Journal of Statistical Software
March 2011, Volume 39, Issue 2. *http://www.jstatsoft.org/*

Kalman Filtering in R

Fernando Tusell
University of the Basque Country

Abstract

Support in R for state space estimation via Kalman filtering was limited to one package, until fairly recently. In the last five years, the situation has changed with no less than four additional packages offering general implementations of the Kalman filter, including in some cases smoothing, simulation smoothing and other functionality. This paper reviews some of the offerings in R to help the prospective user to make an informed choice.

Keywords: state space models, Kalman filter, time series, R.

1. Introduction

The Kalman filter is an important algorithm, for which relatively little support existed in R (R Development Core Team 2010) up until fairly recently. Perhaps one of the reasons is the (deceptive) simplicity of the algorithm, which makes it easy for any prospective user to throw in his/her own quick implementation.

While direct transcription of the equations of the Kalman filter as they appear in many engineering or time series books may be sufficient for some applications, an all-around implementation requires more complex coding. In Section 2, we provide a short overview of available algorithms. It is against this background that we discuss in Section 3 the particular choices made by four packages offering fairly general Kalman filtering in R, with some mention of functions in other packages which cater to particular needs.

Kalman filtering is a large topic. In the sequel we focus on linear Gaussian models and their estimation, which is what the packages we review offer in common (and the foundation on which most anything else rests). Other functionalities present in some of the packages examined include filtering and estimation of non-Gaussian models, simulation and disturbance smoothing and functions to help with the Bayesian analysis of dynamic linear models, etc. none of which are assessed.

2. Kalman filter algorithms

We shall consider a fairly general state-space model specification, sufficient for the purpose of the discussion to follow in Section 3, even if not the most comprehensive. The notation follows Harvey (1989). Let

$$\begin{aligned} \boldsymbol{\alpha}_t &= \boldsymbol{c}_t + \boldsymbol{T}_t \boldsymbol{\alpha}_{t-1} + \boldsymbol{R}_t \boldsymbol{\eta}_t \quad (1) \\ \boldsymbol{y}_t &= \boldsymbol{d}_t + \boldsymbol{Z}_t \boldsymbol{\alpha}_t + \boldsymbol{\epsilon}_t \quad (2) \end{aligned}$$

where $\boldsymbol{\eta}_t \sim N(\boldsymbol{0}, \boldsymbol{Q}_t)$ and $\boldsymbol{\epsilon}_t \sim N(\boldsymbol{0}, \boldsymbol{H}_t)$. The state equation (1) describes the dynamics of the state vector $\boldsymbol{\alpha}_t$, driven by deterministic ($\boldsymbol{c}_t$) and stochastic ($\boldsymbol{\eta}_t$) inputs. The observation (or measurement) equation links the observed response $\boldsymbol{y}_t$ with the unobserved state vector, with noise $\boldsymbol{\epsilon}_t$ and (possibly) deterministic inputs $\boldsymbol{d}_t$. Matrices $\boldsymbol{T}_t$, $\boldsymbol{R}_t$, $\boldsymbol{Z}_t$, $\boldsymbol{Q}_t$ and $\boldsymbol{H}_t$ may depend on a vector of parameters, $\boldsymbol{\theta}$, and be time varying or constant over time. The noises $\boldsymbol{\eta}_t$ and $\boldsymbol{\epsilon}_t$ are assumed serially and also mutually uncorrelated, i.e., $\mathsf{E}[\boldsymbol{\eta}_t \boldsymbol{\epsilon}_s^\top] = \boldsymbol{0}$ for all t, s. The last assumption and the gaussianity of $\boldsymbol{\eta}_t$ and $\boldsymbol{\epsilon}_t$ can be dispensed with; see e.g., Anderson and Moore (1979).

2.1. The Kalman filter.

The Kalman filter equations, with slight notational variations, are standard in any textbook: see, e.g., Anderson and Moore (1979), Simon (2006), Durbin and Koopman (2001), Grewal and Andrews (2001), West and Harrison (1997) or Shumway and Stoffer (2006), to name only a few. We reproduce those equations here, however, as repeated reference is made to them in the sequel. Define

$$\begin{aligned} \boldsymbol{a}_{t-1} &= \mathsf{E}[\boldsymbol{\alpha}_{t-1} | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{t-1}] \quad (3) \\ \boldsymbol{P}_{t-1} &= \mathsf{E}[(\boldsymbol{\alpha}_{t-1} - \boldsymbol{a}_{t-1})(\boldsymbol{\alpha}_{t-1} - \boldsymbol{a}_{t-1})^\top] \quad ; \quad (4) \end{aligned}$$

estimates of the state vector and its covariance matrix at time t with information available at time $t-1$, $\boldsymbol{a}_{t|t-1}$ and $\boldsymbol{P}_{t|t-1}$ respectively, are given by the *time update* equations

$$\begin{aligned} \boldsymbol{a}_{t|t-1} &= \boldsymbol{T}_t \boldsymbol{a}_{t-1} + \boldsymbol{c}_t \quad (5) \\ \boldsymbol{P}_{t|t-1} &= \boldsymbol{T}_t \boldsymbol{P}_{t-1} \boldsymbol{T}_t^\top + \boldsymbol{R}_t \boldsymbol{Q}_t \boldsymbol{R}_t^\top . \quad (6) \end{aligned}$$

Let $\boldsymbol{F}_t = \boldsymbol{Z}_t \boldsymbol{P}_{t|t-1} \boldsymbol{Z}_t{}^\top + \boldsymbol{H}_t$. If a new observation is available at time t, then $\boldsymbol{a}_{t|t-1}$ and $\boldsymbol{P}_{t|t-1}$ can be updated with the *measurement update* equations

$$\begin{aligned} \boldsymbol{a}_t &= \boldsymbol{a}_{t|t-1} + \boldsymbol{P}_{t|t-1} \boldsymbol{Z}_t^\top \boldsymbol{F}_t^{-1} (\boldsymbol{y}_t - \boldsymbol{Z}_t \boldsymbol{a}_{t|t-1} - \boldsymbol{d}_t) \quad (7) \\ \boldsymbol{P}_t &= \boldsymbol{P}_{t|t-1} - \boldsymbol{P}_{t|t-1} \boldsymbol{Z}_t \boldsymbol{F}_t^{-1} \boldsymbol{Z}_t^\top \boldsymbol{P}_{t|t-1} . \quad (8) \end{aligned}$$

Equations (5)–(6) and (7)–(8) taken together make up the Kalman filter. Substituting (5) in (7) and (6) in (8), a single set of equations linking $\boldsymbol{a}_{t-1}$ and $\boldsymbol{P}_{t-1}$ to $\boldsymbol{a}_t$ and $\boldsymbol{P}_t$ can be obtained. In order to start the iteration we need initial values of $\boldsymbol{a}_{-1}$ and $\boldsymbol{P}_{-1}$ (or $\boldsymbol{a}_{0|-1}$ and $\boldsymbol{P}_{0|-1}$).

The filter equations (6) and (8) propagate the covariance matrix of the state, and are said to define a covariance filter (CF). Equivalent equations can be written which propagate the matrix $\boldsymbol{P}_t^{-1}$, giving an information filter (IF). Information filters require in general more computational effort. One possible advantage is that they provide a natural way to specify complete uncertainty about the initial value of a component of the state: we can set the corresponding diagonal term in the information matrix to zero. With a covariance filter, we have to set the corresponding variance in $\boldsymbol{P}_{0|-1}$ to a "large" number, or else use exact diffuse initialization, an option described below.

Direct transcription of the equations making up the Kalman filter into computer code is straightforward. It was soon noticed, though, that the resulting programs suffered from numerical instability; see for instance Bucy and Joseph (1968). In particular, buildup of floating point errors in equation (8) may eventually yield non symmetric or non positive definite $\boldsymbol{P}_t$ matrices. An alternative to equation (8) (more expensive from the computational point of view) is:

$$\boldsymbol{P}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{Z}_t)\boldsymbol{P}_{t|t-1}(\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{Z}_t)^\top + \boldsymbol{K}_t\boldsymbol{H}_t\boldsymbol{K}_t^\top \tag{9}$$

with $\boldsymbol{K}_t = \boldsymbol{P}_{t|t-1}\boldsymbol{Z}_t^\top\boldsymbol{F}_t^{-1}$; but even this ("Joseph stabilized form", see Bucy and Joseph (1968), p. 175) may prove insufficient to prevent roundoff error degeneracy in the filter. A detailed reference describing the pitfalls associated with the numerical implementation of the Kalman filter is Bierman (1977); see also Grewal and Andrews (2001), Chap. 6 and Anderson and Moore (1979), Chap. 6. Square root algorithms, that we discuss next, go a long way in improving the numerical stability of the filter.

2.2. Square root algorithms

Consider a matrix $\boldsymbol{S}_t$ such that $\boldsymbol{P}_t = \boldsymbol{S}_t\boldsymbol{S}_t^\top$; square root covariance filter (SRCF) algorithms propagate $\boldsymbol{S}_t$ instead of $\boldsymbol{P}_t$ with two main benefits (cf. Anderson and Moore (1979), § 6.5): (i) Re-constitution of $\boldsymbol{P}_t$ from $\boldsymbol{S}_t$ will always yield a symmetric non negative matrix, and (ii) The numerical condition of $\boldsymbol{S}_t$ will in general be much better than that of $\boldsymbol{P}_t$. If instead of the covariance matrix we choose to factor the information matrix $\boldsymbol{P}_t^{-1}$ we have a square root information filter algorithm (SRIF).

It is easy to produce a replacement for the time update equation (6). Consider an orthogonal matrix $\boldsymbol{G}$ such that:

$$\begin{bmatrix} \boldsymbol{M} \\ \boldsymbol{0} \end{bmatrix} = \boldsymbol{G} \begin{bmatrix} \boldsymbol{S}_{t-1}^\top\boldsymbol{T}_t^\top \\ (\boldsymbol{Q}^{1/2})^{\mathrm{T}}\boldsymbol{R}_t^\top \end{bmatrix} \tag{10}$$

where $\boldsymbol{M}$ is an upper triangular matrix. Matrix $\boldsymbol{G}$ can be constructed in a variety of ways, including repeated application of Householder or Givens transforms. Multiplication of the transpose of (10) by itself produces, taking into account the orthogonality of $\boldsymbol{G}$,

$$\begin{aligned} \boldsymbol{M}^\top\boldsymbol{M} &= \boldsymbol{T}_t\boldsymbol{S}_{t-1}\boldsymbol{S}_{t-1}^\top\boldsymbol{T}_t^\top + \boldsymbol{R}_t\boldsymbol{Q}_t^{1/2}(\boldsymbol{Q}_t^{1/2})^{\mathrm{T}}\boldsymbol{R}_t^\top & (11)\\ &= \boldsymbol{T}_t\boldsymbol{P}_{t-1}\boldsymbol{T}_t^\top + \boldsymbol{R}_t\boldsymbol{Q}_t\boldsymbol{R}_t^\top \quad ; & (12) \end{aligned}$$

comparison with (6) shows that $\boldsymbol{M}$ is a possible choice for $\boldsymbol{S}_{t|t-1}$. Likewise, it can be shown (Simon 2006, Section 6.3.4) that the orthogonal matrix $\boldsymbol{G}^*$ such that

$$\begin{bmatrix} (\boldsymbol{H}_t + \boldsymbol{Z}_t\boldsymbol{P}_{t|t-1}\boldsymbol{Z}_t^\top) & \boldsymbol{K}_t^{*\top} \\ \boldsymbol{0} & \boldsymbol{M}^* \end{bmatrix} = \boldsymbol{G}^* \begin{bmatrix} (\boldsymbol{H}_t^{1/2})^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{S}_{t|t-1}^\top\boldsymbol{Z}_t^\top & \boldsymbol{S}_{t|t-1}^\top \end{bmatrix} \tag{13}$$

produces in the left-hand side of (13) a block $\boldsymbol{M}^*$ that can be taken as $\boldsymbol{S}_t$, and thus performs the measurement update.

Although the matrices performing the block triangularizations described in equations (10) and (13) can be obtained quite efficiently (Lawson and Hanson 1974; Gentle 2007), clearly the computational effort is greater than that required by the time and measurement updates in equations (6) and (8); square root filtering does have a cost.

In the equations above $\boldsymbol{G}$ and $\boldsymbol{G}^*$ can be chosen so that $\boldsymbol{M}$ and $\boldsymbol{M}^*$ are Cholesky factors of the corresponding covariance matrices. This needs not be so, and other factorizations are possible. In particular, the factors in the singular value decomposition of $\boldsymbol{P_{t-1}}$ can be propagated: see for instance Zhang and Li (1996) and Appendix B of Petris *et al.* (2009). Also, we may note that time and measurement updates can be merged in a single triangularization (see Anderson and Moore 1979, p. 148, and Vanbegin and Verhaegen 1989).

2.3. Sequential processing

In the particular case where $\boldsymbol{H_t}$ is diagonal, the components of $\boldsymbol{y_t}^\top = (y_{1t}, \ldots, y_{pt})$ are uncorrelated. We may pretend that we observe one y_{it} at a time and perform p univariate measurement updates similar to (7)–(8), followed by a time update (5)–(6) – see Durbin and Koopman (2001, Section 6.4) or Anderson and Moore (1979) for details.

The advantage of sequential processing is that $\boldsymbol{F_t}$ becomes 1×1 and the inversion of a $p \times p$ matrix in equation (7)–(8) is avoided. Clearly, the situation where we stand to gain most from this strategy is when the dimension of the observation vector $\boldsymbol{y_t}$ is large.

Sequential processing can be combined with square root covariance and information filters, although in the latter case the computational advantages seem unclear (Anderson and Moore 1979, p. 142; see also Bierman 1977).

Aside from the case where $\boldsymbol{H_t}$ is diagonal, sequential processing may also be used when $\boldsymbol{H_t}$ is block diagonal – in which case we can perform a sequence of reduced dimension updates – or when it can be reduced to diagonal form by a linear transformation. In the last case, assuming full rank, let $\boldsymbol{H_t^{-1/2}}$ be a square root of $\boldsymbol{H_t^{-1}}$. Multiplying (2) through by $\boldsymbol{H_t^{-1/2}}$ we get

$$\boldsymbol{y_t^*} = \boldsymbol{d_t^*} + \boldsymbol{Z_t^*}\boldsymbol{\alpha_t} + \boldsymbol{\epsilon_t^*} \tag{14}$$

with $\mathsf{E}[\boldsymbol{\epsilon_t^*}\boldsymbol{\epsilon_t^*}^\top] = \boldsymbol{I}$. If matrix $\boldsymbol{H_t}$ is time-invariant, the same linear transformation will decorrelate the measurements at all times.

2.4. Smoothing and the simulation smoother

The algorithms presented produce predicted $\boldsymbol{a_{t|t-1}}$ or filtered $\boldsymbol{a_t}$ values of the state vector $\boldsymbol{\alpha_t}$. Sometimes it is of interest to estimate $\boldsymbol{a_{t|N}}$ for $0 < t \leq N$, i.e., $\mathsf{E}[\boldsymbol{\alpha_t}|\boldsymbol{y_0}, \ldots, \boldsymbol{y_N}]$, the value of the state vector given all past and future observations. It turns out that this can be done running once the Kalman filter and then a recursion backwards in time (Durbin and Koopman 2001, Section 4.3, Harvey 1989, Section 3.6).

In some cases, and notably for the Bayesian analysis of the state space model, it is of interest to generate random samples of state and disturbance vectors, conditional on the observations $\boldsymbol{y_0}, \ldots, \boldsymbol{y_N}$. Frühwirth-Schnatter (1994) and Carter and Kohn (1994) provided algorithms to that purpose, improved by de Jong (1995). Durbin and Koopman (2001, Section 4.7) draws on work of the last author; the algorithm they present is fairly involved. Recently, Durbin and Koopman (2002) have provided a much simpler algorithm for simulation smoothing of the Gaussian state space model; see also Strickland *et al.* (2009).

2.5. Exact diffuse initial conditions

As mentioned above, the Kalman filter iteration needs starting values $\boldsymbol{a_{-1}}$ and $\boldsymbol{P_{-1}}$. When nothing is known about the initial state, a customary practice has been to set $\boldsymbol{P_{-1}}$ with "large" elements along the main diagonal, to reflect our uncertainty. This practice has been criticized on the ground that

> "While [it can] be useful for approximate exploratory work, it is not recommended for general use, since it can lead to large rounding errors."

(Durbin and Koopman 2001, p. 101). Grewal and Andrews (2001, Section 6.3.2) show how this may come about when elements of $\boldsymbol{P_{-1}}$ are set to values "too large" relative to the measurement variances. An alternative is to use an information filter with the initial information matrix (or some diagonal elements of it) set to zero. Durbin and Koopman (2001, Section 5.3) advocate a different approach: see also Koopman and Durbin (2003) and Koopman (1997). The last reference discusses several alternatives and their respective merits.

2.6. Maximum likelihood estimation

An important special case of the state space model is that in which some or all of the matrices $\boldsymbol{T_t}$, $\boldsymbol{R_t}$, $\boldsymbol{Z_t}$, $\boldsymbol{Q_t}$ and $\boldsymbol{H_t}$ in equations (1)–(2) are time invariant and depend on a vector of parameters, $\boldsymbol{\theta}$ that we seek to estimate.

Assuming $\boldsymbol{\alpha_0} \sim N(\boldsymbol{a_0}, \boldsymbol{P_0})$ with both $\boldsymbol{a_0}$ and $\boldsymbol{P_0}$ known, the log likelihood is given by,

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \log p(\boldsymbol{y_0}, \ldots, \boldsymbol{y_N} | \boldsymbol{\theta}) \\ &= -\frac{(N+1)p}{2} \log(2\pi) - \frac{1}{2} \sum_{t=0}^{N} \left(\log |\boldsymbol{F_t}| + \boldsymbol{e_t}^\top \boldsymbol{F_t}^{-1} \boldsymbol{e_t} \right) \end{aligned} \tag{15}$$

where $\boldsymbol{e_t} = \boldsymbol{y_t} - \boldsymbol{Z_t a_t}$. (Durbin and Koopman 2001, Section 7.2; the result goes back to Schweppe 1965). Except for $|\boldsymbol{F_t}|$, all other quantities in (15) are computed when running the Kalman filter (cf. equations (5)–(8)); therefore, the likelihood is easily computed. (If we resort to sequential processing, Section 2.3, the analog of equation (15) does not require computation of determinants nor matrix inversions.)

Maximum likelihood estimation can therefore be implemented easily: it suffices to write a routine computing (15) as a function of the parameters $\boldsymbol{\theta}$ and use R functions such as `optim` or `nlminb` to perform the optimization. An alternative is to use the EM algorithm which quite naturally adapts to this likelihood maximization problem, but that approach has generally been found slower than the use of quasi-Newton methods, see for instance Shumway and Stoffer (2006), p. 345.

3. Kalman filtering in R

There are several packages available from the Comprehensive R Archive Network (CRAN) offering general Kalman filter capabilities, plus a number of functions scattered in other packages which cater to special models or problems. We describe five of those packages in chronological order of first appearance on CRAN.

3.1. Package dse

Package **dse** (Gilbert 2011) is the R offering including Kalman filtering that has been longest in existence. Versions for R in the CRAN archives date back at least to year 2000 (initially, its content was split in packages **dse1** and **dse2**). Before that, the software existed since at least 1996, at which time it ran on S-PLUS and R. A separate version for R has been in existence since 1998. The version used here is 2009.10-2.

dse is a large package, with fairly extensive functionality to deal with multivariate time series. The author writes,

> While the software does many standard time-series things, it is really intended for doing some non-standard things. [Package **dse**] is designed for working with multivariate time series and for studying estimation techniques and forecasting models.

dse implements three (S3) classes[1] of objects: `TSdata`, `TSmodel` and `TSestModel`, which stand for data, model and estimated model. Methods take, for instance, a `TSmodel` and `TSdata` object to return a `TSestModel`. There is a wide range of models available, covering just about any variety of the (V)(AR)(MA)(X) family. As regards state space models, two representations are supported, the non-innovations form

$$\begin{aligned} \boldsymbol{\alpha_t} &= \boldsymbol{F\alpha_{t-1}} + \boldsymbol{Gu_t} + \boldsymbol{Q\eta_t} && (16)\\ \boldsymbol{y_t} &= \boldsymbol{H\alpha_t} + \boldsymbol{R\epsilon_t}, && (17) \end{aligned}$$

and the innovations form

$$\begin{aligned} \boldsymbol{\alpha_t} &= \boldsymbol{F\alpha_{t-1}} + \boldsymbol{Gu_t} + \boldsymbol{Ke_{t-1}} && (18)\\ \boldsymbol{y_t} &= \boldsymbol{H\alpha_t} + \boldsymbol{Re_t}. && (19) \end{aligned}$$

For the relative advantages of each form, and a much wider discussion on the equivalence of different representations, see Gilbert (1993).

The first representation is similar to (1)–(2), with $\boldsymbol{Gu_t}$ taking the place of $\boldsymbol{c_t}$. There is a neat separation in `TSdata` objects between inputs $\boldsymbol{u_t}$ and outputs $\boldsymbol{y_t}$. Either representation is specified with time-invariant matrices, $\boldsymbol{F}$, $\boldsymbol{G}$, etc. There is no provision for time-varying matrices, nor for missing data.

All system matrices can be made of any mixture of constants and parameters to estimate. A very useful default is that all entries are considered parameters to estimate, with the exception of those set at 0.0 or 1.0; completely general specification of fixed constants and parameters to estimate in the system matrices can be obtained by specifying a list of logical matrices with the same sizes and names as the system matrices, and a logical `TRUE` in place of the constants. Consider, for instance the measurements of the annual flow of the Nile river at Ashwan, 1871–1970 (in **datasets**). We can set a local level model,

$$\begin{aligned} \alpha_t &= \alpha_{t-1} + \eta_t && (20)\\ y_t &= \alpha_t + \epsilon_t && (21) \end{aligned}$$

as follows:

[1]For a description of S3 and S4 classes the reader may turn respectively to Chambers and Hastie (1992) and Chambers (2008).

```
R> m1.dse <- dse::SS(F = matrix(1, 1, 1), Q = matrix(40, 1, 1),
+    H = matrix(1, 1, 1), R = matrix(130, 1, 1),
+    z0 = matrix(0, 1, 1), P0 = matrix(10^5, 1, 1))
```

Matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ in (16)–(17) are 1×1 and have been set to values different from 0.0 and 1.0, implying they are to be estimated. Should we wish to fix the value of $\boldsymbol{Q}$ to 40 and estimate only $\boldsymbol{R}$, the specification would become:

```
R> m1b.dse <- dse::SS(F = matrix(1, 1, 1), Q = matrix(40, 1, 1),
+    H = matrix(1, 1, 1), R = matrix(130, 1, 1),
+    constants = list(Q = matrix(TRUE, 1, 1), P0 = matrix(TRUE, 1, 1)),
+    z0 = matrix(0, 1, 1), P0 = matrix(10^5, 1, 1))
```

(The scope resolution operator :: will in general be unnecessary; it is used here because both **dse** and **sspir** include a function named `SS`.) There is a general purpose function `l` to estimate models, which dispatches to specialized functions according to the type of model. It takes two arguments, a `TSmodel` and a `TSdata` (or objects that can be coerced to those). With state-space models, `l` calls `l.SS`, which by default returns a `TSestModel` (but can be made to return other things, like a computed likelihood, which makes the output from `l.SS` a suitable input to an optimization routine).

dse is a mixed language package, with some routines coded in Fortran. In particular, function `l.SS` invokes alternatively R code or Fortran compiled code, according to whether argument `compiled` is `FALSE` or `TRUE`. Both the R and Fortran implementations are covariance filters, following closely the algorithm described in equations (5)–(8), with (5) substituted in (7) and (6) in (8), and additional steps of the form $\boldsymbol{P}_{t|t-1} = (\boldsymbol{P}_{t|t-1} + \boldsymbol{P}_{t|t-1}^{\boldsymbol{T}})/2$ and $\boldsymbol{P}_t = (\boldsymbol{P}_t + \boldsymbol{P}_t^{\boldsymbol{T}})/2$ to prevent rounding error from destroying the symmetry of $\boldsymbol{P}_{t|t-1}$ and $\boldsymbol{P}_t$. The Fortran implementation uses LAPACK (Anderson *et al.* 1999) routines.

Routines for different estimation methods are provided in the package. A convenient function is `estMaxLik`, which performs maximum likelihood for Gaussian models. The following code estimates the univariate local level model in equations (16)–(17) (defined in `m1.dse` above) with the well known Nile data:

```
R> data("Nile", package = "datasets")
R> m1b.dse.est <- estMaxLik(m1b.dse, TSdata(output = Nile))
```

However, it is important to realize that the likelihood which is optimized is not that given by (15), but rather a large sample approximation (see for instance Harvey 1989, Section 3.4.1). For this reason, package **dse** is dropped from some comparisons in Section 4.2, as the results differ from those obtained with other packages. The Kalman filtering routine `l.SS`, however, is still compared in terms of speed with its counterparts in other packages.

As is the case with other packages reviewed in the following, **dse** goes well beyond what has been described. It is geared towards the use of alternative representations and evaluation of different estimation procedures (package **EvalEst**, Gilbert 2010, provides a complement of functions for the last purpose). Aside from that, it offers useful utility functions to check stability, observability, compute the McMillan degree, solve Riccati equations, and more.

3.2. Package sspir

The first version of **sspir** (Dethlefsen *et al.* 2009) on CRAN dates back to 2005; the version we comment upon is version 0.2.8 of 2009. It does not contain documentation other than the manual pages, but is well described in Dethlefsen and Lundbye-Christensen (2006). In addition, the package is used in Cowpertwait and Metcalfe (2009).

sspir is a pure R package, i.e., contains no compiled code. It uses an object-oriented approach. A Gaussian state space model as in (1)–(2) (but with no deterministic inputs $\boldsymbol{c_t}$, $\boldsymbol{d_t}$ nor matrix $\boldsymbol{R_t}$) is represented by an object of (S3) class `SS`. There is a constructor function of the same name that yields an `SS` object when invoked with the required information. The covariance matrices $\boldsymbol{Q_t}$, $\boldsymbol{H_t}$ (`Wmat` and `Vmat` in the notation of the package) can be constant matrices or functions, while measurement and transition matrices $\boldsymbol{Z_t}$ and $\boldsymbol{T_t}$ (`Fmat` and `Gmat` in the notation of the package) have to be functions of time, a parameter vector ϕ and possibly a set of covariates. For instance, the local level model for the Nile data in equations (20)–(21) can be specified as follows:

```
R> data("Nile", package = "datasets")
R> m1.sspir <- sspir::SS(Fmat = function(tt, x, phi) {
+     return(matrix(1))
+ }, Gmat = function(tt, x, phi) {
+     return(matrix(1))
+ }, Vmat = function(tt, x, phi) {
+     return(matrix(exp(phi[1])))
+ }, Wmat = function(tt, x, phi) {
+     return(matrix(exp(phi[2])))
+ }, y = as.matrix(Nile, ncol = 1))
R> str(m1.sspir)
```

```
List of 19
 $ y         : num [1:100, 1] 1120 1160 963 1210 1160 1160 813 1230 1370 ...
 $ x         :List of 1
  ..$ x: logi NA
 $ Fmat      :function (tt, x, phi)
 $ Gmat      :function (tt, x, phi)
 $ Vmat      :function (tt, x, phi)
 $ Wmat      :function (tt, x, phi)
 $ m0        : num [1, 1] 0
 $ C0        : num [1, 1] 100
 $ phi       : num [1:2] 1 1
 $ n         : int 100
 $ d         : int 1
 $ p         : num 1
 $ ytilde    : logi NA
 $ iteration : num 0
 $ m         : logi NA
 $ C         : logi NA
 $ mu        : logi NA
 $ likelihood: logi NA
 $ loglik    : logi NA
 - attr(*, "class")= chr "SS"
```

The constructor yields (in `m1.sspir`) an object with the passed arguments filled in and some other components (which might also have been specified in the call to `SS`) set to defaults. Among others, we see `y`, containing the time series, `m0` and `C0` (initial mean and covariance matrix of the state vector), `phi` (vector of parameters), etc. The only two parameters σ^2_η and σ^2_ϵ have been parameterized as $\exp(\phi_1)$ and $\exp(\phi_2)$ to ensure non negativity. A rich set of accessor functions allow to get/set individual components in an `SS` object. For instance, we can set the initial covariance matrix of the state vector (`C0`) and the model parameters (`phi`),

```
R> C0(m1.sspir) <- matrix(10^7, 1, 1)
R> phi(m1.sspir) <- c(9, 7)
```

and then inspect their values, as in

```
R> phi(m1.sspir)
```

```
[1] 9 7
```

Functions `kfilter` and `smoother` perform Kalman filtering and smoothing on the `SS` object `m1.sspir`, returning a new `SS` object with the original information, the filtered (or smoothed) estimates of the state, and their covariance matrices in slots `m` and `C` respectively:

```
R> m1.sspir.f <- kfilter(m1.sspir)
```

Aside from the use of function `SS` to set up a state space model, **sspir** offers a function `ssm` based on the familiar formula notation of functions like `lm`, `glm`, etc. For instance, the local level model for the Nile data could be specified as follows:

```
R> mod2.sspir <- ssm(Nilo ~ tvar(1), family = "gaussian",
+     C0 = diag(1) * 10^7, phi = exp(c(9, 7)))
R> class(mod2.sspir)
```

```
[1] "ssm" "lm"
```

The returned object is of class `ssm`. Marker `tvar` specifies that a right hand side element is time varying (in the example above, the intercept). The `family` argument can take values `gaussian`, `binomial`, `Gamma`, `poisson`, `quasibinomial`, `quasipoisson` and `quasi`, each with several choices of link functions. If `gaussian` is specified, the function `kfs` runs the standard Kalman smoother. For a non-Gaussian family, `kfs` runs an iterated extended Kalman smoother: it iterates over a sequence of approximating Gaussian models.

Function `ssm` provides a very convenient user interface, even if it can only be used for univariate time series. The use of `ssm` is further enhanced by ancillary functions like `polytrig`, `polytime`, `season` and `sumseason`, which help to define polynomial trends and trigonometric polynomials as well as seasonal dummies.

sspir includes other useful functions for: simulation from a (Gaussian) state-space model (`recursion`), simulation smoothing (`ksimulate`) and EM estimation (`EMalgo`).

Regarding the algorithmic approach followed, **sspir** implements a covariance filter, with the faster (8) rather than the more stable (9) update for the state covariance. Somewhat surprisingly, missing values are not allowed.

All in all, the design of the package emphasizes usability and flexibility: the fact that system matrices can (or need) be specified as functions of time, covariates and parameters affords unparallelled flexibility. This flexibility has a price, however, as every single use of a matrix forces its recreation from scratch by the underlying function. As we will see in Section 4, this, coupled with the all-R condition of the package, severely impacts performance.

3.3. Package dlm

Version 0.7-1 of package **dlm** was submitted to CRAN in August 2006. The version we review is 1.1-1 (see Petris 2010). Other than the manual pages, an informative vignette accompanies the package. A recent book, Petris *et al.* (2009), provides an introduction to the underlying theory and a wealth of examples on the use of the package.

Like **sspir**, the state-space model considered is a simplified version of (1)–(2), with no intercepts $\boldsymbol{c_t}$, $\boldsymbol{d_t}$ and no $\boldsymbol{R_t}$ matrix. The emphasis is on Bayesian analysis of dynamic linear models (DLMs), but the package can also be used for Kalman filtering/smoothing and maximum likelihood estimation. The design objectives as stated in the concluding remarks of the vignette Petris (2010) are

> "... flexibility and numerical stability of the filtering, smoothing, and likelihood evaluation algorithms. These two goals are somewhat related, since naive implementations of the Kalman filter are known to suffer from numerical instability for general DLMs. Therefore, in an environment where the user is free to specify virtually any type of DLM, it was important to try to avoid as much as possible the aforementioned instability problems."

In order to cope with numerical instability, the author of the package has chosen to implement a form of square root filter described in Zhang and Li (1996), which propagates factors of the singular value decomposition (SVD) of $\boldsymbol{P_t}$. It is claimed that the algorithm is more robust and general than standard square root filters. The computation is done in C, with extensive use of LAPACK routines. Initial conditions $\boldsymbol{a_{0|-1}}$ and $\boldsymbol{P_{0|-1}}$ have to be specified (or the defaults accepted). There is no provision for exact initial diffuse conditions.

Aside from the heavy duty compiled functions, the package provides a rich set of interface functions in R, to help with the specification of state-space models. A time-invariant model is an object of (S3) class `"dlm"`. A general constructor is provided, not unlike `SS` in package **sspir**, which returns a `dlm` object when invoked with arguments `V`, `W` (covariance matrices of the measurement and state equations, respectively), `FF` and `GG` (measurement equation matrix and transition matrix respectively), and `m0`, `C0` (prior mean and covariance matrix of the state vector). For instance,

```
R> m1.dlm <- dlm(FF = 1, V = 0.8, GG = 1, W = 0.1, m0 = 0, C0 = 100)
```

creates a `dlm` object representing the model in (20)–(21) with $\sigma^2_\epsilon = 0.8$ and $\sigma^2_\eta = 0.1$. The class and structure of the object is

```
R> str(m1.dlm)

List of 6
 $ m0: num 0
 $ C0: num [1, 1] 100
 $ FF: num [1, 1] 1
 $ V : num [1, 1] 0.8
 $ GG: num [1, 1] 1
 $ W : num [1, 1] 0.1
 - attr(*, "class")= chr "dlm"
```

Much as in **sspir**, there is a full complement of functions to get/set specific components of `dlm` class objects.

Time-varying system matrices and covariates are handled in a manner reminiscent of the approach used in `SSFPack` (Koopman *et al.* 1999). Whenever one of the matrices `FF`, `GG`, `V` or `W` has time-varying elements, we have to specify a similarly sized matrix `JFF`, `JGG`, `JV` or `JW` with zeros in the places of time invariant elements and a positive integer otherwise. If entry `JFF[i,j]` is set to `k`, that means that at time `s` `FF[i,j]` is equal to `X[s,k]`. For instance, assuming `y` is an N length vector of observations and `X` an $N \times 2$ matrix of covariates, a linear regression model

$$y_t = \beta_{0t} + \beta_{1t}X_{1t} + \beta_{2t}X_{2t} + \epsilon_t \tag{22}$$

in which all β's are allowed to change over time following a random walk dynamics with $\sigma^2_{\beta_0} = \sigma^2_{\beta_1} = \sigma^2_{\beta_2} = 0.1$, could be specified as follows:

```
R> m2.dlm <- dlm(FF = matrix(c(1, 0, 0), 1, 3),
+    JFF = matrix(c(0, 1, 2), 1, 3), GG = diag(3), W = 0.1 * diag(3),
+    V = 4, m0 = c(0, 0, 0), C0 = 10^9 * diag(3), X = X)
```

Unlike with **sspir**'s objects of class `SS`, the series of observations is not included in the `dlm` object.

sspir enhanced usability with the `ssm` function interface and ancillary functions to create common patterns in the system matrices. **dlm** has ancillary functions `dlmModARMA`, `dlmModPoly`, `dlmModReg`, `dlmModSeas` and `dlmModTrig`, which help with ARMA, polynomial trends, regression terms and seasonal components in two different forms. For instance, we might use

```
R> m3.dlm <- dlmModARMA(ar = c(0.3, 0.2, 0.4), ma = c(0.9, 0.1))
R> m3.dlm

$FF
     [,1] [,2] [,3]
[1,]    1    0    0

$V
     [,1]
[1,]    0
```

```
$GG
     [,1] [,2] [,3]
[1,]  0.3    1    0
[2,]  0.2    0    1
[3,]  0.4    0    0

$W
     [,1] [,2] [,3]
[1,]  1.0 0.90 0.10
[2,]  0.9 0.81 0.09
[3,]  0.1 0.09 0.01

$m0
[1] 0 0 0

$C0
      [,1]  [,2]  [,3]
[1,] 1e+07 0e+00 0e+00
[2,] 0e+00 1e+07 0e+00
[3,] 0e+00 0e+00 1e+07
```

for an ARMA(3,2) model (the `ar=` and `ma=` arguments could be lists of matrices, so multivariate time series can be handled just as easily).

While **dlm** does not offer a function like `ssm` allowing simple formula notation, it implements a very powerful mechanism which greatly eases the creation of system matrices. Generic operator `'+'` has been overloaded to "add" model components into a comprehensive model for a time series. Thus, one can write

```
R> m4.dlm <- dlmModPoly(order = 1) + dlmModSeas(4)
```

to create a local level plus (period 4) seasonal model.

In addition, a new "outer sum" operator has been defined on `dlm` objects that combines models for different time series into a joint model. This eases the definition of models in which different components have different dynamics. For instance, `m3.dlm` and `m4.dlm` could be combined as in:

```
R> m3Plusm4.dlm <- m4.dlm %+% m3.dlm
```

This yields a bivariate time series model, where the first component follows a local level plus seasonal dynamics, while the second is an ARMA(3,2), with a state vector of total dimension $d = 7$. The transition matrix is

```
R> GG(m3Plusm4.dlm)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    0    0    0  0.0    0    0
[2,]    0   -1   -1   -1  0.0    0    0
```

```
[3,]    0    1    0    0  0.0    0    0
[4,]    0    0    1    0  0.0    0    0
[5,]    0    0    0    0  0.3    1    0
[6,]    0    0    0    0  0.2    0    1
[7,]    0    0    0    0  0.4    0    0
```

and the observation matrix

```
R> FF(m3Plusm4.dlm)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    0    0    0    0    0
[2,]    0    0    0    0    1    0    0
```

It is assumed that the blocks "added" are independent, but the resulting `dlm` object can be amended if they are not. Both the overloaded generic `+` and `%+%` are real productivity boosters.

Maximum likelihood estimation of the Gaussian model is handled by function `dlmMLE`: it requires as arguments a vector of parameters and a function which builds the state space model for the given values of the parameters, and then invokes the standard optimizing function `optim`.

Package **dlm** is geared towards Bayesian analysis of time series, and offers facilities which go way beyond what we have described. For a fuller account, the reader is referred to Petris *et al.* (2009).

3.4. Package FKF

Package **FKF** (Luethi *et al.* 2010) first appeared on CRAN in February 2009; the version we comment upon is version 0.1.0. The name of the package stands for "Fast Kalman Filter", and emphasis is on speed. From the `DESCRIPTION` file,

> "Due to the speed of the filter, the fitting of high-dimensional linear state space models to large datasets becomes possible."

It contains only two functions, `fkf` and `plot.fkf`. The second is an ancillary function to plot filtering results. `fkf` is a wrap envelope in R of a C routine implementing the filter. The state space model considered is as described by equations (1)–(2) with an added input matrix $\boldsymbol{S_t}$ in the measurement equation, which is thus

$$\boldsymbol{y_t} = \boldsymbol{d_t} + \boldsymbol{Z_t}\boldsymbol{\alpha_t} + \boldsymbol{S_t}\boldsymbol{\epsilon_t}. \tag{23}$$

All system matrices $\boldsymbol{c_t}$, $\boldsymbol{T_t}$, $\boldsymbol{d_t}$, $\boldsymbol{Z_t}$, $\boldsymbol{Q_t}$, $\boldsymbol{S_t}$ and $\boldsymbol{H_t}$ may be constant or time varying. The latter case requires their definition as three dimensional arrays, the last dimension being time.

Aside from doing all computations in compiled code, the authors have chosen to use a covariance filter rather than a square root filter, consistent with their quest of maximum speed: they code equations (5)–(8) almost verbatim.

Computation is multivariate, i.e., they do not take advantage of sequential processing – which for large dimensions of the observation vector $\boldsymbol{y_t}$ and diagonal $\boldsymbol{H_t}$ might offer substantial advantages. Rather, they focus on fast linear algebra computations, with extensive resort to BLAS and LAPACK routines. Not using the sequential approach makes it harder to deal with scattered missing values in vector $\boldsymbol{y_t}$, which were not supported in previous versions of the package (but are as of version 0.1.0)

Function `fkf` computes and returns the likelihood, as in equation (15). Computation of the likelihood is stopped if $\boldsymbol{F_t}$ happens to be singular, something which should never be the case for positive definite matrices $\boldsymbol{H_t}$. Function `fkf` returns both filtered $\boldsymbol{a_t}$ and predicted $\boldsymbol{a_{t|t-1}}$ estimations of the state. It makes no provision, though, for smoothed values $\boldsymbol{a_{t|N}}$ nor simulation smoothing.

While limited in the scope of the facilities it offers, **FKF** excels in the speed of the filter, as will be apparent in the examples shown later.

3.5. Package KFAS

Package **KFAS** (Helske 2010) is the most recent addition to Kalman filtering in R. It includes functions for Kalman filtering, smoothing, simulation smoothing and disturbance smoothing. It also includes an ancillary function to forecast from the output of the Kalman filter. The model supported is (1)–(2), except for $\boldsymbol{c_t}$ and $\boldsymbol{d_t}$.

The package follows closely Durbin and Koopman (2001) and papers by the same authors (Koopman and Durbin 2003, 2000) implementing algorithms as presented in those references. Time varying system matrices are supported, as are missing values. The R functions do some error checking and call Fortran 90 routines which perform most of the computational effort; extensive use is made of BLAS and LAPACK for all linear algebra computations. As in the case of the preceding package, time-varying matrices are handled by using three dimensional arrays, the last dimension being time. The package contains seven user visible functions: `kf`, `ks`, `simsmoother`, `distsmoother`, `eflik`/`eflik0` and `forecast` for, respectively, Kalman filtering, smoothing, simulation smoothing, disturbance smoothing, approximate log-likelihood computation of non-Gaussian models and forecasting.

One of the characteristics that set this package apart is the use of sequential processing (Section 2.3 above). Aside from the expected performance advantages when the dimension of $\boldsymbol{y_t}$ is large, this approach makes it easy to deal with irregularly missing entries: they are simply dropped from $\boldsymbol{y_t}$ (and so are the corresponding rows of $\boldsymbol{Z_t}$ and rows and columns of $\boldsymbol{H_t}$)

Sequential processing requires that the components of $\boldsymbol{y_t}$ be uncorrelated or else that the "decorrelation" described just prior to equation (14) is carried out. This is done by the `kf` function which at each step of the Kalman filter iteration, checks for diagonality $\boldsymbol{H_t}$ (or a sub-matrix $\boldsymbol{\tilde{H}_t}$, if some rows and columns have been dropped on account of missing values in $\boldsymbol{y_t}$). If it is not diagonal, the observation equation is transformed to a form like (14) by multiplying by a square root of $\boldsymbol{H_t}^{-1}$ (or $\boldsymbol{\tilde{H}_t^{-1}}$): a Cholesky factor is used, in the case of full rank), and the resulting $\boldsymbol{y_t^*}$ is processed one element at a time.

Another feature that sets this package apart is the use of exact initial diffuse conditions *à la* Durbin and Koopman (2001). Function `kf` can be supplied with a proper initial distribution or an exact diffuse initialization for some or all of the elements of the initial state vector.

Although processing data sequentially, **KFAS** still implements a standard covariance filter, much as in equations (5)–(8).

3.6. Other functions performing Kalman filtering

There are functions in other R packages which cater to particular cases of state space models. The function `StructTS` by B.D. Ripley in recommended package **stats** fits basic structural models (see Harvey (1989)) to univariate time series. It is quite reliable and fast, and is used as a benchmark in some of the comparisons that follow in Section 4. (Functions `KalmanLike`, `KalmanRun`, `KalmanSmooth` and `KalmanForecast` in the same package **stats** can also be used, although are not intended to be called directly, but rather by other functions.) Package **Stem** (Cameletti 2009) includes `Stem.Model` and `Stem.Estimation` functions, specifying and fitting (using a EM algorithm, entirely coded in R) a particular spatio-temporal model. Kalman filtering and smoothing routines are present also in other packages, including **timsac** (The Institute of Statistical Mathematics 2009) and **cts** (Tunnicliffe-Wilson and Wang 2006).

4. Features and speed comparison

4.1. Features

A brief description of some of the capabilities of each package has been given in the preceding section. Table 1 is provided as a quick summary, by no means complete. It should be understood, on the other hand, that some features can be easily implemented even if not already coded in the package. For instance, maximum likelihood estimation only requires a few lines of code once we have a routine which computes the likelihood of the state space model (which all packages provide).

	dse	**sspir**	**dlm**	**FKF**	**KFAS**
	2009.10-2	0.2.8	1.1-1	0.1.0	0.6.0
Coded in:	R + Fortran	R	R + C	R + C	R + Fortran
Class model (if any)	S3	S3	S3	S3	
Algorithm	CF	CF	SRCF	CF	CF
Sequential processing					$\surd$
Exact diffuse initialization					$\surd$
Missing values in $\boldsymbol{y}_t$ allowed			$\surd$	$\surd$	$\surd$
Time varying matrices		$\surd$	$\surd$	$\surd$	$\surd$
Simulator	$\surd$		$\surd$		
Smoother	$\surd$	$\surd$	$\surd$		$\surd$
Simulation smoother		$\surd$	$\surd$		$\surd$
Disturbance smoother					$\surd$
MLE routine	$\surd$	$\surd$	$\surd$		
Non-Gaussian models		$\surd$			$\surd$

Table 1: Feature comparison of five packages performing Kalman filtering. CF = Covariance filter, SRCF = Square root covariance filter.

4.2. Speed comparison

In order to assess the relative speed of the four packages examined, we have conducted a few experiments. In all of them we have estimated parameters in Gaussian models of different sizes by maximum likelihood. This requires repeated runs of the Kalman filter and has been done either using a function provided in the package (e.g., `dlmMLE` in package **dlm**) or writing a small function which computes the log likelihood (using the Kalman filtering routine in each package) and then handing over that likelihood to function `optim` for optimization.

The following data sets have been used in the experiments:

Nile data. This is the well-known Nile data described in Durbin and Koopman (2001) and referred to earlier. It is a short ($N = 100$ observations) univariate time series. A local level model involving just two parameters (σ^2_η and σ^2_ϵ, variances of the innovation and the measurement noise) has been fitted by maximum likelihood, with all four packages and with function `StructTS`.

Exchange rates data. Data consists of $N = 2465$ daily observations of the exchange rate (against the US dollar, USD) of four currencies (AUD, BEF, CHF and DEM), for the period from 1988-01-04 to 1997-08-15. Their joint evolution is displayed in Figure 1

Regression artificial data. The data consists of $N = 1000$ artificially generated observations from the linear model $y_t = \beta_1 X_1 + \ldots + \beta_5 X_5 + \epsilon$, with $\beta_i = i$ for $i = 1, \ldots, 5$. The vector $\boldsymbol{\beta}$ can be thought of as an invariant state vector (i.e., $\boldsymbol{Q_t} = \mathbf{0}$). The model can be estimated by ordinary least squares and sequentially, by means of the Kalman filter. The purpose of this experiment was to check the accuracy of the algorithms in an example with $\boldsymbol{Q_t} = \mathbf{0}$, which might expose numerical instability when equations (6) and (8) are used.

Results. The benchmarks were run on a 2GHz. Xeon machine, running Linux and R compiled for 64 bits with an optimized BLAS library (ATLAS, see Whaley *et al.* 2001). A machine using a non-optimized library might see different relative performances, as some of the packages rely very heavily on BLAS. We have selected algorithm L-BFGS-B, Byrd *et al.* (1995), as coded in function `optim` in R. The feasible region for parameters (non-negative half line for variances, $(-1, 1)$ for correlation coefficients) has been specified in the call to `optim`. When coding bounds for variances or correlation coefficients, we found that we had to force parameters to be slightly away from the boundary of the feasible region: lower bounds of 10^{-6} were given for variances and a feasible interval of $(-0.99, 0.99)$ for correlation coefficients. Common initial values, tolerance and scaling have been used for each experiment with all four packages.

The list of models fitted and the times taken are summarized in Table 2, which we comment briefly next.

In the case of the Nile example (local level model fitted to a short univariate series), the interpreted nature of the **sspir** is seen taxing its performance very heavily: it takes much longer than the second slowest package. Even among the packages implementing the Kalman filter in compiled code, there are noticeable differences, **FKF** being clearly the fastest. Interestingly,

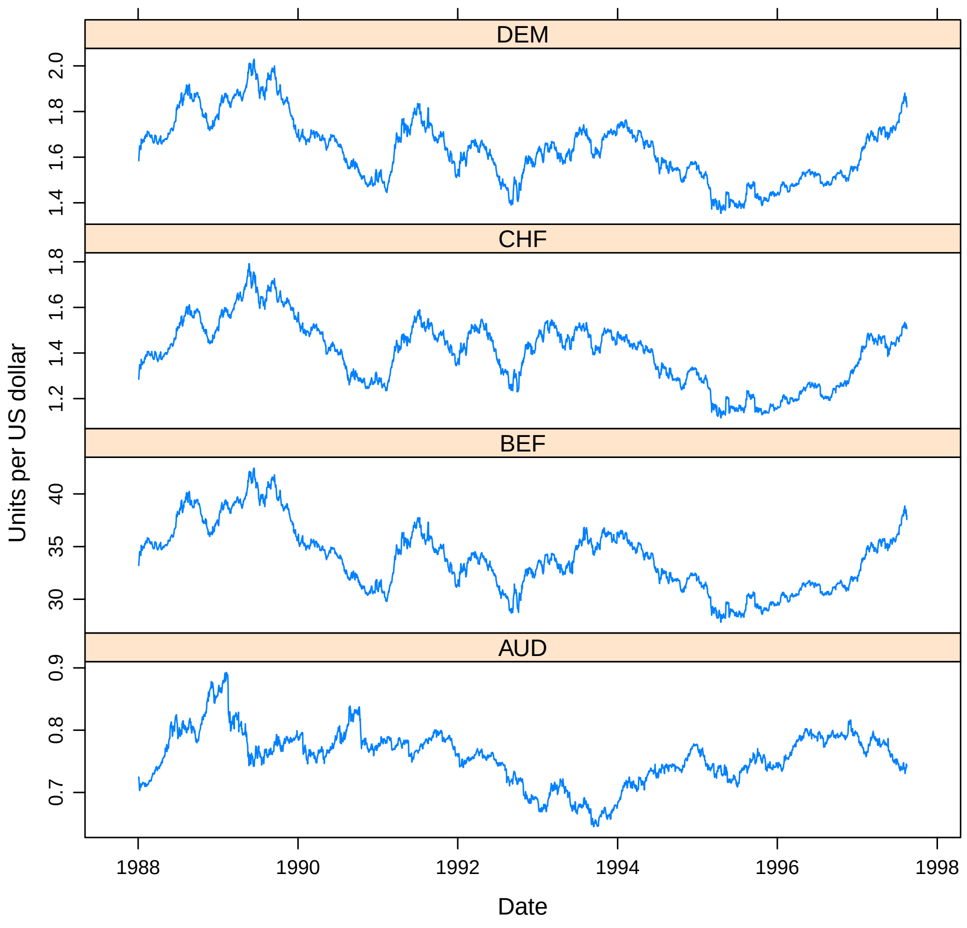

Figure 1: Daily exchange rate against the US dollar (USD) of the Australian dollar (AUD), Belgian franc (BEF), Swiss franc (CHF) and German mark (DEM). Period 1988-01-04 to 1997-08-15.

in spite of using a type of square root algorithm (SVD factorization), **dlm** ends up being about as fast as **KFAS** in this particular example.

The Nile example is a particular case of structural model, which can be fit most conveniently using function `StrucTS`. Calling,

```
R> fit <- StructTS(Nile, type = "level")
```

returns the local level model fit in about 0.02 seconds!

For the exchange rates data, we have considered p-variate subsets with $p = 2, 3$ and 4 currencies, to see the performance of the different packages as the dimension of the observation vector increases (and in particular to assess the benefits afforded by the sequential approach used in package **KFAS**). To each p-variate time series of length $N = 2465$ we have fitted a time-invariant local level model. Observations are assumed contemporaneously correlated,

	sspir	**dlm**	**FKF**	**KFAS**	
				proper	diffuse
I. Nile example Local level model ($p = 1$, $d = 1$, $N = 100$)	10.13	0.17	0.07	0.17	0.15
II. Exchange rates example Local level model $\boldsymbol{V_t}$ equicorrelated ($p = 2$, $d = 2$, $N = 2465$)	1058.22	19.28	7.40	6.63	6.44
III. Exchange rates example Local level model $\boldsymbol{V_t}$ equicorrelated ($p = 3$, $d = 3$, $N = 2465$)	4105.41	117.32	27.50	47.76	74.69
IV. Exchange rates example Local level model $\boldsymbol{V_t}$ equicorrelated ($p = 4$, $d = 4$, $N = 2465$)	6853.85	264.25	64.89	122.15	98.67
V. Exchange rates example Single factor model $\boldsymbol{V_t}$ diagonal ($p = 3$, $d = 1$, $N = 2465$)	3334.48	62.68	$31.41^{\ddagger}$	21.17	19.60
VI. Regression model $\boldsymbol{Q_t} = \boldsymbol{0}$ ($p = 1$, $d = 5$, $N = 1000$)	8.52	1.13	0.23	0.48	0.28

Table 2: Times in seconds for the maximum likelihood estimation of the models indicated. p, d, N are respectively the dimension of $\boldsymbol{y_t}$, the dimension of the state vector and the length of time series. Symbol ‡ marks abnormal exit in the optimization routine; see text.

with pairwise common correlation ρ: $\boldsymbol{V_t}$ has p distinct variances along the main diagonal and off-diagonal (i, j) element of the form $\sigma_i\sigma_j\rho$ for all (i, j). We have assumed the disturbance driving the state vector to have a diagonal covariance matrix $\boldsymbol{Q_t}$. Thus, we have $2p + 1$ parameters: p variances in each of $\boldsymbol{V_t}$ and $\boldsymbol{Q_t}$ plus ρ. (We are not claiming that this model is anywhere close to optimal for the data at hand, although one could find some rationale for it.)

For the series BEF, CHF and DEM, with a very similar evolution with respect to the US$ (see Figure 1), we have also considered a dynamic single factor model,

$$\boldsymbol{y_t} = \boldsymbol{Z_t}\alpha_t + \boldsymbol{\epsilon_t} \tag{24}$$

$$\alpha_t = \alpha_{t-1} + \eta_t. \tag{25}$$

Matrices $\boldsymbol{T_t}$ and $\boldsymbol{V_t}$ are assumed time invariant, with $\boldsymbol{V_t}$ diagonal. (This is the best scenario for sequential processing, which benefits from large p relative to d and uncorrelated inputs.)

Thus, there are six parameters to estimate: three variances in $\boldsymbol{V_t}$ plus σ^2_η and two parameters in $\boldsymbol{T_t}$ (the first entry in $\boldsymbol{T_t}$ has been set to 1 to insure identifiability).

In all cases, a common set of starting values has been used with all the packages, roughly within an order of magnitude of the optima found. Not only the starting values were important, but also the scaling, which took some trial and error to set.

We see again that the interpreted nature of the Kalman filter routine in **sspir** exacts a heavy toll on performance, with running times from 50 to over a hundred times longer than those of the other packages. It is not only that code is interpreted, but also there is heavy overhead from repeated function calls: each iteration of the Kalman filter calls a `filterstep` function. Likewise, the likelihood term of (15) computed at each iteration requires an additional function call, which incurs further overhead by checking formal consistency of its arguments over and over again.

With one of the models, when optimizing a likelihood computed with package **FKF**, `optim` stopped with a message `ERROR: ABNORMAL_TERMINATION_IN_LNSRCH`: the returned values of the parameters where in close agreement, though, with those returned when using **sspir**, **dlm** or **KFAS**.

We ran many more experiments, including in the data set exchange rates for more currencies, up to twelve at a time: the $\boldsymbol{V_t}$-equicorrelated local level model then has $12 + 12 + 1$ free parameters. In general, with dimensions larger than those reported in Table 2, attempts to fit models by maximum likelihood were quite sensitive to the starting values and scaling supplied to `optim`, frequently unsuccessful and always costly.

Based only on the results reported in Table 2, we can conclude that **FKF** is fastest, by a factor of about 2, except in example II, where **KFAS** is slightly faster, and example V where it is significantly faster. In turn, **KFAS** is faster than **dlm** by a factor between 2 and 3. Contrary to our initial expectation, the use of an exact diffuse initialization in **KFAS** does not seem to add to the running time.

Examples such as those in Table 2 are interesting in that they give timings for a common task in time series analysis – maximum likelihood estimation of a model. However, those times only partially reflect the relative performance of the Kalman filtering routines, as they include the running time of the optimizing routine – in our case, `optim`.

To better assess the relative speed of the different Kalman filter routines, we can run them on time series of different lengths, with varying state and measurement dimensions. We have excluded from the comparison package **sspir**, whose running times are clearly in a different category. On the other hand, we have included **dse**, which was not included in Table 2 because it computes the likelihood in a different manner, and hence does not reach the same optima as the rest of the packages.

The results can be seen in Figure 2. We have used multivariate random walks of length $N =$ 300 to 1900, state dimensions $d = 2, 4, 8$ and 12 and measurements of dimension $p = 2, 4, 8, 10$ and 12. Timings have been obtained by averaging the total time of 25 runs using the function `system.time`.

The timing method is not very accurate, even after averaging 25 different realizations. Specially annoying is the fact that some of the timings do not monotonically increase with N, as one would expect. (Prior to each set of 25 Kalman filter runs, a garbage collection was forced, to minimize the chance that it be automatically called within the timed loop. We may have been unsuccessful at preventing such undesired garbage collections in all cases.) Nonetheless,

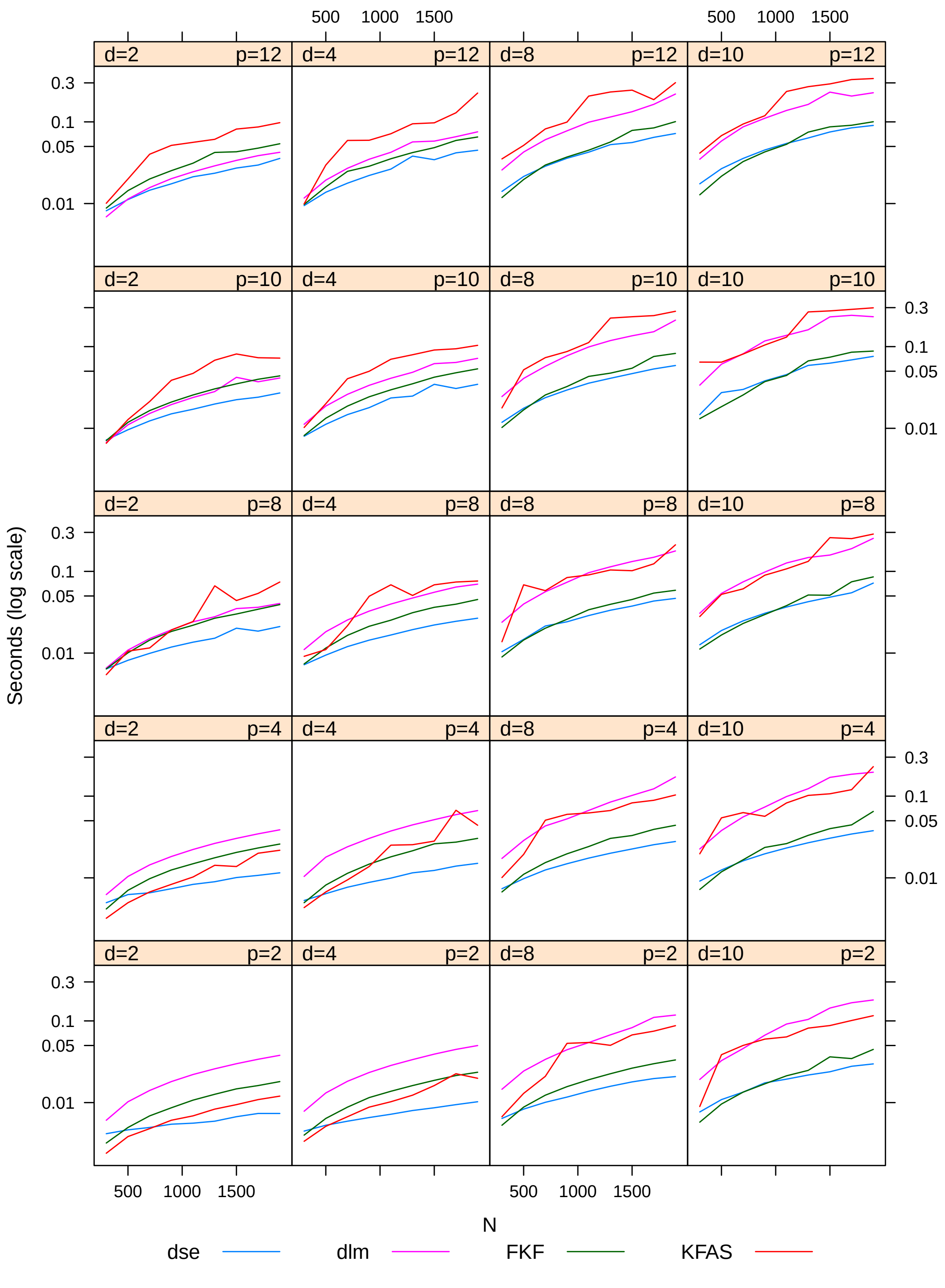

Figure 2: Average running time in (in seconds) of a Kalman filtering operation, over a time series of length N with state dimension d and observation vector of dimension p.

					KFAS	
True β_i	OLS	**sspir**	**dlm**	**FKF**	proper	diffuse
1.000000	1.052865	1.052864	1.052864	1.052864	1.052864	1.052865
2.000000	2.006062	2.006060	2.006060	2.006060	2.006060	2.006062
3.000000	2.996563	2.996559	2.996559	2.996559	2.996559	2.996563
4.000000	4.061145	4.061140	4.061140	4.061140	4.061140	4.061145
5.000000	4.988728	4.988723	4.988723	4.988723	4.988723	4.988728

Table 3: Real and estimated parameter values in the regression example with $N = 1000$. The **OLS** column displays the estimated values of the betas by ordinary least squares. The remaining columns show the estimates using the Kalman filter as implemented by the four packages with initialization $\boldsymbol{a_0} = \mathbf{0}$, $\boldsymbol{P_0} = 1000 \times \boldsymbol{I}$ or exact diffuse (last column).

the patterns in Figure 2 are clear: for most combinations of state and measurement dimension and most series lengths, **FKF** and **dse** Kalman filtering routines come out fastest. **dlm** is in general about 4-5 times slower (note the log scale in Figure 1), as one would expect given the greater computational effort demanded by the square root algorithm it uses; however, for small state dimensions and relatively large measurement dimensions, it does quite well, with performances close to **FKF** and **dse**.

Unexpectedly, **KFAS**, which one would expect to improve relatively to the others when p is large, displays the opposite effect, at least for small d: as we go upwards in the first and second columns of Figure 1, the performance of **KFAS** degrades relatively to the others. For models with larger state dimension ($d = 8$ and $d = 10$), timings for **KFAS** are close to those of **dlm**, and both considerably above those of **dse** and **FKF**.

It is important to realize, though, that routines in different packages produce more or less comprehensive outputs, which in part accounts for their different running times. It should also be borne in mind packages like **dse** which assume time-invariant system matrices, are given an advantage over those which make allowance for time-varying matrices (incurring the overhead of three rather than two index arrays, the third index for time).

Besides looking at the running times, it is interesting to compare results in the estimated parameters.

Table 3 shows the estimated betas in the regression example (VI in Table 2). All four packages, when run with the same initial conditions, produce results which match to 5-6 decimal places. Function `kf` in package **KFAS**, when run with exact diffuse initial conditions, produces slightly different results. All estimations are in good agreement with those obtained by ordinary least squares, as it should be (see Eubank 2006, Section 6.3).

The differences in the estimated values when exact initial diffuse conditions and approximate diffuse conditions were used, are quite small, as is in general the case with "long" time series. This is not always so: on the issue of fitting structural models and the impact of initial conditions Ripley (2002) gives some insightful comments.

Agreement among the estimates produced by the different packages is not quite so good in the other experiments, but estimates are still close. For the case of the single factor model fitted to the currencies exchange rates (V in Table 2) we can see the results in Table 4. Some discrepancies among estimates computed by the different packages can be seen starting at the

	Estimated by:			
Parameter	**sspir**	**dlm**	**FKF**	**KFAS**
$\boldsymbol{V}_{\mathbf{11}} \times 10^5$	7.345022	7.329658	7.336880	7.331016
$\boldsymbol{V}_{\mathbf{22}} \times 10^3$	1.154476	1.145747	1.184988	1.145720
$\boldsymbol{V}_{\mathbf{33}} \times 10^6$	3.614857	3.623933	3.592682	3.625587
$\boldsymbol{Z}_{\mathbf{21}}$	1.055430	1.054714	1.054527	1.054738
$\boldsymbol{Z}_{\mathbf{23}}$	0.942436	0.942302	0.942290	0.942294
$\sigma_\eta^2 \times 10^5$	4.585722	4.600876	4.603320	4.599717
sspir $\ell(\boldsymbol{\theta})$	21780.17	21780.22	21779.52	21780.23
dlm $-\ell(\boldsymbol{\theta})$	-28575.72	-28575.77	-28575.08	-28575.77
FKF $\ell(\boldsymbol{\theta})$	21780.18	21780.22	21779.54	21780.22
KFAS $\ell(\boldsymbol{\theta})$	28575.72	28575.77	28575.08	28575.77

Table 4: Estimates of parameters for the currency exchange rates single factor model, suitably scaled. Last four rows report the (minus) log likelihood $\ell(\boldsymbol{\theta})$ of each set of estimates using the Kalman filter routines of all four packages.

second significant digit.

The last four rows of Table 4 display the log likelihood evaluated by each of the four packages at the parameter values directly above. (This allows comparison of the four sets of estimates in terms of the log likelihood as computed by any of the four packages.) There is good agreement among the values in each row (not between rows, as some packages compute the likelihood up to a constant term). If anything, it looks as if the parameter estimates computed by **FKF** give a slightly lower log likelihood, whichever package is used to evaluate it, but the difference is hardly of any consequence.

5. Discussion

R now offers a choice of packages for state space modelling and estimation. Each package has different strengths and emphasizes different tasks.

Both **sspir** and **dlm** offer a nice user interface, following different approaches. For univariate time series, the formula interface of **sspir** may be the easiest and fastest to learn and use. The approach taken in **dlm** is quite general and elegant and enables the user to build a model "adding" blocks with the redefined `'+'` operator and the outer sum %+%. Both offer ancillary functions to help in the model specification task.

Packages **FKF** and **KFAS** offer a much more austere interface: the user is left alone to produce the inputs to the filtering routines. **FKF** is indisputably fastest. **KFAS** offers a rich set of functions for filtering, smoothing, simulation smoothing, etc. and has quite acceptable speed. It is unique in offering exact initial diffuse conditions for all or part of the components in the state vector: all other packages resort to a "vague" proper a priori distribution for the initial state vector when there is no a priori information.

KFAS is also unique in its sequential processing approach, although in the examples examined this did not appear to give it an advantage. It is possible that the need to "decorrelate" the inputs partly offsets the advantages of sequential processing. It is also possible, given the extensive use of BLAS routines for all vector and matrix operations, that the increased number of calls (by a factor of p, the dimension of $\boldsymbol{y_t}$) pushes overhead which partially negates the greater efficiency of sequential processing.

dse and **FKF** provide the fastest Kalman filtering routines, although in the former case neither missing values nor time-varying system matrices are supported.

Algorithm wise, **dlm** is the only one to use a form of square root algorithm, based on the singular value decomposition of the covariance matrix of the state vector. It has performed quite reliably in the examples reported and others. Even though it uses an algorithm requiring in principle more effort than a standard covariance filter, it takes only about 2-3 times longer than **KFAS** in the examples reported in Table 2.

We expected the choice of algorithm to have a greater impact, particularly since the covariance filter implemented by **sspir**, **FKF** and **KFAS** does not make use of (9), but rather of (8). As it happens, the parameter estimation results in the regression example check to 5-6 decimal places, and in the other examples run we found differences only after the second decimal place, like those reported in Table 4. When the likelihood optimization does not stop abnormally, the four packages appear to give results equivalent for all practical purposes.

We found occasional problems when one or several parameters approached the feasibility boundary. This is quite common with basic structural models, where one or more noise variances are frequently estimated at zero (cf. Ripley (2002)).

Although the packages analyzed are already of high quality, there is room for improvement. We have mentioned before the approach taken by **sspir** resorting extensively to function calls which make the code modular and easy to read, but add overhead. In the compiled code of the other packages, there could be a case for inlining some operations presently handled by BLAS calls. Above all, achieving speeds of the kind exhibited by function `StructTS` probably requires code which handles different problems differently. For instance, the (quite common) case of a time invariant $\boldsymbol{T_t}$ (or $\boldsymbol{F_t}$) equal to the unit matrix simplifies the transition from $\boldsymbol{a_{t|t-1}}$ to $\boldsymbol{a_t}$ in Equation 5, saving a matrix multiplication (and further simplifications would be realized in Equation 6). Similar savings can be realized with specialized code for other often-occurring particular cases of sparse transition matrices.

As a further example, time varying system matrices are implemented, e.g., in **KFAS**, by three dimensional arrays, the third index being time. In the event of, say, two regimes requiring some of the system matrices to have one of two "states", we have nevertheless to fill the entire three dimension array, with third dimension equal to N, the length of the series.

This may be relatively innocuous efficiency wise: but if the time varying matrix is $\boldsymbol{V_t}$ and happens to be non-diagonal, we will be decorrelating the inputs over and over again – and this may be quite time consuming.

At the time being, most users will probably want to use more than one of the packages examined. However uncomfortable it may be at times, this allows the user to reap the benefits from the best features of each package, and have access to open source tools for state space modelling and estimation that could only be dreamt of a few years back.

Acknowledgments

I am indebted to the authors of the packages mentioned above (Paul Gilbert, Claus Dethlefsen, Giovannis Petris, David Lüthi and Jouni Helske), for help, comments or clarifications offered at various points in time. I am also indebted for comments to Ignacio Díaz-Emparanza, two anonymous referees and the editor of the journal. The usual disclaimer applies. Partial support from grants ECO2008-05622 (MCyT) and IT-347-10 (Basque Government) is gratefully acknowledged.

References

Anderson BDO, Moore JB (1979). *Optimal Filtering*. Prentice-Hall.

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Croz JD, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999). *LAPACK Users' Guide*. 3rd edition. SIAM.

Bierman GJ (1977). *Factorization Methods for Discrete Sequential Estimation*. Dover Publications.

Bucy RS, Joseph PD (1968). *Filtering for Stochastic Processes with Application to Guidance*. Interscience, New York.

Byrd RH, Lu P, Nocedal J, Zhu C (1995). "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing*, **16**, 1190–.1208.

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**(3), 541–553.

Chambers JM (2008). *Software for Data Analysis: Programming with R*. Springer-Verlag, New York.

Chambers JM, Hastie TJ (1992). *Statistical Models in S*. Wadsworth & Brooks/Cole, Pacific Grove.

Cowpertwait PSP, Metcalfe AV (2009). *Introductory Time Series with R*. Springer-Verlag, New York.

de Jong P (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. URL http://www.jstatsoft.org/v16/i01/.

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

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

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

Eubank RL (2006). *A Kalman Filter Primer.* Chapman & Hall/CRC.

Frühwirth-Schnatter S (1994). "Data Augmentation and Dynamic Linear Models." *Journal of Time Series Analysis*, **15**(2), 183–202.

Gentle JE (2007). *Matrix Algebra: Theory, Computations, and Applications in Statistics.* Springer-Verlag, New York.

Gilbert P (2010). ***EvalEst**: Dynamic Systems Estimation – Extensions.* R package version 2010.02-1, URL http://CRAN.R-project.org/package=EvalEst.

Gilbert PD (1993). "State Space and ARMA Models: An Overview of the Equivalence." *Technical Report 94-3*, Bank of Canada. URL http://www.bank-banque-canada.ca/pgilbert/.

Gilbert PD (2011). *Brief User's Guide: Dynamic Systems Estimation.* R package vignette, version 2009.12-1, URL http://CRAN.R-project.org/package=dse.

Grewal MS, Andrews A (2001). *Kalman Filtering : Theory and Practice Using MATLAB.* Second edition. John Wiley & Sons. ISBN 0-471-39254-5.

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 0.6.0, URL http://CRAN.R-project.org/package=KFAS.

Koopman S, Durbin J (2000). "Fast Filtering and Smoothing for Non-Stationary Time Series Models." *Journal of the American Statistical Association*, **92**, 1630–1638.

Koopman SJ (1997). "Exact Initial Kalman Filtering and Smoothing for Nonstationary Time Series Models." *Journal of the American Statistical Association*, **92**(440), 1630–1638.

Koopman SJ, Durbin J (2003). "Filtering and Smoothing of State Vector for Diffuse State-Space Models." *Journal of Time Series Analysis*, **24**(1), 85–98.

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

Lawson CL, Hanson RJ (1974). *Solving Least Squares Problems.* Prentice-Hall, Englewood Cliffs.

Luethi D, Erb P, Otziger S (2010). ***FKF**: Fast Kalman Filter.* R package version 0.1.1, URL http://CRAN.R-project.org/package=FKF.

Petris G (2010). "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.

R Development Core Team (2010). *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/`.

Schweppe FC (1965). "Evaluation of Likelihood Functions for Gaussian Signals." *Information Theory*, **11**, 61–70.

Shumway RH, Stoffer DS (2006). *Time Series Analysis and Its Applications – With R Examples*. Springer-Verlag, New York.

Simon D (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons.

Strickland CM, Turner IW, Denham R, Mengersen KL (2009). "Efficient Bayesian Estimation of Multivariate State Space Models." *Computational Statistics & Data Analysis*, **53**, 4116–4125.

The Institute of Statistical Mathematics (2009). ***timsac**: Time Series Analysis and Control Package*. R package version 1.2.1, URL `http://CRAN.R-project.org/package=timsac`.

Tunnicliffe-Wilson G, Wang Z (2006). ***cts**: Continuous Time Autoregressive Models*. R package version 1.0-1, URL `http://CRAN.R-project.org/src/contrib/Archive/cts/`.

Vanbegin M, Verhaegen M (1989). "Algorithm 675: Fortran Subroutines for Computing the Square Root Covariance Filter and Square Root Information Filter in Dense or Hessenberg Forms." *ACM Transactions on Mathematical Software*, **15**(3), 243–256.

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

Whaley RC, Petitet A, Dongarra JJ (2001). "Automated Empirical Optimizations of Software and the ATLAS Project." *Parallel Computing*, **27**(1–2), 3–35.

Zhang Y, Li XR (1996). "Fixed-Interval Smoothing Algorithm Based on Singular Value Decomposition." In *Proceedings of the 1996 IEEE International Conference on Control Applications*, pp. 916–921.

Affiliation:

Fernando Tusell
Department of Statistics and Econometrics (EA-III)
Facultad de CC.EE. y Emp. – UPV/EHU

Avda. Lehendakari Aguirre, 83
E-48015 Bilbao, Spain
E-mail: fernando.tusell@ehu.es
URL: http://www.et.bs.ehu.es/~etptupaf/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/

Volume 39, Issue 2 *Submitted:* 2010-01-12
March 2011 *Accepted:* 2010-08-17