`dlm`: an `R` package for Bayesian analysis of Dynamic Linear Models

Giovanni Petris
University of Arkansas, Fayetteville AR

2009-01-14

1 Defining and manipulating Dynamic Linear Models

Package `dlm` focuses on Bayesian analysis of Dynamic Linear Models (DLMs), also known as linear state space models (see [H, WH]). The package also includes functions for maximum likelihood estimation of the parameters of a DLM and for Kalman filtering. The algorithms used for Kalman filtering, likelihood evaluation, and sampling from the state vectors are based on the singular value decomposition (SVD) of the relevant variance matrices (see [ZL]), which improves numerical stability over other algorithms.

1.1 The model

A DLM is specified by the following equations:

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

for $t = 1, \ldots, n$, together with a *prior* distribution for θ_0:

$$\theta_0 \sim \mathcal{N}(m_0, C_0).$$

Here y_t is an m-dimensional vector, representing the observation at time t, while θ_t is a generally unobservable p-dimensional vector representing the state of the system at time t. The v_t's are observation errors and the w_t's evolution errors. The matrices F_t and G_t have dimension m by p and p by p, respectively, while V_t and W_t are variance matrices of the appropriate dimension.

1.2 Defining DLMs with `dlm`

One of the simplest DLMs is the random walk plus noise model, also called first order polynomial model. It is used to model univariate observations, the state vector is unidimensional, and it is described by the equations

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

The model is *constant*, i.e., the various matrices defining its dynamics are time-invariant. Moreover, $F_t = G_t = [1]$. The only parameters of the model are the observation and evolution variances V and W. These are usually estimated from available data using maximum likelihood or Bayesian techniques. In package `dlm` a constant DLM is represented as a list with components `FF, V, GG, W`, having class `"dlm"`. A random walk plus noise model, with $V = 0.8$ and $W = 0.1$, can be defined in R as follows:

```
> dlm(FF = 1, V = 0.8, GG = 1, W = 0.1, m0 = 0, C0 = 100)
```

Note that the mean and variance of the prior distribution of θ_0 must be specified, as they are an integral part of the model definition. An alternative way to define the same models is

```
> dlmModPoly(order = 1, dV = 0.8, dW = 0.1, C0 = 100)
```

This function has default values for `m0, C0, dV` and `dW` (the last two are used to specify the *diagonal* of V and W, respectively). In fact it also has a default value for the *order* of the polynomial model, so the user must be aware of these defaults before using them light-heartedly. In particular, the default values for `dV` and `dW` should be more correctly thought as place holders that are there just to allow a complete specification of the model.

Consider the second-order polynomial model obtained as

```
> myMod <- dlmModPoly()
```

Individual components of the model can be accessed and modified in a natural way by using the exctractor and replacement functions provided by the package.

```
> FF(myMod)
```

```
     [,1] [,2]
[1,]    1    0
```

```
> W(myMod)
```

```
     [,1] [,2]
[1,]    0    0
[2,]    0    1
```

```
> m0(myMod)
```

```
[1] 0 0
```

```
> V(myMod) <- 0.8
```

In addition to `dlmModPoly`, `dlm` provides other functions to create DLMs of standard types. They are summarized in Table 1. With the exception of

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.

`dlmModARMA`, which handles also the multivariate case, the other creator functions are limited to the case of univariate observations. More complicated DLMs can be explicitely defined using the general function `dlm`.

1.3 Combining models: sums and outer sums

From a few basic models, one can obtain more general models by means of different forms of "addition". In general, suppose that one has k independent DLMs for m-dimensional observations, where the ith one is defined by the system

$$\begin{cases} y_t^{(i)} = F_t^{(i)}\theta_t^{(i)} + v_t^{(i)}, & v_t^{(i)} \sim \mathcal{N}(0, V^{(i)}) \\ \theta_t^{(i)} = G_t^{(i)}\theta_{t-1}^{(i)} + w_t^{(i)}, & w_t^{(i)} \sim \mathcal{N}(0, W^{(i)}) \end{cases} \tag{1}$$

$m_0^{(i)}$ and $C_0^{(i)}$ are the mean and variance of the initial state in DLM i. Note that the state vectors may have different dimensions $p_1, \ldots, p_k$ across different DLMs. Each model may represent a simple feature of the observation

process, such as a stochastic trend, a periodic component, and so on, so that $y_t = y_t^{(1)} + \cdot + y_t^{(k)}$ is the actual observation at time t. This suggests to combine, or add, the DLMs into a comprehensive one by defining the state of the system by $\theta_t' = \left(\theta_t^{(1)'}, \ldots, \theta_t^{(k)'}\right)$, together with the matrices

$$F_t = \left(F_t^{(1)} \mid \ldots \mid F_t^{(k)}\right), \qquad V_t = \sum_{i=1}^{k} V_t^{(i)},$$

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

$$m_0' = \left(m_0^{(1)'}, \ldots, m_0^{(k)'}\right), \qquad C_0 = \begin{bmatrix} C_0^{(1)} & & \\ & \ddots & \\ & & C_0^{(k)} \end{bmatrix}.$$

The form just described of model composition can be thought of as a sum of models. Package `dlm` provides a method function for the generic `+` for objects of class `dlm` which performs this sum of DLMs.

For example, suppose one wants to model a time series as a sum of a stochastic linear trend and a quarterly seasonal component, observed with noise. The model can be set up as follows:

```
> myMod <- dlmModPoly() + dlmModSeas(4)
```

The nonzero entries in the V and W matrices can be specified to have more meaningful values in the calls to `dlmModPoly` and/or `dlmModSeas`, or changed after the combined model is set up.

There is another natural way of combining DLMs which resembles an "outer" sum. Consider again the k models (1), but suppose the dimension of the observations may be different for each model, say $m_1, \ldots, m_k$. An obvious way of obtaining a multivariate model including all the $y_t^{(i)}$ is to consider the models to be independent and set $y_t' = \left(y_t^{(1)'}, \ldots, y_t^{(k)'}\right)$. Note that each $y_t^{(i)}$ may itself be a random vector. This corresponds to the definition of a new DLM with state vector $\theta_t' = \left(\theta_t^{(1)'}, \ldots, \theta_t^{(k)'}\right)$, as in the previous case,

and matrices

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

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

$$m_0' = \left(m_0^{(1)'}, \ldots, m_0^{(k)'}\right), \qquad C_0 = \begin{bmatrix} C_0^{(1)} & & \\ & \ddots & \\ & & C_0^{(k)} \end{bmatrix}.$$

For example, suppose you have two time series, the first following a stochastic linear trend and the second a random noise plus a quarterly seasonal component, both series being observed with noise. A joint DLM for the two series, assuming independence, can be set up in `R` as follow.

```
> dlmModPoly(dV = 0.2, dW = c(0, 0.5)) %+%
+     (dlmModSeas(4, dV = 0, dW = c(0, 0, 0.35)) +
+      dlmModPoly(1, dV = 0.1, dW = 0.03))
```

1.4 Time-varying models

In a time-varying DLM at least one of the entries of F_t, V_t, G_t, or W_t changes with time. We can think of any such entry as a time series or, more generally, a numeric vector. All together, the time-varying entries of the model matrices can be stored as a multivariate time series or a numeric matrix. This idea forms the basis for the internal representation of a time-varying DLM. A object `m` of class `dlm` may contain components named `JFF, JV, JGG, JW`, and `X`. The first four are matrices of integers of the same dimension as `FF, V, GG, W`, respectively, while `X` is an n by m matrix, where n is the number of observations in the data. Entry (i, j) of `JFF` is zero if the corresponding entry of `FF` is time invariant, or k if the vector of values of $F_t[i, j]$ at different times is stored in `X[,k]`. In this case the actual value of `FF[i,j]` in the model object is not used. Similarly for the remaining matrices of the DLM. For example, a dynamic linear regression can be modeled as

$$\begin{cases} y_t = \alpha_t + x_t\beta_t + v_t & v_t \sim \mathcal{N}(0, V_t) \\ \alpha_t = \alpha_{t-1} + w_{\alpha,t}, & w_{\alpha,t} \sim \mathcal{N}(0, W_{\alpha,t}) \\ \beta_t = \beta_{t-1} + w_{\beta,t}, & w_{\beta,t} \sim \mathcal{N}(0, W_{\beta,t}). \end{cases}$$

Here the state of the system is $\theta_t = (\alpha_t,\ \beta_t)'$, with β_t being the regression coefficient at time t and x_t being a covariate at time t, so that $F_t = [1,\ x_t]$. Such a DLM can be set up in `R` as follows,

```
> u <- rnorm(25)
> myMod <- dlmModReg(u, dV = 14.5)
> myMod$JFF
```

```
     [,1] [,2]
[1,]    0    1
```

```
> head(myMod$X)
```

```
       [,1]
[1,]  0.639
[2,]  0.947
[3,]  3.128
[4,]  2.733
[5,] -0.480
[6,]  0.156
```

Currently the outer sum of time-varying DLMs is not implemented.

2 Maximum likelihood estimation

It is often the case that one has unknown parameters in the matrices defining a DLM. While package `dlm` was primarily developed for Bayesian inference, it offers the possibility of estimating unknown parameters using maximum likelihood. The function `dlmMLE` is essentially a wrapper around a call to `optim`. In addition to the data and starting values for the optimization algorithm, the function requires a function argument that "builds" a DLM from any specific value of the unknown parameter vector. We illustrate the usage of `dlmMLE` with a couple of simple examples.

Consider the Nile river data set. A reasonable model can be a random walk plus noise, with unknown system and observation variances. Parametrizing variances on a log scale, to ensure positivity, the model can be build using the function defined below.

```
> buildFun <- function(x) {
+     dlmModPoly(1, dV = exp(x[1]), dW = exp(x[2]))
+ }
```

Starting the optimization from the arbitrary $(0, 0)$ point, the MLE of the parameter can be found as follows.

```
> fit <- dlmMLE(Nile, parm = c(0,0), build = buildFun)
> fit$conv

[1] 0

> dlmNile <- buildFun(fit$par)
> V(dlmNile)

      [,1]
[1,] 15100

> W(dlmNile)

     [,1]
[1,] 1468
```

For comparison, the estimated variances obtained using `StructTS` are

```
> StructTS(Nile, "level")

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

Variances:
  level  epsilon
   1469    15099
```

As a less trivial example, suppose one wants to take into account a jump in the flow of the river following the construction of Ashwan dam in 1898. This can be done by inflating the system variance in 1899 using a multiplier bigger than one.

```
> buildFun <- function(x) {
+     m <- dlmModPoly(1, dV = exp(x[1]))
+     m$JW <- matrix(1)
+     m$X <- matrix(exp(x[2]), nc = 1, nr = length(Nile))
+     j <- which(time(Nile) == 1899)
+     m$X[j,1] <- m$X[j,1] * (1 + exp(x[3]))
+     return(m)
+ }
> fit <- dlmMLE(Nile, parm = c(0,0,0), build = buildFun)
> fit$conv
```

```
[1] 0

> dlmNileJump <- buildFun(fit$par)
> V(dlmNileJump)

      [,1]
[1,] 16300

> dlmNileJump$X[c(1, which(time(Nile) == 1899)), 1]

[1] 2.79e-02 6.05e+04
```

The conclusion is that, after accounting for the 1899 jump, the level of the series is essentially constant in the periods before and after that year.

3 Filtering, smoothing and forecasting

Thanks to the fact that the joint distribution of states and observations is Gaussian, when all the parameters of a DLM are known it is fairly easy to derive conditional distributions of states or future observations conditional on the observed data. In what follows, for any pair of integers (i, j), with $i \leq j$, we will denote by $y_{i:j}$ the observations from the ith to the jth, inclusive, i.e., $y_i, \ldots, y_j$; a similar notation will be used for states. The *filtering* distribution at time t is the conditional distribution of θ_t given $y_{1:t}$. The *smoothing* distribution at time t is the conditional distribution of $\theta_{0:t}$ given $y_{1:t}$, or sometimes, with an innocuous abuse of language, any of its marginals, e.g., the conditional distribution of θ_s given $y_{1:t}$, with $s \leq t$. Clearly, for $s = t$ this marginal coincides with the filtering distribution. We speak of *forecast* distribution, or *predictive* distribution, to denote a conditional distribution of future states and/or observations, given the data up to time t. Recursive algorithms, based on the celebrated Kalman filter algorithm or extensions thereof, are available to compute filtering and smoothing distributions. Predictive distributions can be easily derived recursively from the model definition, using as "prior" mean and variance – m_0 and C_0 – the mean and variance of the filtering distribution. This is the usual Bayesian sequential updating, in which the posterior at time t takes the role of a prior distribution for what concerns the observations after time t. Note that any marginal or conditional distribution, in particular the filtering, smoothing and predictive distributions, is Gaussian and, as such, it is identified by its mean and variance.

3.1 Filtering

Consider again the last model fitted to the Nile river data on page 7. Taking the estimated parameters as known, we can compute the filtering distribution using `dlmFilter`. If n is the number of observations in the data set, `dlmFilter` returns the mean and variance of the $n+1$ filtering distributions that can be computed from the data, i.e., the distribution of θ_t given $y_{1:t}$ for $t = 0, 1, \ldots, n$ (for $t = 0$, this is by convention the prior distribution of θ_0).

```
> nileJumpFilt <- dlmFilter(Nile, dlmNileJump)
> plot(Nile, type = 'o', col = "seagreen")
> lines(dropFirst(nileJumpFilt$m), type = 'o',
+       pch = 20, col = "brown")
```

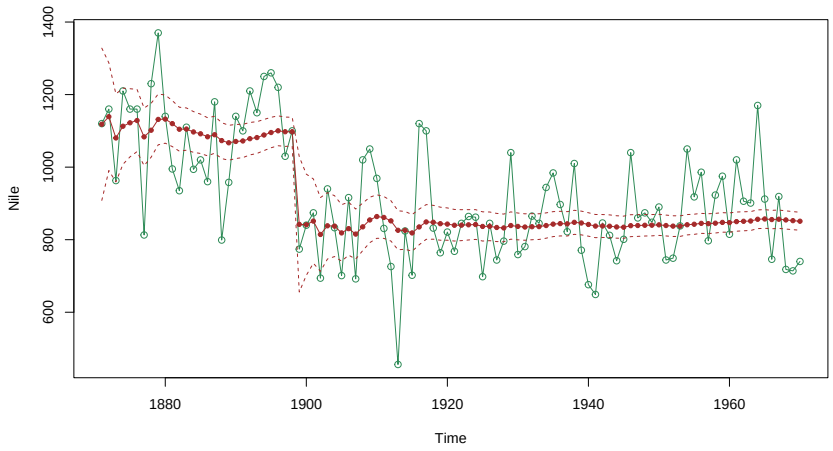

Figure 1: Nile data with filtered level

The variances $C_0, \ldots, C_n$ of the filtering distributions are returned in terms of their singular value decomposition (SVD). The SVD of a symmetric nonnegative definite matrix Σ is $\Sigma = UD^2U'$, where U is orthogonal and D is diagonal. In the list returned by `dlmFilter`, the U and D matrices corresponding to the SVD of $C_0, \ldots, C_n$ can be found as components `U.C` and `D.C`, respectively. While `U.C` is a list of matrices, since the D part in the SVD is diagonal, `D.C` consists in a matrix, storing in each row the diagonal entries of succesive D matrices. This decomposition is useful for further calculations one may be interested in, such as smoothing. However, if the

filtering variances are of interest per se, then `dlm` provides the utility function `dlmSvd2var`, which reconstructs the variances from their SVD. Variances can then be used for example to compute filtering probability intervals, as illustrated below.

```
> attach(nileJumpFilt)
> v <- unlist(dlmSvd2var(U.C, D.C))
> pl <- dropFirst(m) + qnorm(0.05, sd = sqrt(v[-1]))
> pu <- dropFirst(m) + qnorm(0.95, sd = sqrt(v[-1]))
> detach()
> lines(pl, lty = 2, col = "brown")
> lines(pu, lty = 2, col = "brown")
```

In addition to filtering means and variances, `dlmFilter` also returns means and variances of the distributions of θ_t given $y_{1:t-1}$, $t = 1, \ldots, n$ (one-step-ahead forecast distributions for the states) and means of the distributions of y_t given $y_{1:t-1}$, $t = 1, \ldots, n$ (one-step-ahead forecast distributions for the observations). The variances are returned also in this case in terms of their SVD.

3.2 Smoothing

The function `dlmSmooth` computes means and variances of the smoothing distributions. It can be given a data vector or matrix together with a specific object of class `dlm` or, alternatively, a "filtered DLM" produced by `dlmFilter`. The following `R` code shows how to use the function in the Nile river example.

```
> nileJumpSmooth <- dlmSmooth(nileJumpFilt)
> plot(Nile, type = 'o', col = "seagreen")
> attach(nileJumpSmooth)
> lines(dropFirst(s), type = 'o', pch = 20, col = "brown")
> v <- unlist(dlmSvd2var(U.S, D.S))
> pl <- dropFirst(s) + qnorm(0.05, sd = sqrt(v[-1]))
> pu <- dropFirst(s) + qnorm(0.95, sd = sqrt(v[-1]))
> detach()
> lines(pl, lty = 2, col = "brown")
> lines(pu, lty = 2, col = "brown")
```

As a second example, consider the UK gas consumption data set. On a logarithmic scale, this can be reasonably modeled by a DLM containing a quarterly seasonal component and a local linear trend, in the form of an integrated random walk. We first estimate the unknown variances by ML.

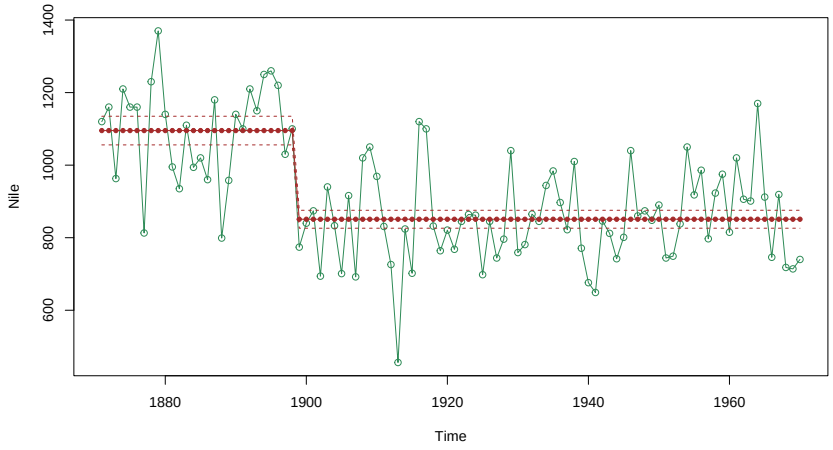

Figure 2: Nile river data with smoothed level

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

[1] 0

> dlmGas <- buildFun(fit$par)
> drop(V(dlmGas))

[1] 0.121

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

[1] 0.0626 0.0278
```

Based on the fitted model, we can compute the smoothing estimates of the states. This can be used to obtain a decomposition of the data into a smooth trend plus a stochastic seasonal component, subject to measurement error.

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

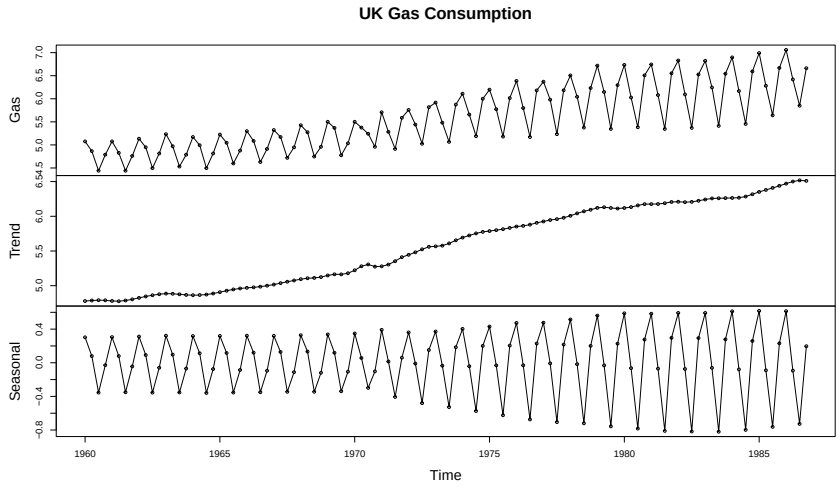

Figure 3: Gas consumption with "trend + seasonal" decomposition

3.3 Forecasting

Means and variances of the forecast distributions of states and observations can be obtained with the function `dlmForecast`, as illustrated in the code below. Means and variances of future states and observations are returned in a list as components `a`, `R`, `f`, and `Q`.

```
> gasFilt <- dlmFilter(lGas, mod = dlmGas)
> gasFore <- dlmForecast(gasFilt, nAhead = 20)
> sqrtR <- sapply(gasFore$R, function(x) sqrt(x[1,1]))
> pl <- gasFore$a[,1] + qnorm(0.05, sd = sqrtR)
> pu <- gasFore$a[,1] + qnorm(0.95, sd = sqrtR)
> x <- ts.union(window(lGas, start = c(1982, 1)),
+               window(gasSmooth$s[,1], start = c(1982, 1)),
+               gasFore$a[,1], pl, pu)
> plot(x, plot.type = "single", type = 'o', pch = c(1, 0, 20, 3, 3),
+      col = c("darkgrey", "darkgrey", "brown", "yellow", "yellow"),
```

```
+      ylab = "Log gas consumption")
> legend("bottomright", legend = c("Observed",
+                         "Smoothed (deseasonalized)",
+                         "Forecasted level", "90% probability limit"),
+        bty = 'n', pch = c(1, 0, 20, 3, 3), lty = 1,
+        col = c("darkgrey", "darkgrey", "brown", "yellow", "yellow"))
```

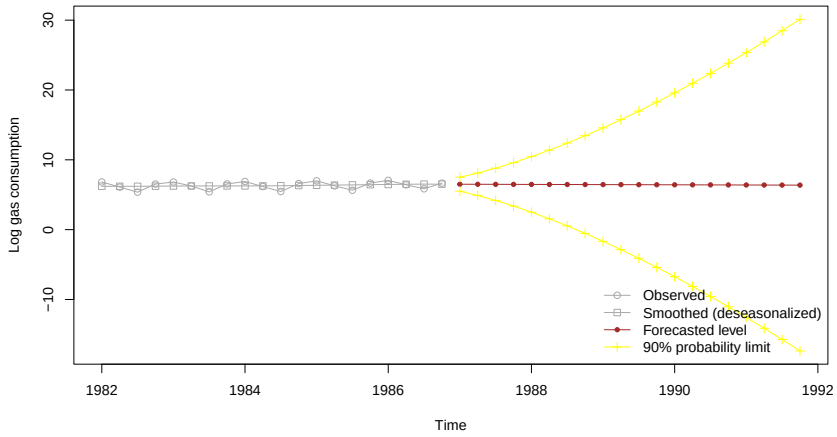

Figure 4: Gas consumption forecast

4 Bayesian analysis of Dynamic Linear Models

If all the parameters defining a DLM are known, then the functions for smoothing and forecasting illustrated in the previous section are all is needed to perform a Bayesian analysis. At least, this is true if one is interested in posterior estimates of unobservable states and future observations or states. In almost every real world application a DLM contains in its specification one or more unknown parameters that need to be estimated. Except for a few very simple models and special priors, the posterior distribution of the unknown parameters – or the joint posterior of parameters and states – does not have a simple form, so the common approach is to use Markov chain Monte Carlo (MCMC) methods to generate a sample from the posterior[1].

[1] An alternative to MCMC is provided by Sequential Monte Carlo methods, which will not be discussed here.

MCMC is highly model- and prior-dependent, even within the class of DLMs, and therefore we cannot give a general algorithm or canned function that works in all cases. However, package `dlm` provides a few functions to facilitate posterior simulation via MCMC. In addition, the package provides a minimal set of tools for analyzing the output of a sampler.

4.1 Forward filtering backward sampling

One way of obtaining a sample from the joint posterior of parameters and unobservable states is to run a Gibbs sampler, alternating draws from the full conditional distribution of the states and from the full conditionals of the parameters. While generating the parameters is model dependent, a draw from the full conditional distribution of the states can be obtained easily in a general way using the so-called Forward Filtering Backward Sampling (FFBS) algorithm, developed independently by [CK, FS, S]. The algorithm consists essentially in a simulation version of the Kalman smoother. An alternative algorithm can be found in [DK]. Note that, within a Gibbs sampler, when generating the states, the model parameters are fixed at their most recently generated value. The problem therefore reduces to that of drawing from the conditional distribution of the states given the observations for a completely specified DLM, which is efficiently done based on the general structure of a DLM.

In many cases, even if one is not interested in the states but only in the unknown parameters, keeping also the states in the posterior distribution simplifies the Gibbs sampler. This typically happens when there are unknown parameters in the system equation and system variances, since usually conditioning on the states makes those parameters independent of the data and results in simpler full conditional distributions. In this framework, including the states in the sampler can be seen as a data augmentation technique.

In package `dlm`, FFBS is implemented in the function `dlmBSample`. To be more precise, this function only performs the backward sampling part of the algorithm, starting from a filtered model. The only argument of `dlmBSample` is in fact a `dlmFiltered` object, or a list that can be interpreted as such.

In the code below we generate and plot (Figure 5) ten simulated realizations from the posterior distribution of the unobservable "true level" of the Nile river, using the random walk plus noise model (without level jump, see Section 2). Model parameters are fixed for this example to their MLE.

```
> plot(Nile, type = 'o', col = "seagreen")
```

```
> nileFilt <- dlmFilter(Nile, dlmNile)
> for (i in 1:10) # 10 simulated "true" levels
+     lines(dropFirst(dlmBSample(nileFilt)), col = "brown")
```

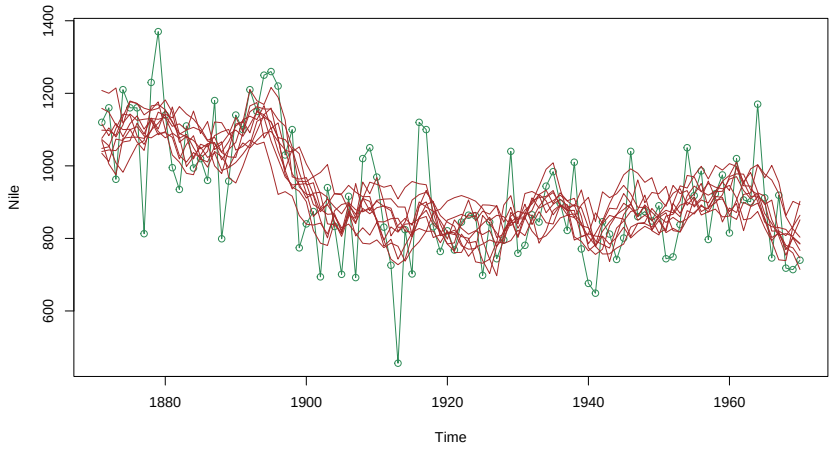

Figure 5: Nile river with simulated *true level*s

The figure would look much different had we used the model with a jump!

4.2 Adaptive rejection Metropolis sampling

Gilks and coauthors developed in [GBT] a clever adaptive method, based on rejection sampling, to generate a random variable from any specified continuous distribution. Although the algorithm requires the support of the target distribution to be bounded, if this is not the case one can, for all practical purposes, restrict the distribution to a very large but bounded interval. Package `dlm` provides a port to the original C code written by Gilks with the function `arms`. The arguments of `arms` are a starting point, two functions, one returning the log of the target density and the other being the indicator of its support, and the size of the requested sample. Additional arguments for the log density and support indicator can be passed via the `...` argument. The help page contain several examples, most of them unrelated to DLMs. A nontrivial one is the following, dealing with a mixture of normals

target. Suppose the target is

$$f(x) = \sum_{i=1}^{k} p_i \phi(x; \mu_i, \sigma_i),$$

where $\phi(\cdot; \mu, \sigma)$ is the density of a normal random variable with mean μ and variance σ^2. The following is an `R` function that returns the log density at the point `x`:

```
> lmixnorm <- function(x, weights, means, sds) {
+     log(crossprod(weights, exp(-0.5 * ((x - means) / sds)^2
+                                - log(sds))))
+ }
```

Note that the weights p_i's, as well as means and standard deviations, are additional arguments of the function. Since the support of the density is the entire real line, we use a reasonably large interval as "practical support".

```
> y <- arms(0, myldens = lmixnorm,
+           indFunc = function(x,...) (x > (-100)) * (x < 100),
+           n = 5000, weights = c(1, 3, 2),
+           means = c(-10, 0, 10), sds = c(7, 5, 2))
> summary(y)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -34.4    -3.9     2.3     1.6     9.0    18.2

> library(MASS)
> truehist(y, prob = TRUE, ylim = c(0, 0.08), bty = 'o')
> curve(colSums(c(1, 3, 2) / 6 *
+               dnorm(matrix(x, 3, length(x), TRUE),
+                     mean = c(-10, 0, 10), sd = c(7, 5, 2))),
+       add = TRUE)
> legend(-25, 0.07, "True density", lty = 1, bty = 'n')
```

A useful extension in the function `arms` is the possibility of generating samples from multivariate target densities. This is obtained by generating a random line through the starting, or current, point and applying the univariate ARMS algorithm along the selected line. Several examples of this type of application are contained in the help page.

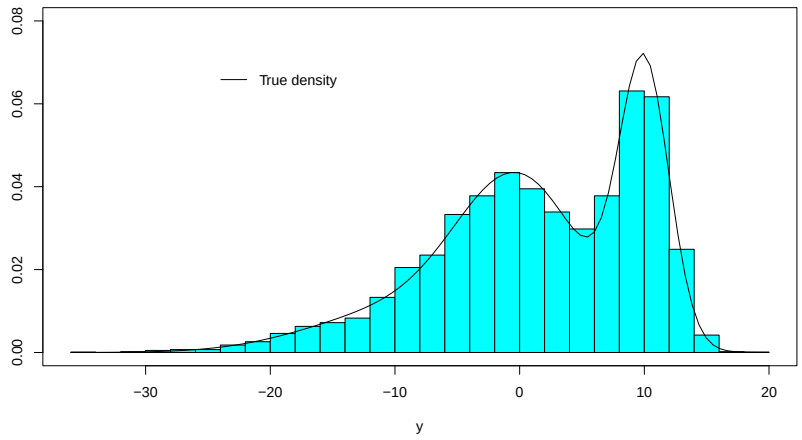

Figure 6: Mixture of 3 Normals

4.3 Gibbs sampling: an example

We provide in this section a simple example of a Gibbs sampler for a DLM with unknown variances. This type of model is rather common in applications and is sometimes referred to as *d-inverse-gamma* model.

Consider again the UK gas consumption data modeled as a local linear trend plus a seasonal component, observed with noise. The model is based on the unobserved state

$$\theta_t = \left(\mu_t \ \beta_t \ s_t^{(1)} \ s_t^{(2)} \ s_t^{(3)}\right)',$$

where μ_t is the current level, β_t is the slope of the trend, $s_t^{(1)}, s_t^{(2)}$ and $s_t^{(3)}$ are the seasonal components during the current quarter, previous quarter, and two quarters back. The observation at time t is given by

$$y_t = \mu_t + s_t^{(1)} + v_t, \qquad v_t \sim \mathcal{N}(0, \sigma^2).$$

We assume the following dynamics for the unobservable states:

$$\begin{aligned}
\mu_t &= \mu_{t-} + \beta_{t-1}, \\
\beta_t &= \beta_{t-1} + w_t^\beta, \qquad w_t^\beta \sim \mathcal{N}(0, \sigma_\beta^2) \\
s_t^{(1)} &= -s_{t-1}^{(1)} - s_{t-1}^{(2)} - s_{t-1}^{(3)} + w_t^s, \qquad w_t^\beta \sim \mathcal{N}(0, \sigma_s^2) \\
s_t^{(2)} &= s_{t-1}^{(1)}, \\
s_t^{(3)} &= s_{t-1}^{(2)}.
\end{aligned}$$

In terms of the DLM representing the model, the above implies

$$\begin{aligned}
V &= [\sigma^2], \\
W &= \mathrm{diag}(0, \sigma_\beta^2, \sigma_s^2, 0, 0).
\end{aligned}$$

For details on the model, see [WH]. The unknown parameters are therefore the three variances σ^2, σ_β^2, and σ_s^2. We assume for their inverse, i.e., for the three precisions, independent gamma priors with mean $a, a_{\theta,2}, a_{\theta,3}$ and variance $b, b_{\theta,2}, b_{\theta,3}$, respectively.

Straightforward calculations show that, adding the unobservable states as latent variables, a Gibbs sampler can be run based on the following full conditional distributions:

$$\begin{aligned}
\theta_{0:n} &\sim \mathcal{N}(), \\
\sigma^2 &\sim \mathcal{IG}\left(\frac{a^2}{b} + \frac{n}{2}, \frac{a}{b} + \frac{1}{2} SS_y\right), \\
\sigma_\beta^2 &\sim \mathcal{IG}\left(\frac{a_{\theta,2}^2}{b_{\theta,2}} + \frac{n}{2}, \frac{a_{\theta,2}}{b_{\theta,2}} + \frac{1}{2} SS_{\theta,2}\right), \\
\sigma_s^2 &\sim \mathcal{IG}\left(\frac{a_{\theta,3}^2}{b_{\theta,3}} + \frac{n}{2}, \frac{a_{\theta,3}}{b_{\theta,3}} + \frac{1}{2} SS_{\theta,3}\right),
\end{aligned}$$

with

$$\begin{aligned}
SS_y &= \sum_{t=1}^{n} (y_t - F_t \theta_t)^2, \\
SS_{\theta,i} &= \sum_{t=1}^{T} (\theta_{t,i} - (G_t \theta_{t-1})_i)^2, \qquad i = 2, 3.
\end{aligned}$$

The full conditional of the states is normal with some mean and variance that we don't need to derive explicitely, since `dlmBSample` will take care of generating $\theta_{0:n}$ from the appropriate distribution.

The function `dlmGibbsDIG`, included in the package more for didactical reasons than anything else, implements a Gibbs sampler based on the full conditionals described above. A piece of `R` code that runs the sampler will look like the following.

```
> outGibbs <- dlmGibbsDIG(lGas, dlmModPoly(2) + dlmModSeas(4),
+                         a.y = 1, b.y = 1000, a.theta = 1,
+                         b.theta = 1000,
+                         n.sample = 1100, ind = c(2, 3),
+                         save.states = FALSE)
```

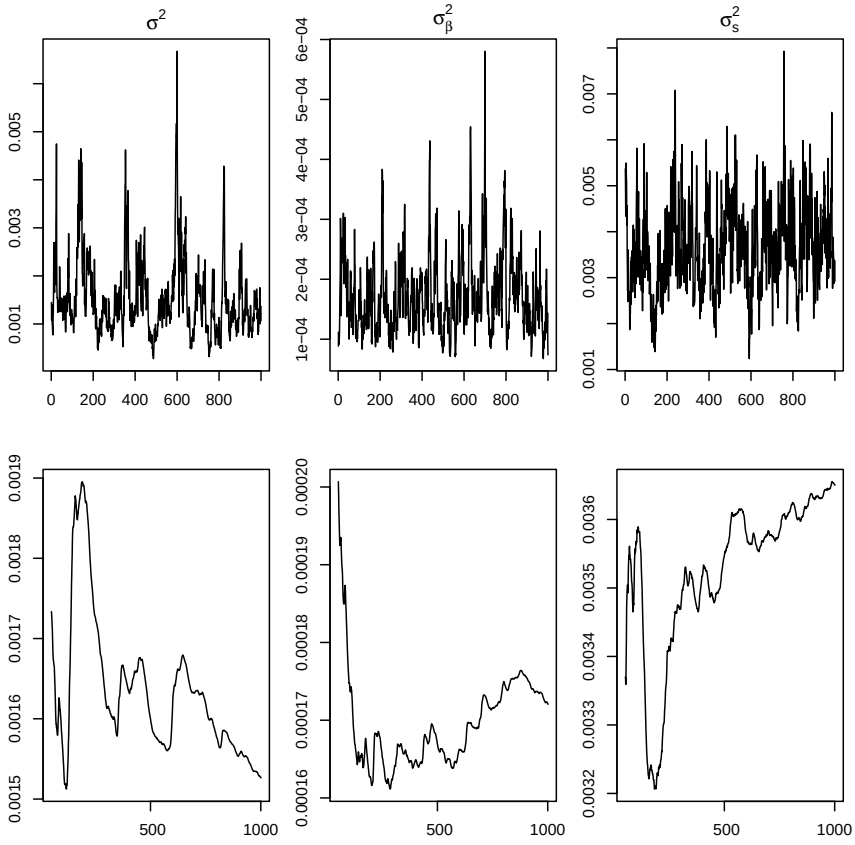

Figure 7: Trace plots (top) and running ergodic means (bottom)

After discarding the first 100 values as burn in, plots of simulated values and running ergodic means can be obtained as follows, see Figure 7.

```
> burn <- 100
> attach(outGibbs)
> dV <- dV[-(1:burn)]
> dW <- dW[-(1:burn), ]
> detach()
> par(mfrow=c(2,3), mar=c(3.1,2.1,2.1,1.1))
> plot(dV, type = 'l', xlab = "", ylab = "",
+      main = expression(sigma^2))
> plot(dW[ , 1], type = 'l', xlab = "", ylab = "",
+      main = expression(sigma[beta]^2))
> plot(dW[ , 2], type = 'l', xlab = "", ylab = "",
+      main = expression(sigma[s]^2))
> use <- length(dV) - burn
> from <- 0.05 * use
> at <- pretty(c(0, use), n = 3); at <- at[at >= from]
> plot(ergMean(dV, from), type = 'l', xaxt = 'n',
+      xlab = "", ylab = "")
> axis(1, at = at - from, labels = format(at))
> plot(ergMean(dW[ , 1], from), type = 'l', xaxt = 'n',
+      xlab = "", ylab = "")
> axis(1, at = at - from, labels = format(at))
> plot(ergMean(dW[ , 2], from), type = 'l', xaxt = 'n',
+      xlab = "", ylab = "")
> axis(1, at =  at - from, labels = format(at))
```

Posterior estimates of the three unknown variances, from the Gibbs sampler output, together with their Monte Carlo standard error, can be obtained using the function `mcmcMean`.

```
> mcmcMean(cbind(dV[-(1:burn)], dW[-(1:burn), ]))
```

```
                W.2         W.3
  1.52e-03   1.72e-04   3.66e-03
 (1.29e-04) (6.40e-06) (1.11e-04)
```

5 Concluding remarks

We have described and illustrated the main features of package `dlm`. Although the package has been developed with Bayesian MCMC-based applications in mind, it can also be used for maximum likelihood estimation and

Kalman filtering/smoothing. The main design objectives had been 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. The algorithms used in the package are based on the sequential evaluation of variance matrices in terms of their SVD. While this can be seen as a form of square root filter (smoother), it is much more robust than the standard square root filter based on the propagation of Cholesky decomposition. In fact, the SVD-based algorithm does not even require the matrix W to be invertible. (It does require V to be nonsingular, however).

As far as Bayesian inference is concerned, the package provides the tools to easily implement a Gibbs sampler for any univariate or multivariate DLM. The functions `dlmBSample` to generate the states and `arms`, the multivariate estension of ARMS, can be used alone or in combination in a Gibbs sampler, allowing the user to carry out Bayesian posterior inference for a wide class of models and priors.

References

[CK] Carter, C.K. and Kohn, R. (1994). On Gibbs sampling for state space models. *Biometrika*, 81.

[DK] Durbin, J. and Koopman, S.J. (2001). *Time Series analysis by state space methods*. Oxford University Press.

[FS] Früwirth-Schnatter, S. (1994). Data augmentation and dynamic linear models. *journal of Time Series Analysis*, 15.

[GBT] Gilks, W.R., Best, N.G. and Tan, K.K.C. (1995). Adaptive rejection Metropolis sampling within Gibbs sampling (Corr: 97V46 p541-542 with Neal, R.M.), *Applied Statistics*, 44.

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

[S] Shephard, N. (1994). Partial non-Gaussian state space models. *Biometrika*, 81.

[WH] West, M. and Harrison, J. (1997). *Bayesian forecasting and dynamic models*. (Second edition. First edition: 1989), Springer, N.Y.

[ZL] Zhang, Y. and Li, R. (1996). Fixed-interval smoothing algorithm based on singular value decomposition. *Proceedings of the 1996 IEEE international conference on control applications*.