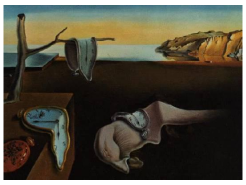

Journal of Statistical Software

October 2010, Volume 36, Issue 12. *http://www.jstatsoft.org/*

An R Package for Dynamic Linear Models

Giovanni Petris
University of Arkansas

Abstract

We describe an R package focused on Bayesian analysis of dynamic linear models. The main features of the package are its flexibility to deal with a variety of constant or time-varying, univariate or multivariate models, and the numerically stable singular value decomposition-based algorithms used for filtering and smoothing. In addition to the examples of "out-of-the-box" use, we illustrate how the package can be used in advanced applications to implement a Gibbs sampler for a user-specified model.

Keywords: state space models, Kalman filter, forward filtering backward sampling, Bayesian inference, R.

1. Overview

State space models provide a very rich class of models for the analysis and forecasting of time series data. They are used in a large number of applied areas outside statistics, such as econometrics, signal processing, genetics, population dynamics. Dynamic linear models (DLMs) are a particular class of state space models that allow many of the relevant inferences to be carried out exactly using the Kalman filter—at least in the case of a completely specified model. At the same time, they are flexible enough to capture the main features of a wide array of different data. Estimating unknown parameters in a DLM requires numerical techniques, but the Kalman filter can be used in this case as a building block for evaluating the likelihood function or simulating the unobservable states.

The R (R Development Core Team 2010) package **dlm** (Petris 2010) provides an integrated environment for Bayesian inference using DLMs. While the user can find in the package functions for Kalman filtering and smoothing, as well as maximum likelihood estimation, we believe the main feature lies in the tools that the package provides for simulation-based Bayesian inference. The package is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=dlm`.

DLMs are defined in Section 2, but a detailed discussion of the model and its properties are beyond the scope of this article. For an introduction, the reader can consult West and Harrison (1997) or Petris, Petrone, and Campagnoli (2009).

The layout of the paper is as follows. Section 2 deals with model specification in R. In Section 3 we briefly touch on how the Kalman filter and smoother are implemented in **dlm**. We assume the reader is familiar with filtering and smoothing for DLMs. Section 4 covers parameter estimation, from both a maximum likelihood and a Bayesian perspective. In Section 5 we discuss differences and similarities between **dlm** and other functions already available in R, or included in other contributed packages, for DLM analysis. Finally, Section 6 concludes the paper.

2. Model specification

A DLM is specified by the set of equations

$$\begin{aligned} y_t &= F_t\theta_t + v_t, & v_t &\sim \mathcal{N}(0, V_t), \\ \theta_t &= G_t\theta_{t-1} + w_t, & w_t &\sim \mathcal{N}(0, W_t), \end{aligned} \tag{1}$$

$t = 1, \ldots$. The specification of the model is completed by assigning a prior distribution for the initial (pre-sample) state θ_0. This is assumed to be normally distributed with mean m_0 and variance C_0. In (1) y_t and θ_t are m- and p-dimensional random vectors, respectively, while F_t, G_t, V_t, and W_t are real matrices of the appropriate dimension. The sequences (v_t) and (w_t) are assumed to be independent, both within and between, and independent of θ_0. In most applications, y_t is the value of an observable time series at time t, while θ_t is an unobservable state vector. The DLM provides a very rich and flexible family of models though it is only a highly special case of the more general class of state space models. In package **dlm** we tried to impose as few restrictions as possible on the types of models that can be specified. In particular, DLMs for multivariate observations and nonconstant models can be easily defined within the framework of the package, as we will illustrate below.

An R object representing a DLM can be defined using the function `dlm`. Before discussing the general case, let us start with the simpler case of constant DLMs. These are models of the form (1) in which the real matrices F_t, G_t, V_t, and W_t are time-invariant. When this is the case, we will drop the subscript t in the notation. Clearly, a constant DLM is completely specified by the matrices F, G, V, W, C_0 and the vector m_0. Accordingly, the general creator function `dlm` can take those matrices as arguments. As a very simple example, consider a polynomial model of order one, or random walk plus noise model, which is a univariate model with univariate state vector defined by

$$\begin{aligned} y_t &= \theta_t + v_t, & v_t &\sim \mathcal{N}(0, V), \\ \theta_t &= \theta_{t-1} + w_t, & w_t &\sim \mathcal{N}(0, W). \end{aligned}$$

Suppose one wants to define in R such a model, with $V = 3.1$, $W = 1.2$, $m_0 = 0$, and $C_0 = 100$. This can be done as follows:

```
R> myModel <- dlm(FF = 1, V = 3.1, GG = 1, W = 1.2, m0 = 0, C0 = 100)
```

Extractor and replacement functions for the different components of a model are available. For example, one can print the value of the matrix V using the command `V(myModel)`. If 4.1

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

Table 1: Creator functions for special models.

instead of 3.1 was the value one intended for V, one can fix it with the assignment `V(myModel) <- 4.1`. Polynomial models are so common in applications that a special function has been defined to specify such models in a simplified way. So, the same model can be alternatively defined in R as

```
R> myModel <- dlmModPoly(order = 1, dV = 3.1, dW = 1.2, C0 = 100)
```

In addition to `dlmModPoly`, package **dlm** provides other functions to create standard types of DLMs. They are summarized in Table 1. With the exception of `dlmModARMA`, which also handles the multivariate case, the other creator functions are limited to the case of univariate observations. More complicated DLMs can be explicitly defined using the general function `dlm`. Note also that `dlmModARMA` produces one DLM representation of the specified ARMA process; other representations exist and they can be specified using the more general `dlm`.

A convenient feature of DLMs is that sums and outer sums of DLMs can be defined in a natural way, allowing the user to specify complex models from basic building blocks. A standard example is a DLM representing a time series for quarterly data, in which one wants to include a local linear trend (polynomial model of order 2) and a seasonal component. Such a model can be set up in R very simply in the following way:

```
R> myModel <- dlmModPoly(2) + dlmModSeas(4)
```

In the code above we have used the default values of most of the arguments of `dlmModPoly` and `dlmModSeas`. The user should however keep in mind that the default values, in particular those of the variances V and W, are typically not meaningful for the particular data set at hand, and should be considered as place-holders that allow the user to quickly set up a model like `myModel` in the code above. More meaningful values can be either specified in the call to `dlmMod*` or explicitely set after the model is defined, using the replacement functions provided by the package.

Two DLMs, modeling an m_1- and an m_2-variate time series respectively, can also be combined into a unique DLM for $(m_1 + m_2)$-variate observations. We can think of this as a kind of outer sum. For example, two univariate models for a local trend plus a quarterly seasonal component as the one described above can be combined as follows (here $m_1 = m_2 = 1$):

```
R> bivarMod <- myModel %+% myModel
```

Also in this case the user has to be careful to specify meaningful values for the variances of the resulting model after model combination. Both sums and outer sums of DLMs can be iterated and they can be combined to allow the specification of more complex models for multivariate data from simple standard univariate models.

We describe next how it is possible to specify a time-varying DLM, where at least one of the matrices or variances defining the model is not constant. In package **dlm** we took an approach similar to the one used in the **S+FinMetrics** module of S-PLUS (see Zivot and Wang 2005). A `dlm` object may contain, in addition to the components `FF`, `V`, `GG`, `W`, `m0`, and `C0` described above, one or more of `JFF`, `JV`, `JGG`, `JW`, and `X`. While `X` is a matrix used to store all the time-varying elements of the model, the `J*` components are indicator matrices whose entries signal whether an element of the corresponding model matrix is time-varying and, in case it is, where to retrieve its values in the matrix `X`. For example, in the standard DLM representation of a simple linear regression models, the state vector is $\theta_t = (\beta_{t0}, \beta_{t1})'$, the vector of regression coefficients, which may be constant or time-varying. The system matrix G_t is the 2×2 identity matrix and the observation matrix is $F_t = [1 \; x_t]$, where x_t is the value of the covariate for observation y_t. Assuming the variances V_t and W_t are constant, the only time-varying element of the model is the $(1, 2)$th entry of F_t. Accordingly, the component `X` in the `dlm` object will be a one-column matrix containing the values of the covariate x_t, while `JFF` will be the 1×2 matrix $[0 \; 1]$, where the ‘0’ signals that the $(1, 1)$th component of F_t is constant, and the ‘1’ means that the $(1, 2)$th component is time-varying and its values at different times can be found in the first column of `X`.

3. Kalman filtering and smoothing

Assuming that a DLM is completely specified, i.e., that there are no unknown parameters in its definition, one can use the well-known Kalman filtering and smoothing algorithms to obtain means and variances of the conditional distributions of the unobservable system states given the data. Let us recall that the filtering distribution of θ_t is the distribution of θ_t given $y_1, \ldots, y_t$, while the smoothing distribution of θ_t at time s is the conditional distribution of θ_t given $y_1, \ldots, y_s$, for $s \geq t$. Note that, under our modeling assumptions, all these distributions are Gaussian, hence completely determined by their mean and variance. Package **dlm** provides the function `dlmFilter` and `dlmSmooth` for filtering and smoothing.

It is well-known that a naive implementation of the Kalman filter and smoother may incur in numerical instability issues. These may lead, for example, to calculated variance matrices that are not postive semidefinite. Square-root filter and smoothers, based on the propagation of Cholesky decomposition of the variance matrices, provide more robust algorithms. Even more robust are the singular value decomposition-based algorithms proposed by Wang, Liber, and Manneback (1992) and Zhang and Li (1996), and used in package **dlm**. In this case the filtering and smoothing recursions consist in the sequential calculation of a singular value decomposition of the relevant variance matrices. As a—perhaps annoying—consequence, all the variance matrices returned by `dlmFilter` and `dlmSmooth` are expressed in terms of their singular value decomposition. In order to recover the variance matrices in their usual form, one can use the provided utility function `dlmSvd2var`.

4. Parameter estimation

Clearly, in any realistic statistical application using DLMs, one has to estimate unknown parameters of the DLM. In the simplest and most common cases, for example, such as a polynomial model with the possible addition of a seasonal factor model, these may be observation or system variances only, but, in general, unknown parameters may appear anywhere in the matrices defining the DLM. Package **dlm** provides tools for both Bayesian inference and maximum likelihood estimation of unknown model parameters. We will discuss maximum likelihood first, followed by Bayesian inference for DLMs.

4.1. Maximum likelihood

For maximum likelihood estimation, package **dlm** relies on `optim`, the excellent optimizer available in R. The `dlm` function for maximum likelihood estimation is `dlmMLE`, which calls the `optim` subroutines. A call to the function may look like this:

```
R> dlmMLE(y = myData, parm = init, build = myFun)
```

Argument `y` is a (univariate or multivariate) time series object or a vector/matrix of observations, while `parm` is a vector of initial values for the unknown parameters. The third argument in the example above, `build`, is a user-defined function that takes a parameter vector as first argument and returns a `dlm` object. Loosely speaking, the following procedure is what `dlmMLE` essentially does:

1. Define a target function by compounding the `build` argument with `dlmLL()`, which evaluates the (negative log of the) joint density of the observations, thus defining the negative loglikelihood function;

2. Call `optim` to minimize the negative loglikelihood function defined in 1.

As a specific example, consider a random walk plus noise model, which has two unknown variance parameters. In this case the `build` argument could be defined as follows

```
R> myFun <- function(x) return(dlmModPoly(1, dV = exp(x[1]), dW = exp(x[2])))
```

Note that we have parametrized the variances in term of their logs in order to avoid specifying bounds on the parameter values. One can specify bounds using the standard arguments `lower` and `upper` of `optim`. As a matter of fact, any arguments of `optim` can be specified in `dlmMLE`, which will pass them to `optim`.

As a less trivial example, we consider a bivariate time series of yearly average air temperatures. The first component is an average computed from land-based observation stations (Jones 1994). The second component is an average based on a number of marine-based stations (Parker, Folland, and Jackson 1995). These data are analyzed in Shumway and Stoffer (2006) using a DLM. We show below how to use package **dlm** to fit the model proposed by Shumway and Stoffer. Marginally, each component of the observed bivariate series is assumed to follow a random walk plus noise model. However, since the two series are essentially measurements of the same quantity, the state in the two marginal models is assumed to be the same, and the observation noises are taken to be correlated. Formally, we have a constant DLM specified by the following matrices:

$$F = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad G = \begin{bmatrix} 1 \end{bmatrix}.$$

The observation covariance matrix, V, and the system variance W need to be estimated from the data. In this case, since the model is not a standard one, we use the general creator `dlm` to define a `build` function, which we subsequently use to find the MLEs of the model parameters. In order to avoid an optimization problem with complicated constraints, we parametrize V in terms of the elements of its log-Cholesky decomposition (Pinheiro and Bates 1996). For W, we simply use its log.

```
R> buildTemp <- function(x) {
+    L <- matrix(0, 2, 2)
+    L[upper.tri(L, TRUE)] <- x[1 : 3]
+    diag(L) <- exp(diag(L))
+    modTemp <- dlm(FF = matrix(1, 2, 1), V = crossprod(L),
+      GG = 1, W = exp(x[4]), m0 = 0, C0 = 1e7)
+    return(modTemp)
+ }
R> y1 <- scan("http://www.stat.pitt.edu/stoffer/tsa2/data/HL.dat")
R> y2 <- scan("http://www.stat.pitt.edu/stoffer/tsa2/data/Folland.dat")
R> y <- ts(cbind(y1, y2), start = 1880)
R> fitTemp <- dlmMLE(y, parm = rep(0, 4), build = buildTemp,
+    hessian = TRUE, control = list(maxit = 500))
```

The Hessian matrix at the minimizer of the negative loglikelihood can be used to estimate asymptotic standard errors or, more generally, asymptotic covariance matrices of the MLE. The fitted model is best set up in R using the `build` function itself, together with the MLE of the parameters:

```
R> modTemp <- buildTemp(fitTemp$par)
```

At this point one can use the fitted model for smoothing or forecasting, or just take a look at the parameter estimates. The estimated V and W are as follows.

```
R> V(modTemp)

            [,1]        [,2]
[1,] 0.019503957 0.006512939
[2,] 0.006512939 0.005386751

R> drop(W(modTemp))

[1] 0.002633201
```

The user should be aware that the likelihood function for a general DLM may present local maxima. Therefore, we suggest, as a minimal check, to call `dlmMLE` several times with different starting values. Furthermore, it is not uncommon for the likelihood function to be relatively flat around its maximum, implying that the combination of data and model does not allow for an accurate estimation of the unknown parameters. The Hessian matrix at the maximum can be used to assess the presence and extent of this phenomenon, although we have noticed that the Hessian obtained from dlmMLE is subject to numerical instability and users on different platforms may obtain different results. (The function numDeriv:::hessian offers a stabler alternative for the numerical evaluation of the Hessian matrix).

To illustrate this point further, let us consider again the random walk plus noise model together with the famous Nile river level data (the data set is available in R as Nile). Parametrizing the model directly in terms of the two variances, one can obtain the MLEs as follows.

```
R> buildNile <- function(x) dlmModPoly(1, dV = x[1], dW = x[2])
R> fitNile <- dlmMLE(Nile, parm = rep(100, 2), build = buildNile,
+    lower = rep(1e-8, 2), hessian = TRUE)
R> fitNile$par

[1] 15099.795  1468.427
```

For this very simple example, the estimated asymptotic covariance matrix of the MLEs, obtained by inverting the Hessian returned by dlmMLE, is the following.

```
R> aVar <- solve(fitNile$hessian)
R> aVar

          [,1]      [,2]
[1,] 4226694.0 -571174.9
[2,] -571174.9 1028114.8

R> sqrt(diag(aVar))

[1] 2055.893 1013.960
```

The standard errors are large. A 95% confidence interval for W based on standard asymptotics even includes the value $W = 0$ – and clearly the smoothed level obtained setting $W = 0$ is going to look very different from the one obtained setting W to its MLE. This suggests that plugging the MLE into the model and proceeding with the analysis, ignoring in this way the uncertainty in the parameter estimates, is probably not the best thing to do, from a statistical standpoint. If one has such large standard errors in a simple model like the random walk plus noise, one can easily imagine that the situation will be even worse for more complicated models. As a matter of fact, in several multivariate examples finding MLEs can be a numerically challenging and time-consuming exercise. Moreover, even when one is able to compute a MLE – and to be reasonably confident it is not just a local maximum of the likelihood function – the problem remains of how to deal with the large uncertainty in the estimates. In practice filtering and smoothing estimates of the unobservable states, together with their variances, are customarily computed by plugging in the model the MLE of the unknown parameters, completely ignoring the large uncertainty typically associated to the estimates. A reasonable way to cope with this issue is to take a Bayesian approach, in which posterior distributions – whether of states or parameters – incorporate all the uncertainty about the quantities of interest.

4.2. Bayesian inference

While for a few very special cases (see West and Harrison 1997, sections 4.5 and 16.4) it is possible to compute the posterior distribution of states and unknown parameters in closed form, one has in general to resort to Monte Carlo methods to draw a sample from the posterior distribution of interest. Currently, the most commonly used approach to do so is to implement a Gibbs sampler which draws in turn from the conditional distribution of: (i) the parameters given the data and unobserved states, and; (ii) the states given the data and the parameters. The first step can be further broken down into several Gibbs steps, each drawing a specific subset of parameters given everything else (sampling from a full conditional distribution). The states can be considered latent variables: their inclusion in the Gibbs simulation scheme usually simplifies the implementation and speeds up the mixing of the resulting Markov chain. While drawing the parameters dependends heavily on the model and priors used, the state sequence can be generated from its full conditional distribution using the so-called forward filtering backward sampling (FFBS) algorithm (Carter and Kohn 1994; Früwirth-Schnatter 1994; Shephard 1994). The algorithm is basically a simulation version of the Kalman smoother, consisting in running the Kalman filter first, followed by a backward recursion to generate all the states from the final time T to time 0. Package **dlm** provides an implementation of the backward-sampling portion of the algorithm in the function `dlmBSample`. Together, `dlmFilter` and `dlmBSample` can be used to implement FFBS. It must be stressed that FFBS can be also helpful when, in a model with no unknown parameters, one is interested in the posterior distribution, or some summaries thereof, of a nonlinear functional of the state process. As a very simple example, consider again the Nile River level data set, modelled as a random walk plus noise DLM. The MLE of the system variance W is about 1470 and that of the observation variance V is about 15100. For purpose of illustration, we consider these values as known. The one-dimensional state θ_t represents in this model the "true" level of the river at time t. Suppose one is interested in assessing the largest year-to-year variation, i.e., $\max_t(\theta_t - \theta_{t-1})$. Let us denote this quantity η. Since it is difficult to derive the exact distribution of η, we obtain 1000 simulated samples as follows.

```
R> modNile <- dlmModPoly(1, dV = 15100, dW = 1470)
R> nileFilt <- dlmFilter(Nile, modNile)
R> eta <- replicate(1000, max(diff(dlmBSample(nileFilt))))
```

From the simulated sample, Monte Carlo estimates of quantities like the expected value or the variance of the posterior distribution can be computed in the usual way.

In addition to `dlmBSample`, which can be used as a building block in a user-defined Gibbs sampler, package **dlm** provides a function, `dlmGibbsDIG`, that runs a Gibbs sampler for a particular class of univariate DLMs. The defining properties of these models are the following: (i) the only unknown parameters are in the variances V and W; (ii) W is a diagonal matrix, and; (iii) the unknown variances have independent inverse Gamma prior distributions. We will call a model satisfying these properties a *d-inverse-gamma* model, where d refers to the total number of unknown variances in the model. To illustrate the usage of the function `dlmGibbsDIG`, let us consider the built-in data set `UKgas`, containing quarterly UK gas consumption from 1960 to 1986. From visual inspection of the data it seems that, on a log scale, a DLM obtained by adding a quarterly seasonal factor model to a local linear trend model should fit the data reasonably well. This is Harvey's (1989) so-called basic structural model.

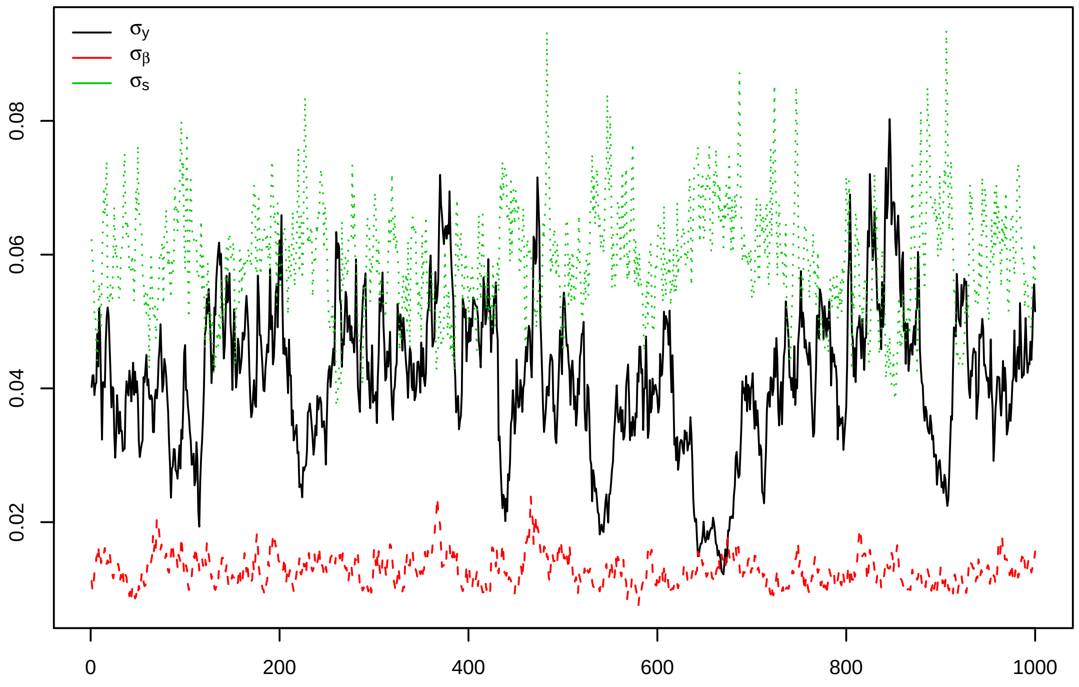

Figure 1: Trace plots of MCMC output.

The observation (F) and system (G) matrices of the model are

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2}$$

The unknown model parameters are the observation variance $V = \sigma_y^2$ and two elements of W, which is assumed to have the diagonal form

$$W = \mathrm{diag}(0,\, \sigma_\beta^2,\, \sigma_s^2,\, 0,\, 0). \tag{3}$$

We take independent inverse gamma priors for the three nonzero variances, with shape parameter $\alpha = 10^{-3}$ and rate parameter $\beta = 10^{-3}$. A Gibbs sampler for this model can be run in R as follows:

```
R> set.seed(999)
R> mcmc <- 1000
R> burn <- 500
R> outGibbsIRW <- dlmGibbsDIG(y = log(UKgas),
+    mod = dlmModPoly(2) + dlmModSeas(4), shape.y = 1e-3, rate.y = 1e-3,
+    shape.theta = 1e-3, rate.theta = 1e-3, n.sample = mcmc + burn,
+    ind = c(2, 3))
```

The argument `ind` specifies the position of the unknown variances on the diagonal of W. A trace plot of the simulated standard deviations, after burn-in, is provided in Figure 1. Package **dlm** also provides basic functionality for MCMC output summary. Ergodic means, i.e., simulation-based Bayes estimates, of the unknown standard deviations, together with their estimated Monte Carlo standard errors, can be obtained as shown below.

```
R> mcmcMean(with(outGibbsIRW, sqrt(cbind(V = dV[-(1 : burn)],
+    dW[-(1 : burn), ]))))

       V           W.2         W.3
  0.041549   0.012866   0.059212
 (0.002081) (0.000262) (0.000876)
```

Monte Carlo standard errors are estimated using Sokal's method (Sokal 1989; Green 2001)

4.3. Example: Outliers and structural breaks

The function `dlmGibbsDIG` in package **dlm** provides an out-of-the-box tool for Bayesian inference in a broad class of commonly used DLMs. For other DLMs, the package provides a couple of functions that can be used as building blocks for an appropriate Gibbs sampler. In addition to `dlmBSample`, used to implement FFBS and described in Section 4.2, in package **dlm** the user can find an R port (function `arms`) of the original C code by W. Gilks that implements adaptive rejection Metropolis sampling (ARMS, Gilks, Best, and Tan 1995). This can be very useful when, within a Gibbs sampler, one needs a draw from a nonstandard distribution. Function `arms` is an improved version of the original ARMS algorithm, allowing the target density to be multivariate. See the examples in the help file. As an illustration, we will show in the present section how to use the tools offered by package **dlm** to set up a Gibbs sampler for a structural model that accounts for possible outliers and structural breaks. The resulting function implementing the Gibbs sampler is shown in the supplementary file '`dlmGibbsDIGt.R`'.

Within a simulation-based Bayesian approach, the most common way to account for outliers in a Gaussian model is to replace the normal distribution with a scale mixture of normals in such a way that, conditionally on the scale parameter, the observations are again normally distributed while, marginally, they have a Student-t distribution. Hyperpriors can be specified for the degrees of freedom of the resulting Student-t distributions. Of course, other parameters may be included in the model as well, in which case the argument outlined above holds conditionally on these additional parameters.

For a DLM like (1), assuming that y_t is univariate and the elements of each w_t are independent, i.e., that W_t is diagonal, we can apply the preceding approach to v_t and to each component of w_t to obtain a model that accounts for possible outliers in the observation process as well as in the state process. The latter can be thought as structural breaks. In order to describe the model in more detail, we need to introduce new notations. Let $\mathcal{G}(\alpha, \beta)$ denote the gamma distribution with mean α/β, so that the gamma distribution with mean a and variance b is $\mathcal{G}(a^2/b, a/b)$. In addition, let $\mathcal{D}ir(\alpha)$ denote the Dirichlet distribution with (vector) parameter α, and $\mathcal{S}am(\mathcal{X}, \pi)$ the distribution of a random element of the finite set $\mathcal{X}$, selected according to the probability vector π. Finally, we denote by $\mathcal{U}nif(a, b)$ the uniform distribution on the interval (a, b). We will focus on the observational variances first, extending our model to system variances in a second time. Given a common scale factor λ^{-1}, and individual scale factors ω_t^{-1} $(t = 1, 2, \dots)$, we assume that

$$v_t \sim \mathcal{N}(0, (\lambda\omega_t)^{-1}). \tag{4a}$$

For the ω_t's we assume independent gamma priors:

$$\omega_t \overset{indep}{\sim} \mathcal{G}(\nu_t/2, \nu_t/2). \tag{4b}$$

Up to this point, the specification is equivalent to assuming that $\lambda^{\frac{1}{2}} v_t$ has a Student-t distribution with ν_t degrees of freedom. Moving up in the hierarchical prior, we make the following distributional assumptions on the model parameters:

$$\lambda \sim \mathcal{G}(a^2/b, a/b), \tag{4c}$$

$$\nu_t \overset{indep}{\sim} \mathcal{S}am(\mathfrak{N}, \pi), \tag{4d}$$

$$a \sim \mathcal{U}nif(0, A), \tag{4e}$$

$$b \sim \mathcal{U}nif(0, B), \tag{4f}$$

$$\pi \sim \mathcal{D}ir(\alpha). \tag{4g}$$

We consider a finite set of possible degrees of freedom $\mathfrak{N} = \{n_1, \dots, n_K\}$, although extensions to continuous ν_t's are easy to devise. For each t, the posterior distribution of ω_t (or ν_t) contains the relevant information about the "outlying-ness" of the observation y_t: values of ω_t smaller than one flag possible outliers. A similar hierarchical prior can be specified for each series of nonzero components of w_t. Let w_{ti} be the ith element of w_t. For the series $(w_{ti})_{t\geq 0}$ we assume a hierarchical prior of the same form as (4a)–(4c). As a notational device, we will use the same symbols as in (4a)–(4c), with an additional subscript "θi"; with this convention we have a prior for the w_{ti}'s defined hierarchically in terms of

$$\lambda_{\theta i},\ \omega_{\theta i,t}\ \nu_{\theta i,t},\ a_{\theta i},\ b_{\theta i},\ \pi_{\theta i},\ A_{\theta i},\ B_{\theta i}.$$

Including the unobservable states as latent variables, all the full conditional distributions are easy to derive and to sample from, with the exception of that of a, b, and $a_{\theta i}, b_{\theta i}$. The full conditional of (a, b) is

$$p(a, b|\dots) \propto \mathcal{G}(\lambda; a, b), \tag{5}$$

where, for every α and β, $\mathcal{G}(\cdot; \alpha, \beta)$ denotes the density of the $\mathcal{G}(\alpha, \beta)$ distribution. Similar expressions hold for the pairs $(a_{\theta i}, b_{\theta i})$ for every i. To draw from these nonstandard distributions we use `arms` on each pair (a, b), $(a_{\theta i}, b_{\theta i})$. A detailed derivation of all the full conditional distributions of the model can be found in Petris *et al.* (2009). We tested the model described above on the seasonally adjusted monthly index of US industrial production of consumer goods[1]. The data are available through the Federal Reserve website at the URL `http://research.stlouisfed.org/fred2/series/IPCONGD`. A look at the time series plot shows that, on a log scale, a local linear trend model, with possible outliers and structural breaks, should give a reasonable fit. The Gibbs sampler was run with the following call to `dlmGibbsDIGt`:

[1]This series is updated frequently; at the time of writing it ended in February 2009.

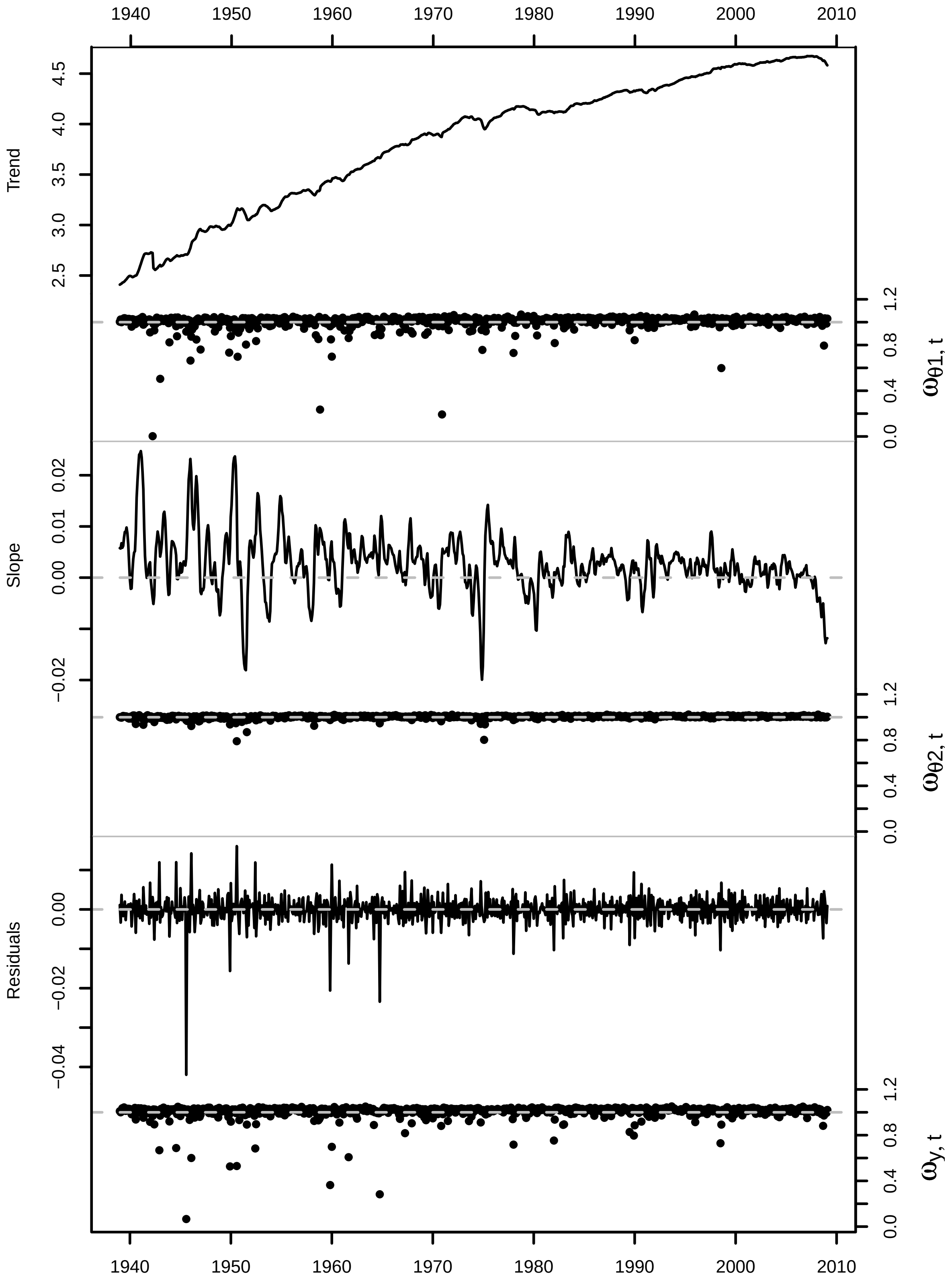

Figure 2: Outlier and structural breaks in industrial production of consumer goods.

```
R> mc <- 2000
R> burn <- 1000
R> gibbsOut <- dlmGibbsDIGt(yM, mod = dlmModPoly(2), A_y = 5e4, B_y = 5e4,
+    ind = 1 : 2, save.states = TRUE, n.sample = mc + burn, thin = 4)
```

Standard convergence diagnostics and visual assessment of trace plots did not raise any reason for concern, so we proceded and used the output of the Gibbs sampler for posterior inference. Graphical summaries are included in Figure 2. The residuals in the bottom panel are defined as $\hat{\epsilon}_t = y_t - \mathsf{E}(F\theta_t|y_{1:T})$. By looking at the residuals and the $\omega_{y,t}$'s, it can be seen that there are a few outliers, some of them fairly extreme (such as those occurred in August 1945 and October 1964), and most of them—the large ones in particular— negative. The trend of the series (top panel of Figure 2) shows some abrupt changes. The most dramatic one, with an estimated $\omega_{\theta 1,t}$ of 0.0025, in April 1942, and a slightly milder one in December 1970 ($\hat{\omega}_{\theta 1,t} = 0.146$). The slope, on the other hand, appears to have been fairly stable—dynamically changing, but in a smooth fashion, without sudden jumps. The current economic downturn, for example, is reflected in a slope which is becoming, month after month, more and more negative.

5. Comparison with other implementations

In this section we discuss some functions and packages available in, or for, R that can be used to make inference in a DLM. For a more extensive comparison of the different packages, see Tusell (2010). We will first focus on the functions that are included in the standard distribution of R, and then move to other functions available in contributed packages.

Function `StructTS`. This function computes the MLE of the variance parameters and the filtered means of the state vectors for a basic structural model, i.e., a constant DLM consisting of a random walk plus noise, a local linear trend, or a local linear trend plus a seasonal component. Smoothed means of the state vectors can be obtained via a call to `tsSmooth`. Forecasts, together with their standard errors, are available by calling `predict` on the fitted model (an object of class `StructTS`). The main function `StructTS`, together with the method functions available for objects of class `StructTS`, provides a very reliable tool to analyze time series using basic structural models. The main limitations are that only univariate time series can be analyzed and only using DLMs of very special types—basic structural models. Moreover, maximum likelihood is the only method available for parameter estimation.

Function `KalmanLike`. This function, together with `KalmanRun`, `KalmanSmooth`, and `KalmanForecast`, provides more flexibility in state space modeling. As a matter of fact, they provide the engine behind `StructTS` and the related functions mentioned above. They can treat more general DLMs, as long as they are constant and univariate. In using them directly, some care needs to be paid as, according to their help page, the functions only perform minimal checks on their arguments and `KalmanLike` may change the value of its arguments.

Package dse. Package **dse** (Gilbert 2009) provides a powerful set of tools to estimate and work with multivariate ARMA and linear state space models, within a frequentist framework.

Beside the different "philosophical" slant, for the pragmatic user the main advantage offered by package **dlm** is the possibility of specifying time-varying models—in **dse** only constant models are allowed. On the other hand, for constant models, **dse** provides several alternative estimation methods for unknown model parameters, in addition to maximum likelihood. In hard estimation problems, these can be used to find initial parameter values for maximum likelihood estimation. This may be potentially useful even for the user of package **dlm** interested in maximum likelihood estimation.

Package sspir. The contributed package **sspir** (Dethlefsen and Lundbye-Christensen 2006) has functions for approximate filtering and smoothing of linear state space models with normally distributed states and univariate observations with distribution in an exponential family, such as Poisson or Binomial. Models of this kind are usually called dynamic generalized linear models (DGLMs), and are used to study time series of counts or proportions. Although focused on DGLMs, package **sspir** provides functions for filtering and smoothing of univariate or multivariate DLMs, both constant and time varying. In this area, at least for completely specified DLMs, there is some overlap with the functionality of package **dlm**, although implementation details differ substantially. An informal comparison of execution times shows that the filtering routine in package **sspir** is about 12 times slower than the corresponding one in package **dlm**. For the smoothing, package **dlm** is only about 7 times faster. However, we think the main advantage of package **dlm** over **sspir** for DLM analysis is that the former provides an integrated environment in which unknown parameters can be estimated by maximum likelihood or Bayesian inference, while the latter requires all model parameters to be known.

6. Conclusions

In the previous sections we have illustrated the main features of the R package **dlm** for Bayesian and likelihood analysis of DLMs. Within this class, **dlm** is very flexible in the types of model that can be specified: essentially any constant or time-varying DLM, univariate or multivariate, over a finite horizon can be defined within the package framework. Moreover, simplified constructors for standard DLMs are provided, and models can be combined with sums or outer sums. This freedom allows the researcher to concentrate on substantive issues, without being limited by the constraints imposed by the software. (Of course, this is a relative freedom that can be enjoyed only within the DLM class.) In view of the great flexibility in the specification of the model, DLMs that are numerically unstable with respect to the standard filtering and smoothing procedures may result. This is the main reason behind the careful choice of the robust singular value decomposition-based algorithms for filtering and smoothing used in package **dlm**. A minor limitation of these algorithms is that they require the observation variances V_t to be nonsingular.

Nowadays, Bayesian inference can be applied for a huge class of models using a limited number of basic techniques—essentially, the Gibbs sampler and Metropolis–Hastings algorithm. However, although the general algorithms are always the same, they have to be taylored to the particular model/prior at hand, and this typically requires a human intervention. Package **dlm** provides a few general purpose functions that are intended to help building a Gibbs sampler in the DLM framework. In fact, `arms` can be useful for any kind of model—not just a DLM— and `dlmBSample` can be used to simulate from the full conditional distribution of the states of any DLM. The function `dlmGibbsDIG`, included in the package, can be used out of the box to perform Bayesian inference for structural time series. However, most of all, it provides an example of how a Gibbs sampler for a DLM can be set up in R using the tools available in the package. Another more advanced example is discussed in Section 4.3.

Aknowledgments

In the development of package **dlm** we have benefited from the suggestions and feedback of several users. In particular, we would like to thank Sonia Petrone, Michael Lavine, and Spencer Graves for their constructive input. Needless to say, without R in the first place, no contributed package would exist: our sincere thanks go to all the developers of R for their continued effort and support.

References

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

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`.

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

Gilbert PD (2009). *Brief User's Guide: Dynamic Systems Estimation (**dse**)*. R package version 2009.10-2, URL `http://CRAN.R-project.org/package=dse`.

Gilks WR, Best NG, Tan KKC (1995). "Adaptive Rejection Metropolis Sampling within Gibbs Sampling." *Applied Statistics*, **44**, 455–472. Corr: 1997, **46**, 541–542, with R.M. Neal.

Green PJ (2001). "A Primer on Markov Chain Monte Carlo." In OE Barndorff-Nielsen, DR Cox, C Klüppelberg (eds.), *Complex Stochastic Systems*. Chapman & Hall/CRC, Boca Raton.

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

Jones PD (1994). "Hemispheric Surface Air Temperature Variations: A Reanalysis and Update to 1993." *Journal of Climatology*, **7**, 1794–1802.

Parker DE, Folland CK, Jackson M (1995). "Marine Surface Temperature: Observed Variations and Data Requirements." *Climatic Change*, **31**, 559–560.

Petris G (2010). ***dlm**: Bayesian and Likelihood Analysis of Dynamic Linear Models.* R package version 1.1-1, URL `http://CRAN.R-project.org/package=dlm`.

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

Pinheiro JC, Bates DM (1996). "Unconstrained Parametrizations for Variance-Covariance Matrices." *Statistics and Computing*, **6**, 289–296.

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/.

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

Shumway RH, Stoffer DS (2006). *Time Series Analysis and Its Applications, with R Examples*. 2nd edition. Springer-Verlag, New York.

Sokal AD (1989). *Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms*. Cours de Troisiéme Cycle de la Physique en Suisse Romande. Lausanne.

Tusell F (2010). "State Space Modeling and Estimation in R." Submitted.

Wang L, Liber G, Manneback P (1992). "Kalman Filter Algorithm Based on Singular Value Decomposition." In *Proceedings of the 31st Conference on Decision and Control*, pp. 1224–1229.

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

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

Zivot E, Wang J (2005). *Modeling Financial Time Series with S-PLUS*. 2nd edition. Springer-Verlag, New York.

Affiliation:

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

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 36, Issue 12 *Submitted:* 2009-04-20
October 2010 *Accepted:* 2010-03-05