dV[-burn]
  15456.6
(   57.6)

R> mcmcMean(dW[-burn])

  dW[-burn]
  1800.6
(  37.9)

R> quantile(dV[-burn], c(0.025, 0.975))

     2.5%     97.5%
9759.739 22318.972

R> quantile(dW[-burn], c(0.025, 0.975))

     2.5%     97.5%
237.0324 5805.1629
```

Note the big difference between the 95% probability interval (237, 5805) for σ_ξ^2 (W, in the notation used by **dlm**) and the 95% confidence interval (-1040, 3977) that can be obtained from the MLE (found in Section 2.2) using its standard error and Normal asymptotic theory. The most plausible explanation is that the Normal asymptotic distribution of the MLE is not a good approximation of its actual sampling distribution. To support this conjecture, one can

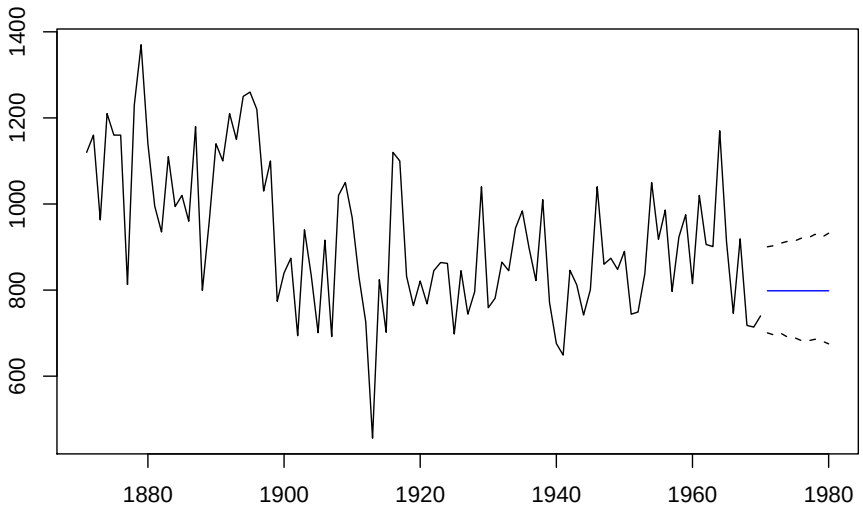

Figure 8: Nile data. Bayesian forecasts and 50% probability limits.

observe that both the posterior distribution and the sampling distribution of the MLE are asymptotically Normal, but Figure 7 clearly shows that the actual posterior distribution of σ_ξ^2 is far from Normal, suggesting that this may be the case also for the sampling distribution of the MLE of σ_ξ^2. A quantile-quantile plot (not shown) of the MCMC draws of a logarithmic transformation of σ_ξ^2 shows, on the other hand, a very good agreement with a Normal distribution, suggesting that, in this example, a logarithmic transformation of the parameter σ_ξ^2 could provide more reliable confidence intervals based on asymptotic maximum likelihood theory.

k-step-ahead state and observation forecasts can be obtained by sampling from the conditional distributions of $(\alpha_{n+k}, y_{t+k})|\{\alpha_n, y_{1:n}\}$. As an illustration, we provide below the 50% probability intervals for the observation forecasts, plotted in Figure 8.

```
R> lastTheta <- theta[length(Nile) + 1, , ]
R> levelSim <- matrix(0, nr = 10, nc = 10000)
R> for (it in 1:10000) {
+    innovSim <- rnorm(10, sd = sqrt(dW[it]))
+    levelSim[, it] <- cumsum(innovSim) + lastTheta[it]
+  }
R> ySim <- matrix(0, nr = 10, nc = 10000)
R> for (it in 1:10000) {
+    innov <- rnorm(10, sd = sqrt(dV[it]))
+    ySim[, it] <- innov + levelSim[, it]
+  }
R> yInts <- apply(ySim, 1, function(x) quantile(x, c(0.25, 0.75)))
```

For Bayesian on-line filtering and forecasting, however, MCMC is not efficient, since the Gibbs sampling has to be run all over again as new data become available. Closed form solutions are possible with some restrictive assumptions and conjugate priors, usually jointly with *discount factors* techniques (West and Harrison 1997, Section 6.3). R functions for

Function	Model
`dlmModARMA`	ARMA process in state space form
`dlmModPoly`	nth order polynomial state space model
`dlmModReg`	Linear regression
`dlmModSeas`	Periodic – Seasonal factors
`dlmModTrig`	Periodic – Trigonometric form

Table 3: Creator functions for special models with **dlm**.

implementing Bayesian conjugate analysis with discount factors that rely on the package **dlm** are available at `http://definetti.uark.edu/~gpetris/dlm/`, but they are not included in the package itself. Bayesian on-line analysis for state-space models is generally implemented through sequential Monte Carlo algorithms, see for example Doucet, de Freitas, and Gordon (2001), Liu (2001), Migon, Gamerman, Lopes, and Ferreira (2005), Gamerman and Lopes (2006), Prado and West (2010). The contributed package **SMC** (Goswami 2008) provides basic building blocks to implement sequential Monte Carlo simulation techniques for state space models.

3. Additional univariate and multivariate examples

3.1. Intervention variables

In addition to `dlmModPoly`, illustrated in Section 2.2, **dlm** provides other functions to create state space models of standard types, which are summarized in Table 3. With the exception of `dlmModARMA`, which handles also the multivariate case, the other creator functions are limited to the case of univariate observations.

As an illustration, with a time-varying state space model, let us consider again the Nile river data, and suppose one wants to take into account a structural break in the level of Nile river flow, following the construction of the Ashwan dam in 1899. This can be done by introducing an intervention variable x_t that is zero until 1898 and one thereafter. As discussed in Commandeur *et al.* (2011, Section 2.2), a local level model with an intervention variable x_t is defined as $y_t = \mu_t + \lambda_t x_t + \epsilon_t$, with state vector $\alpha_t = (\mu_t, \lambda_t)^\top$ and state equation $\mu_t = \mu_{t-1} + \xi_t$, $\lambda_t = \lambda_{t-1} + \zeta_t$, where $(\xi_t, \zeta_t)^\top \sim \mathcal{N}(0, \mathrm{diag}(\sigma_\xi^2, \sigma_\zeta^2))$. If $\sigma_\zeta^2 = 0$, the regression coefficient λ_t is in fact constant; we assume so in the code that follows. This model is a dynamic linear regression, that can be estimated with package **dlm** as shown below. Note that, when accounting for the drop in the river flow following the dam construction, the estimated evolution variance of the level is essentially zero. Figure 9 shows the resulting smoothed estimates of the level; the structural break in 1899 is evident.

```
R> x <- matrix(c(rep(0, 27), rep(1, length(Nile) - 27)), ncol = 1)
R> modNileReg <- dlmModReg(x, dW = c(1, 0))
R> buildFun <- function(theta) {
+    V(modNileReg) <- exp(theta[1])
+    diag(W(modNileReg))[1] <- exp(theta[2])
+    return(modNileReg)
+  }
```

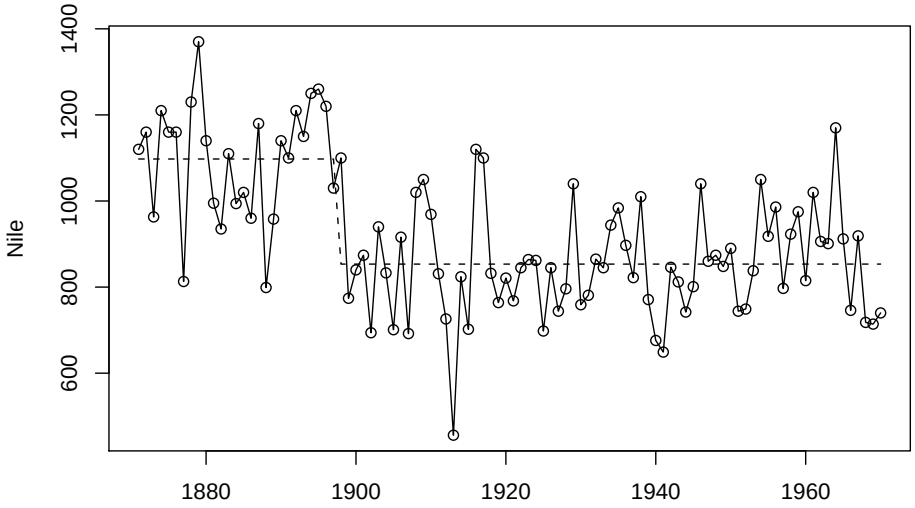

Figure 9: Nile river data: smoothed level, with a local level model and intervention variable. The evolution variance of the level is practically zero.

```
R> fit <- dlmMLE(Nile, parm = rep(0, 2), build = buildFun)
R> modNileReg <- buildFun(fit$par)
R> drop(V(modNileReg))

[1] 16928.26

R> W(modNileReg)[1]

[1] 0.0001575277

R> modSmooth <- dlmSmooth(Nile, mod = modNileReg)
R> plot(Nile, type = "o")
R> lines(ts(modSmooth$s[-1, 1] + modSmooth$s[-1, 2] * x, start = 1871),
+    lty = 2)
```

3.2. Structural time series and model composition

In many applications, the actual observation at time t is conveniently represented as the sum of independent components, i.e., $y_t = y_t^{(1)} + \cdots + y_t^{(k)}$, where the ith one is defined by a state space model with its own state vector and system and covariance matrices, and describes a specific aspect of the time series (Commandeur *et al.* 2011, Section 2.3). If the series $y_t^{(i)}$ $(i = 1, \ldots, k)$ is described by a state space model with matrices $Z_t^{(i)}, H_t^{(i)}, T_t^{(i)}, R_t^{(i)}, Q_t^{(i)}$, it is easy to see that the resulting series y_t is also described by a state space model, with matrices

$$Z_t = \left[Z_t^{(1)} \mid \ldots \mid Z_t^{(k)} \right],$$
$$H_t = H_t^{(1)} + \cdots + H_t^{(k)},$$

$$T_t = \begin{bmatrix} T_t^{(1)} & & \\ & \ddots & \\ & & T_t^{(k)} \end{bmatrix},$$

$$R_t = \begin{bmatrix} R_t^{(1)} & & \\ & \ddots & \\ & & R_t^{(k)} \end{bmatrix},$$

$$Q_t = \begin{bmatrix} Q_t^{(1)} & & \\ & \ddots & \\ & & Q_t^{(k)} \end{bmatrix}.$$

Combining component state space models can be easily implemented in package **dlm**. As said above, **dlm** provides functions to create state space models of standard types. Moreover, a method function is provided for the generic `+` for objects of class `dlm`, so that more elaborate state space models can be easily specified by "summing" basic ones as described above. For example, the code `dlmModPoly(order = 1) + dlmModSeas(frequency = 12)` can be used to specify the DLM representation of a local level model plus a seasonal component for monthly data.

As a simple illustration, let us consider the well known series of quarterly UK gas consumption from 1960 to 1986, available in R as `UKgas`. Suppose that we want to describe the series, on a logarithmic scale, by a state space model containing a quarterly seasonal component and a local linear trend, in the form of an integrated random walk (i.e., in the notation of expression (4) in Commandeur *et al.* (2011), $\sigma_\xi^2 = 0$). The model, and the MLE of the unknown parameters, can be obtained by **dlm** as follows.

```
R> lGas <- log(UKgas)
R> dlmGas <- dlmModPoly() + dlmModSeas(4)
R> buildFun <- function(x) {
+    diag(W(dlmGas))[2:3] <- exp(x[1:2])
+    V(dlmGas) <- exp(x[3])
+    return(dlmGas)
+  }
R> fit <- dlmMLE(lGas, parm = rep(0, 3), build = buildFun)
R> dlmGas <- buildFun(fit$par)
R> drop(V(dlmGas))

[1] 0.1206739

R> diag(W(dlmGas))[2:3]

[1] 0.06258571 0.02775090
```

The smoothed estimates, based on the fitted model, provide a decomposition of the data into a smooth trend plus a stochastic seasonal component, plus measurement error. These components are plotted in Figure 10.

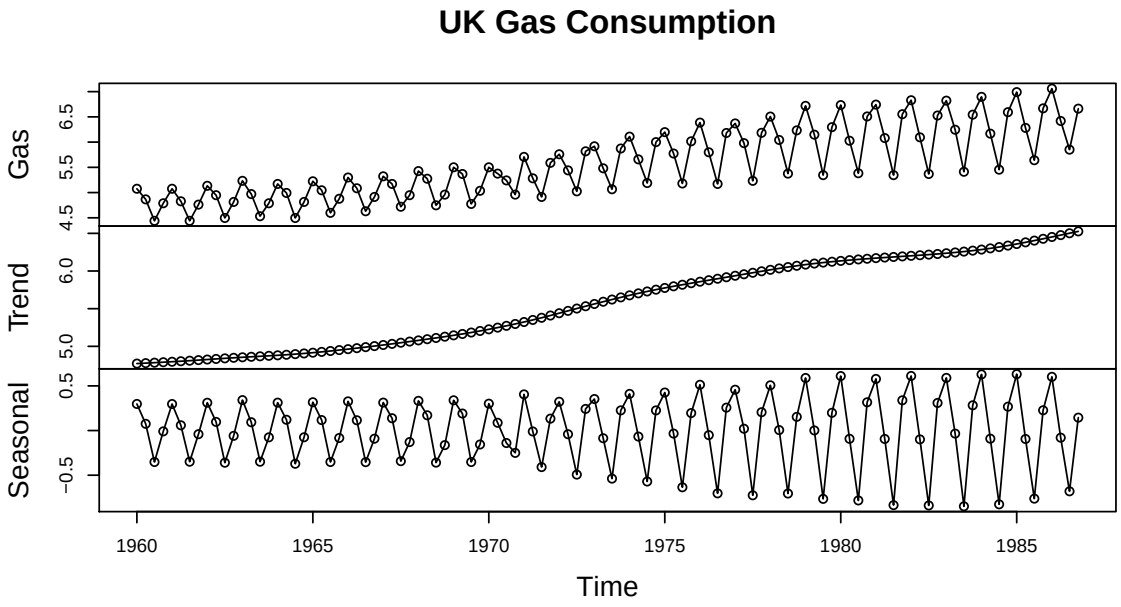

Figure 10: UK gas data. Smoothed estimates of trend and stochastic seasonal component.

```
R> gasSmooth <- dlmSmooth(lGas, mod = dlmGas)
R> x <- cbind(lGas, dropFirst(gasSmooth$s[, c(1, 3)]))
R> colnames(x) <- c("Gas", "Trend", "Seasonal")
R> plot(x, type = "o", main = "UK Gas Consumption")
```

3.3. Multivariate models

Multivariate state space models can be analyzed in R using package **dlm** and package **KFAS**. Package **dlm** provides a convenient function (`dlmSum`, or its alias `%+%`) to combine several univariate models into a comprehensive multivariate model. Suppose that the series $y_t^{(i)}$ $(i = 1, \ldots, k)$ are described by state space models as in Section 3.2. One can write $y_t = (y_t^{(1)}, \ldots, y_t^{(k)})$. Assuming that the components $y_t^{(i)}$ are independent, the k-variate series y_t is described by a state space model with

$$Z_t = \begin{bmatrix} Z_t^{(1)} & & \\ & \ddots & \\ & & Z_t^{(k)} \end{bmatrix},$$

$$H_t = \begin{bmatrix} H_t^{(1)} & & \\ & \ddots & \\ & & H_t^{(k)} \end{bmatrix},$$

and where T_t, R_t, Q_t are the same as in Section 3.2. We can interpret this operation as an *outer sum* of the models representing $y_t^{(1)}, \ldots, y_t^{(k)}$. Using this notion of outer sum we see that,

for example, seemingly unrelated time series equations (SUTSE) models can be effortlessly set up in **dlm** by "adding" p copies of the corresponding univariate model. For instance, a multivariate local level model for three time series series can be defined as

```
R> modUni <- dlmModPoly(1)
R> modUni %+% modUni %+% modUni
```

Clearly, the assumption of independence among the components will be unrealistic in most interesting applications. Correlation among the $y_t^{(i)}$ can be introduced by changing the values of the observation and system covariance matrices (using the replacement function `V()` and `W()`) or, as is more often the case, by estimating those matrices. For higher-dimensional time series, it may be convenient to use `dlmSum` together with `do.call`. The following produces a multivariate local level model for 13 time series at once.

```
R> do.call(dlmSum, rep(list(modUni), 13))
```

In the rest of the section we will consider a dynamic version of the capital asset pricing model (CAPM), illustrating how to fit it and estimate the unobservable betas using **KFAS**. Simplifying a bit, the CAPM assumes that the excess return (relative to a risk-free asset) on an asset in any time period is proportional, on average, to the excess return on a portfolio representing the entire market. The proportionality constants will be referred to as the *betas*. Considering several assets at once, and allowing the betas to vary over time, one can set up the following multivariate dynamic regression model:

$$\alpha_t = (\beta_{1t}, \ldots, \beta_{mt})^\top, \quad T_t = R_t = I_m, \quad Z_t = x_t I_m,$$
$$H_t = \Sigma_\epsilon, \quad Q_t = \Sigma_\beta,$$

where x_t is the market excess return in period t. Note that here Σ_ϵ and Σ_β are $m \times m$ variance matrices accounting for correlated observation errors and correlated changes of the betas, respectively. The data we will use for this example are monthly returns on four stocks (Mobil, IBM, Weyer, and Citicorp) from January 1978 to December 1987, together with the 30-day Treasury Bill as a proxy for the risk-free asset. The value-weighted average returns for all the stocks listed at the New York and American Stock Exchanges will be used as a proxy for the market returns. The data, originally used in Berndt (1991), are now available on the internet.

```
R> tmp <- ts(read.table("http://shazam.econ.ubc.ca/intro/P.txt",
+    header = TRUE), start = c(1978, 1), frequency = 12) * 100
R> y <- tmp[, 1:4] - tmp[, "RKFREE"]
R> colnames(y) <- colnames(tmp)[1:4]
R> market <- tmp[, "MARKET"] - tmp[, "RKFREE"]
R> rm("tmp")
R> m <- NCOL(y)
```

To estimate the unknown parameters, we are going to parameterize each variance matrix in terms of its log Choleski decomposition. The code below shows how to find the MLEs of Σ_ϵ and Σ_β, assuming a diffuse prior for α_1 with infinite variances in P_1. The first two lines are just a convenient way to set up, using the standard functions `sapply` and `seq_along`, the three-way $(m \times m \times n)$ array required by `kf` containing the matrix Z_t at each time $t = 1, \ldots, n$.

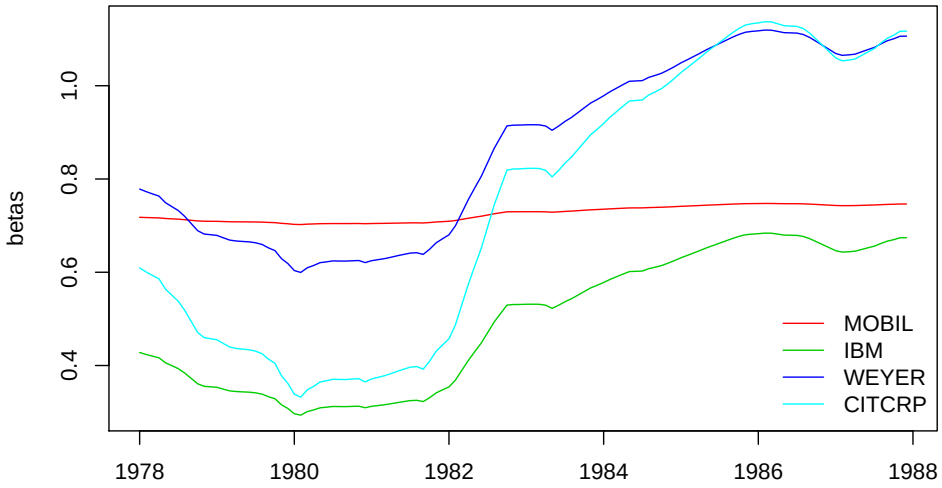

Figure 11: Multivariate dynamic capital asset pricing model: betas for four assets.

```
R> Zt <- sapply(seq_along(market), function(i) market[i] %x% diag(m))
R> dim(Zt) <- c(m, m, length(market))
R> Rt <- diag(nr = m)
R> logLik <- function(theta) {
+    a <- diag(exp(0.5 * theta[1:m]), nr = m)
+    a[upper.tri(a)] <- theta[(m + 1):k]
+    Ht <- crossprod(a)
+    a <- diag(exp(0.5 * theta[1:m + k]), nr = m)
+    a[upper.tri(a)] <- theta[-(1:(k + m))]
+    Qt <- crossprod(a)
+    lik <- kf(yt = t(y), Zt = Zt, Tt = diag(nr = m), Rt = Rt, Ht = Ht,
+      Qt = Qt, a1 = rep(0, m), P1 = matrix(0, m, m),
+      P1inf = diag(rep(1, m)), optcal = c(FALSE, FALSE, FALSE, FALSE))
+    return(-lik$lik)
+  }
R> fit <- optim(par = rep(0, 2 * k), fn = logLik, method = "BFGS",
+    control = list(maxit = 500))
R> fit$conv

[1] 0
```

With the MLEs of the unknown parameters one can compute the smoothing estimates of the betas (Figure 11), as the following code illustrates.

```
R> theta <- fit$par
R> a <- diag(exp(0.5 * theta[1:m]), nr = m)
R> a[upper.tri(a)] <- theta[(m+1):k]
R> Ht <- crossprod(a)
R> a <- diag(exp(0.5 * theta[1:m + k]), nr = m)
```

```
R> a[upper.tri(a)] <- theta[-(1:(k + m))]
R> Qt <- crossprod(a)
R> smoothCAPM <- ks(kf(yt = t(y), Zt = Zt, Tt = diag(nr = m), Rt = Rt,
+    Ht = Ht, Qt = Qt, a1 = rep(0, m), P1 = matrix(0, m, m),
+    P1inf = diag(rep(1, m))))
R> betas <- ts(t(smoothCAPM$ahat), start = start(market),
+    freq = frequency(market))
```

As one can see, while the beta of Mobil did not change much, those of the other three stocks did change considerably over the period considered, suggesting that a dynamic framework may be more appropriate than a static one for this type of analysis.

4. Conclusions

Without aiming at being exhaustive, we have tried to guide the reader through the main tools and contributed packages available in R for the analysis of Gaussian, linear state space models. Continuing in R leads to extensions for non-linear, non-Gaussian state space models, and for discretely evolving, or hidden Markov, models. In fact, while the user can find reliable functions and packages for linear Gaussian state space processes, the choice appears more limited for other classes of state space models. It is worth however to underline that the open source philosophy of R allows to easily take existing packages and extend those for one's purposes, when a full package is not available.

There are in fact many ways of extending the standard linear Gaussian state space model discussed in the previous sections. One possibility is to keep the general form of the state evolution unchanged, and assume that the observation depends on the current state in a nonlinear and/or non-Gaussian way. The two most popular classes of models obtained by relaxing the assumptions in this way (discussed in Commandeur *et al.* 2011, Section 5) are stochastic volatility (SV) models and dynamic generalized linear models (DGLM). Support for both in R is still fairly limited. Package **sspir** allows to perform extended Kalman filter and smoother for univariate DGLMs having Poisson or Binomial distribution of the observations. There are functions for approximate Kalman filtering and smoothing for these types of models also in package **KFAS**; unfortunately, when trying a very simple Poisson example the returned approximate loglikelihood resulted consistently to be `NaN`, making it impossible to estimate model parameters.

Many SV models can be analyzed with the help of package **sde** (Iacus 2009, 2008), although the focus of the package is on continuous time models. A Bayesian analysis of SV can be done using the integrated nested Laplace approximation (INLA); see Rue, Martino, and Chopin (2009). The R package **INLA** Rue and Martino (2011) (not on CRAN) can be used to perform INLA in R.

Hidden Markov models (HMMs) are another important class of state space models, where the state α_t follows a Markov chain with a finite number of states, while the observations can have any distribution depending on the current state, although Gaussian and other exponential family distributions are the most commonly used in practice. Book-length treatments of HMMs can be found in Frühwirth-Schnatter (2006); MacDonald and Zucchini (2009). Several packages are available on CRAN for the analysis of HMMs. In addition, one can use package **repeated** (Lindsey 2009), not available on CRAN. Contributed packages are also available for

the analysis of state space models in more specific applied contexts; one recent example is the package **Stem** (Cameletti 2009), for spatio-temporal models.

While the analysis of general state space models with R is beyond the scope of this work, we hope that the overview of the basic Gaussian, linear case provided here can be helpful also as a starting point for the reader interested in analysing more general and complex models with R.

Acknowledgments

We would like to thank an anonymous referee and the special volume editors for their comments and suggestions. They all helped us to communicate our thoughts more clearly in the final version of this paper. We would also like to thank the special volume editors for extending the invitation to contribute to this volume of the Journal of Statistical Software on state space models.

References

Berndt RE (1991). *The Practice of Econometrics*. Addison-Wesley.

Cameletti M (2009). ***Stem**: Spatio-Temporal Models in R*. R package version 1.0, URL http://CRAN.R-project.org/package=Stem.

Carter CK, Kohn R (1994). "On Gibbs Sampling for State Space Models." *Biometrika*, **81**, 541–553.

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

de Jong P, Shephard N (1995). "The Simulation Smoother for Time Series Models." *Biometrika*, **82**(2), 339–350.

Dethlefsen C, Lundbye-Christensen S (2006). "Formulating State Space Models in R with Focus on Longitudinal Regression Models." *Journal of Statistical Software*, **16**(1), 1–15. ISSN 1548-7660. URL http://www.jstatsoft.org/v16/i01/.

Dethlefsen C, Lundbye-Christensen S, Luther Christensen A (2009). ***sspir**: State Space Models in R*. R package version 0.2.8, URL http://CRAN.R-project.org/package=sspir.

Doucet A, de Freitas N, Gordon N (eds.) (2001). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York.

Durbin J, Koopman SJ (2001). *Time Series Analysis by State Space Methods*. Number 24 in Oxford Statistical Science Series. Oxford University Press, Oxford.

Durbin J, Koopman SJ (2002). "A Simple and Efficient Simulation Smoother for State Space Time Series Analysis." *Biometrika*, **89**(3), 603–615.

Frühwirth-Schnatter S (2006). *Finite Mixture and Markov Switching Models*. Springer-Verlag, New York.

Früwirth-Schnatter S (1994). "Data Augmentation and Dynamic Linear Models." *Journal of Time Series Analysis*, **15**, 183–202.

Gamerman D, Lopes HF (2006). *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Texts in Statistical Science, 2nd edition. Chapman & Hall/CRC Press, London.

Gilbert P (2011). ***numDeriv**: Accurate Numerical Derivatives*. R package version 2010.11-1, URL http://CRAN.R-project.org/package=numDeriv.

Gilbert PD (2009). *Brief User's Guide: Dynamic Systems Estimation*. URL http://CRAN.R-project.org/package=dse.

Goswami G (2008). ***SMC**: Sequential Monte Carlo (SMC) Algorithm*. R package version 1.0, URL http://CRAN.R-project.org/package=SMC.

Graves S, Dorai-Raj S, Francois R (2010). ***sos**: SOS*. R package version 1.3-0, URL http://CRAN.R-project.org/package=sos.

Harrison PJ, Stevens CF (1976). "Bayesian Forecasting." *Journal of the Royal Statistical Society B*, **38**, 205–247.

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

Helske J (2010). ***KFAS**: Kalman filter and Smoothers for Exponential Family State Space Models*. R package version 0.5.1, URL http://CRAN.R-project.org/package=KFAS.

Hyndman RJ (2011). ***forecast**: Forecasting Functions for Time Series*. R package version 2.17, URL http://CRAN.R-project.org/package=forecast.

Hyndman RJ, Khandakar Y (2008). "Automatic Time Series Forecasting: The **forecast** Package for R." *Journal of Statistical Software*, **27**(3), 1–22. URL http://www.jstatsoft.org/v27/i03/.

Iacus SM (2008). *Simulation and Inference for Stochastic Differential Equations – With R Examples*. Springer-Verlag, New York.

Iacus SM (2009). ***sde**: Simulation and Inference for Stochastic Differential Equations*. R package version 2.0.10, URL http://CRAN.R-project.org/package=sde.

Lindsey JK (2009). ***repeated**: Non-Normal Repeated Measurements Models*. R package version 1.0, URL http://www.commanster.eu/rcode.html.

Liu J (2001). *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag, New York.

MacDonald IL, Zucchini W (2009). *Hidden Markov Models for Time Series: An Introduction Using R*. Monographs on Statistics & Applied Probability. Chapman & Hall/CRC, London.

Migon HS, Gamerman D, Lopes HF, Ferreira MAR (2005). "Bayesian Dynamic Models." In D Day, C Rao (eds.), *Handbook of Statistics*, volume 25, chapter 19, pp. 553–588. Elsevier B.V.

Monahan J (ed.) (2001). *Numerical Methods of Statistics*. Cambridge University Press, Cambridge.

Petris G (2010a). ***dlm**: Bayesian and Likelihood Analysis of Dynamic Linear Models*. R package version 1.1-2, URL http://CRAN.R-project.org/package=dlm.

Petris G (2010b). "An R Package for Dynamic Linear Models." *Journal of Statistical Software*, **36**(12), 1–16. URL http://www.jstatsoft.org/v36/i12/.

Petris G, Petrone S, Campagnoli P (2009). *Dynamic Linear Models with R*. Springer-Verlag, New York.

Prado R, West M (2010). *Time Series: Modeling, Computation, and Inference*. Chapman & Hall/CRC Press, London.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Ripley BD (2002). "Time Series in R 1.5.0." *R News*, **2**(2), 2–7. URL http://CRAN.R-project.org/doc/Rnews/.

Rue H, Martino S (2011). ***INLA**: Functions Which Allow to Perform a Full Bayesian Analysis of Structured Additive Models Using Integrated Nested Laplace Approximaxion*. R package version 0.0, revision ef2284e0b8, URL http://www.R-INLA.org/.

Rue H, Martino S, Chopin N (2009). "Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations." *Journal of the Royal Statistical Society B*, **71**, 319–392.

Shephard N (1994). "Partial Non-Gaussian State Space Models." *Biometrika*, **81**, 115–131.

Tusell F (2011). "Kalman Filtering in R." *Journal of Statistical Software*, **39**(2), 1–27. URL http://www.jstatsoft.org/v39/i02/.

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

Affiliation:

Giovanni Petris
Department of Mathematical Sciences
University of Arkansas
Fayetteville, AR 72701, United States of America
E-mail: GPetris@uark.edu

Sonia Petrone
Department of Decision Sciences
Università Bocconi
Milano, Italy
E-mail: sonia.petrone@unibocconi.it

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 41, Issue 4 *Submitted:* 2010-01-26
May 2011 *Accepted:* 2010-11-10