GKF Package Example (Version 1.7.0)

Tarak Kharrat and Georgi N. Boshnakov

May 19, 2018

Introduction

GKF is an R package for computations with state space models. It deals with the linear Gaussian as well as the nonlinear non-Gaussian case. It is mostly written in C++ and relies fully on linear algebra subroutines contained in the Armadillo library (Sanderson et al., 2010).

This paper shows how GKF can be used to fit general state space models and is by no means a tutorial on state space models. We will explain how filtering and smoothing can be done, how to estimate the parameters by maximum likelihood and how to sample (smoother) from the (conditional) posterior distribution of the signal through univariate and multivariate examples. We refer readers interested in a more theoretical treatement to Durbin and Koopman (2012). We do not restrict our treatment to the GKF package but alternatives using other R packages (when it is possible) are presented. This approach has two advantages: validate the computation done by GKF and allow users to choose between the different tools.

As of May 2018, the most complete implementation seems to be provided by package KFAS, which has excellent documentation, see Helske (2017). Our package GKF may be attractive if direct support of intercept terms in the observation and/or the state equations is desired (with KFAS this requires a little bit of effort). Package FKF also supports intercept terms in the state and observation equations but has been archived on CRAN for some time (GKF can be used as an almost drop in replacement see the examples in the following sections).

The remainder of this paper is structured as follows. In Section 1 we recall the state space model, set the notation framework and show how to define such models with GKF using the case of a univariate autoregressive moving average (ARMA) process. An example of maximum likelihood estimation is presented in Section 3. Filtering and smoothing are treated in Section 4 in the univariate and multivariate case. Section 5 describes sampling (smoother) from the signal posterior distribution which can be useful for estimation based on simulation techniques such as importance sampling. An additional (multivariate) illustration is fully studied in Section 6. A nonlinear non-Gaussian case is discussed in Section 7. We conclude in Section 8.

1 The Model

Let $\{y_t\}$, $t = 1, \dots, n$, be the $(d \times 1)$ observed process. For the observed process, $\{y_t\}$, we consider a general nonlinear non-Gausian model, driven by a linear Markov state process. In the case of the nonlinear non-Gaussian observation model we assume that the conditional density of y_t, given the information up to time t, is a function of a $d \times 1$ signal vector θ_t and a vector of parameters ψ. We denote this density by $p(y_t|\theta_t;\psi)$ and write symbolically

$$y_t \sim p(y_t|\theta_t;\psi). \tag{1}$$

The linear Gaussian observation model is usually specified by an explicit equation for y_t:

$$y_t = \theta_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, H_t), \quad t = 1, \dots, n, \tag{2}$$

where the ϵ_t's are independent and the variance matrices H_t are fixed and known or can partly depend on the vector parameter ψ. The linear Gaussian observation model can be viewed as a particular case of the general model, in which $p(y_t|\theta_t;\psi)$ is the Gaussian density $\mathcal{N}(\theta_t, H_t)$ and the variance matrices do not depend on the past information.

In both cases, the signal is modelled as a linear transformation of a state vector α_t, i.e.

$$\theta_t = c_t + Z_t\alpha_t,$$

where the intercept vector c_t $(d \times 1)$ and the factor matrix Z_t $(d \times m)$ can also partialy depend on ψ.

For the state α_t we consider the linear Markov model with Gaussian innovations defined by the following (transition) equation:

$$\alpha_{t+1} = d_t + T_t\alpha_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, Q_t), \quad t = 1, \dots, n, \tag{3}$$

where α_t $(m \times 1)$ is the state vector, d_t $(m \times 1)$ is the intercept, T_t $(m \times m)$ is the transition (state) matrix and η_t $(m \times 1)$ is the innovation (with $(m \times m)$ variance matrix Q_t) of the transition equation. The vector d_t and matrices T_t and Q_t can partly depend on the parameter vector ψ and be partly fixed and known for $t = 1, \dots, n$. The matrix Q_t is only required to be **positive semi-definite**. The initial state vector α_0 is normally distributed with mean a and variance P_0. The innovations η_t are serially independent and are independent of the initial state vector for $t = 1, \dots, n$.

The model specified by equations (2)–(3) is known also as dynamic linear model (dlm).

The power of state space models comes from efficient recursive computations based on the above equations. Nevertheless, in development and presentation of the methodology it is often easier and more clear to work with the big matrices and vectors obtained by pooling together the corresponding quantities for $t = 1, \dots, n$. To this end, define the all times observation vector $(dn \times 1)$ $y = (y_1', \dots, y_n')'$, state vector $(nm \times 1)$ $\alpha = (\alpha_1', \dots, \alpha_n')'$, signal vector $(dn \times 1)$ $\theta = (\theta_1', \dots, \theta_n')'$, intercept vector $(dn \times 1)$ $c = (c_1', \dots, c_n')'$, linear innovations vector $(dn \times 1)$ $\epsilon = (\epsilon_1', \dots, \epsilon_n')$ and matrices $(dn \times mn)$ $\mathcal{Z} = \operatorname{diag}(Z_1, \dots, Z_n)$ and $(dn \times dn)$ $H = \operatorname{diag}(H_1, \dots, H_n)$. For the all time signal we now have

$$\theta = c + \mathcal{Z}\alpha.$$

The linear observation equation can be written in matrix form as

$$y = \theta + \epsilon = c + \mathcal{Z}\alpha + \epsilon, \quad \epsilon \sim \mathcal{N}(0, H). \tag{4}$$

The state vector, α, has a multivariate Gaussian distribution. More specifically, $\alpha \sim \mathcal{N}(d, \Omega)$, where d $(nm \times 1)$ is given by $d = T(a', d_1', \dots, d_{n-1}')'$, Ω $(nm \times nm)$ by $\Omega = T\operatorname{diag}(P_1, Q_1, \dots, Q_{n-1})T'$ and the (i, j)th block, T_{ij}, of the lower block-triangular matrix T (for $i, j = 1, \dots, n$) is given by

$$T_{ij} = \begin{cases} 0, & \text{if } i < j \\ I, & \text{if } i = j \\ T_{i-1} \times \cdots \times Tj, & \text{if } i > j. \end{cases} \tag{5}$$

The pattern of T is important for computations, so we give it explicitly:

$$T = \begin{bmatrix} I & 0 & 0 & 0 & & 0 & 0 \\ T_1 & I & 0 & 0 & & 0 & 0 \\ T_2T_1 & T_2 & I & 0 & & 0 & 0 \\ T_3T_2T_1 & T_3T_2 & T_3 & I & & 0 & 0 \\ & & & & \ddots & & \vdots \\ T_{n-2}\cdots T_1 & T_{n-2}\cdots T_2 & T_{n-2}\cdots T_3 & T_{n-2}\cdots T_4 & & I & 0 \\ T_{n-1}\cdots T_1 & T_{n-1}\cdots T_2 & T_{n-1}\cdots T_3 & T_{n-1}\cdots T_4 & & T_{n-1} & I \end{bmatrix} \tag{6}$$

From the above it follows that the all times signal vector is multivariate normal, as well, and

$$\theta \sim \mathcal{N}(\mu, \Psi), \quad \mu = c + \mathcal{Z}d, \quad \Psi = \mathcal{Z}\Omega\mathcal{Z}'. \tag{7}$$

(fixme: give citations in this section)

2 Defining and manipulating state space models with GKF

2.1 Defining state space models with GKF

GKF accepts system variables prepared as matrices or arrays. Time-varying parameters are given as arrays. Time-invariant parameters can also be specified as matrices which are replicated as needed. The correspondence between the variables in the equations in Section 1 and the names in GKF are given in the following table. For convenience we also give the parameter names in package fkf.

parameter	a_0	P_0	c_t	d_t	Z_t	T_t	??	Q_t
GKF name	a0	P0	ct	dt	Zt	Tt	Ht	Qt
fkf name	a0	P0	ct	dt	Zt	Tt	GGt	HHt

The time series is a $(d \times n)$ matrix, where d is the dimension of the time series.

A consistency check is done in each function (before computation starts) which can be switched off by setting `checkInputs` to `FALSE`. It is prudent to keep `checkInputs = TRUE` unless the performance becomes crucial and correctness of the arguments is ensured.

2.1.1 Autoregressive moving average model

An autoregressive moving average ARMA(p, q) model is defined as

$$y_t \quad = \quad \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \zeta_t + \theta_1 \zeta_{t-1} + \theta_q \zeta_{t-q}, \quad \zeta_t \sim \mathcal{N}(0, \sigma^2) \tag{8}$$

with non-negative integers p and q and where ζ_t is a serially independent series of $\mathcal{N}(0, \sigma^2)$ disturbances. This can be written in the form

$$y_t \quad = \quad \sum_{j=1}^{r} \phi_j y_{t-j} + \zeta_t + \sum_{j=1}^{r} \theta_j \zeta_{t-j}, \quad t = 1, \ldots, n \tag{9}$$

where $r = \max(p, q+1)$ and for which some coefficients are zero. These models can be represented in state space form (see Durbin and Koopman, 2012, chap. 3.4).

For example, consider the ARMA$(2, 1)$. In this case $r = 2$. There are various ways to put the model in state space form. We set the state vector to $\alpha_t = (y_t, \phi_2 y_{t-1} + \theta_1 \zeta_t)'$. The associated transition equation is

$$\begin{pmatrix} y_{t+1} \\ \phi_2 y_t + \theta_1 \zeta_{t+1} \end{pmatrix} = \begin{bmatrix} \phi_1 & 1 \\ \phi_2 & 0 \end{bmatrix} \begin{pmatrix} y_t \\ \phi_2 y_{t-1} + \theta_1 \zeta_t \end{pmatrix} + \begin{pmatrix} 1 \\ \theta_1 \end{pmatrix} \zeta_{t+1}$$

The measurement equation is given by $y_t = Z_t \alpha_t$, which implies that $\epsilon_t = 0$ (and hence $H_t = 0$), $Z_t = [1\ 0]$. Note that $\eta_t = \binom{1}{\theta_1} \zeta_{t+1}$ and hence $Q_t = \sigma^2 \binom{1}{\theta_1} \binom{1}{\theta_1}'$.

Here we use Example 1 from the demos in package FKF (Luethi et al., 2014), where the following function to construct a state space representation from the four parameters of an ARMA$(2, 1)$ model can be found:

```
> arma21ss <- function(ar1, ar2, ma1, sigma) {
+     Tt <- matrix(c(ar1, ar2, 1, 0), ncol = 2)
+     Zt <- matrix(c(1, 0), ncol = 2)
+     ct <- matrix(0)
+     dt <- matrix(0, nrow = 2)
+     GGt <- matrix(0)
+     H <- matrix(c(1, ma1), nrow = 2) * sigma
+     HHt <- H %*% t(H)
+     a0 <- c(0, 0)
+     P0 <- matrix(1e6, nrow = 2, ncol = 2)
+     list(a0 = a0, P0 = P0, ct = ct, dt = dt,
+          Zt = Zt, Tt = Tt, GGt = GGt, HHt = HHt)
+ }
```

For simplicity, the initial covariance matrix is set to a diffuse value. Note that the usual convention is to set the diffuse initial matrix to a diagonal matrix, e.g.

```
> P0 <- diag(1e6, nrow = 2, ncol = 2)
```

but we keep the original function from FKF for comparative purposes.

Here we simulate from an ARMA$(2, 1)$ process.

```
> ar1 <- 0.6
> ar2 <- 0.2
> ma1 <- -0.2
> sigma <- sqrt(0.2)
> n <- 1000
> a <- arima.sim(model = list(ar = c(ar1, ar2), ma = ma1), n = n,
+                innov = rnorm(n) * sigma)
```

This prepares the model and the time series for `GKF`:

```
> sp <- arma21ss(ar1, ar2, ma1, sigma)
> yt <- rbind(a)
```

The time series is transformed to a matrix with one row, since this is a univariate time series.

We can compute, for example, the unconditional log-likelihood of the process and check the result against the computation of the `FKF` package

```
> logLikikGKF <- FKF(a0 = sp$a0, P0 = sp$P0, dt = sp$dt, ct = sp$ct, Tt = sp$Tt,
+                    Zt = sp$Zt, Qt = sp$HHt, Ht = sp$GGt, yt = yt)$logLik
```

Note on 2018-03-29: We load an object created by "fkf" since the package FKF has been archived recently on CRAN, so may not be available. If you have it installed, run the commented out command below to create it:

```
> ## logLikfkf <- FKF::fkf(a0 = sp$a0, P0 = sp$P0, dt = sp$dt, ct = sp$ct, Tt = sp$Tt,
> ##                       Zt = sp$Zt, HHt = sp$HHt, GGt = sp$GGt, yt = yt)$logLik
> logLikfkf <- readRDS("logLikfkf.rds")
```

Note the notation differences for the innovation variance and perturbation variance between `FKF` and `GKF`.

The log-likelihoods computed by GKF and FKF are the same:

```
> all.equal(logLikikGKF, logLikfkf)

[1] TRUE
```

3 Maximum Likelihood Estimation

As discussed in Section 1, the system variables can partly depend on some unknown vector ψ that needs to be estimated. One way to do that is by maximum likelihood. We illustrated in the previous section that `GKF` returns the log-likelihood $p(y|\psi)$ when the function `FKF` is called. We can use it as the objective function of a desired optimization routine after providing some initial guess to obtain estimates of ψ. We illustrate this procedure for the local level model and the Nile river dataset.

3.1 The local level model

The local level model is another example of univariate dynamic linear model. It is described by the equations

$$\begin{aligned} y_t &= \alpha_t + \epsilon_t \quad \epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2) \\ \alpha_{t+1} &= \alpha_t + \eta_t \quad \eta_t \sim \mathcal{N}(0, \sigma_\eta^2) \end{aligned}$$

for $t = 1, \ldots, n$, where the ϵ_t's and ζ_t's are all mutually independent and are independent of α_1. Although it has a simple form, this model is not an artificial special case and indeed it provides the basis for the analysis of important real problems in practical time series analysis.

3.2 The Nile River dataset

The Nile River dataset is shipped with `R` and represents the measurements of the annual flow of the river at Ashwan betwenn 1871 and 1970 (100 observation). The local level model with unknown measurement and transition variances is knowm to describe the data reasonably well. The following peace of code show how to fit this model to the data using `optim` to minimize the inverse of the log-likelihood. We also assume that some observations are missing to illustrate how `GKF` treats `NA` (missing values are respresented by `NA`)

```
> y <- c(Nile)
> y[c(3, 10)] <- NA # NA values can be handled
> dt <- ct <- matrix(0)
> Zt <- Tt <- matrix(1)
> a0 <- y[1] # Estimation of the first year flow
> P0 <- matrix(100) # Variance of a0
> yt <- rbind(y)
> fit.GKF <- optim(c(Qt = var(y, na.rm = TRUE) * .5,
+                     Ht = var(y, na.rm = TRUE) * .5 ),
+                   fn = function(par, ...)
+                       -FKF(Qt = matrix(par[1]),
+                            Ht = matrix(par[2]), ...)$logLik,
+                   yt = yt, a0 = a0, P0 = P0, dt = dt, ct = ct,
+                   Zt = Zt, Tt = Tt, checkInputs = TRUE)
> Qt <- matrix(fit.GKF$par[1])
> Ht <- matrix(fit.GKF$par[2])
> GKF.obj <- FKF(a0 = a0, P0 = P0, dt = dt, ct = ct, Tt = Tt, Zt = Zt,
+                 Qt = Qt, Ht = Ht, yt =  yt)
> fit.stats <- StructTS(y, type = "level")
```

The objects created above are used in the examples in the following sections.

4 Filtering and Smoothing with `GKF`

In the linear Gaussian case, when all the system variables are known, the joint distribution of states and observations is Gaussian. Hence, it is fairly easy to derive conditional distributions of states, signals, disturbances, innovations or future observations conditional on the observed data. Such operations are called filtering, smoothing and forecasting (**TODO:** not implemented by `GKF` yet), respectively and are discussed in the following sections for the signal θ_t or state $\alpha_t, t = 1, \ldots, n$ vectors depending on the context. Note that given the linear transformation defining the signal from the state, when the Gaussian assumption applies, it is straightforward to obtain the conditional (on the data) mean and variance of the signal knowing the mean and variance of the state.

4.1 Filtering

The filtering distribution of the state at time t is the conditional distribution of α_t given the observed data up to time t, i.e, $Y_t = (y_1', \ldots, y_t')'$. Given that this distribution is Gaussian, filtering reduces to the computation of $a_{t|t} = \mathsf{E}[\alpha_t|Y_t]$ (the mode of $p(\alpha_t|y_1, \ldots, y_t)$) and $P_{t|t} = \text{var}[\alpha_t|Y_t]$, which is achieved by a (forward) recursive procedure, based on the celebrated Kalman filter algorithm or extensions thereof. The main ingredient of the Kalman filter derivation is recalled in Appendix A and the resulting equations are collected in Equation (15) although in a different context. Readers interested in the details of the derivation are referred to Durbin and Koopman (2012, chap. 4).

We illustrate the filtering for the local level model described earlier and for a multivariate example inspired from Petris et al. (2009, chap. 3.3.2).

4.1.1 Filtering for the Local Level Model

Consider again the model fitted to the Nile river data. Taking the estimated parameters as known, filtering was computed (by a call to FKF) and results stored in GKF.obj (following standard notations, filtered mean is named att and the associated variance Ptt). Figure 1 (page 6) shows flow data together with fitted local levels

```
> plot(y, main = "Nile flow", col = "brown")
> lines(GKF.obj$att[1, ], type = "o", pch = 20, col = "blue")
> legend("top", c("Nile flow data", "Local level (GKF)"),
+        col = c("brown", "blue"), lty = 1)
```

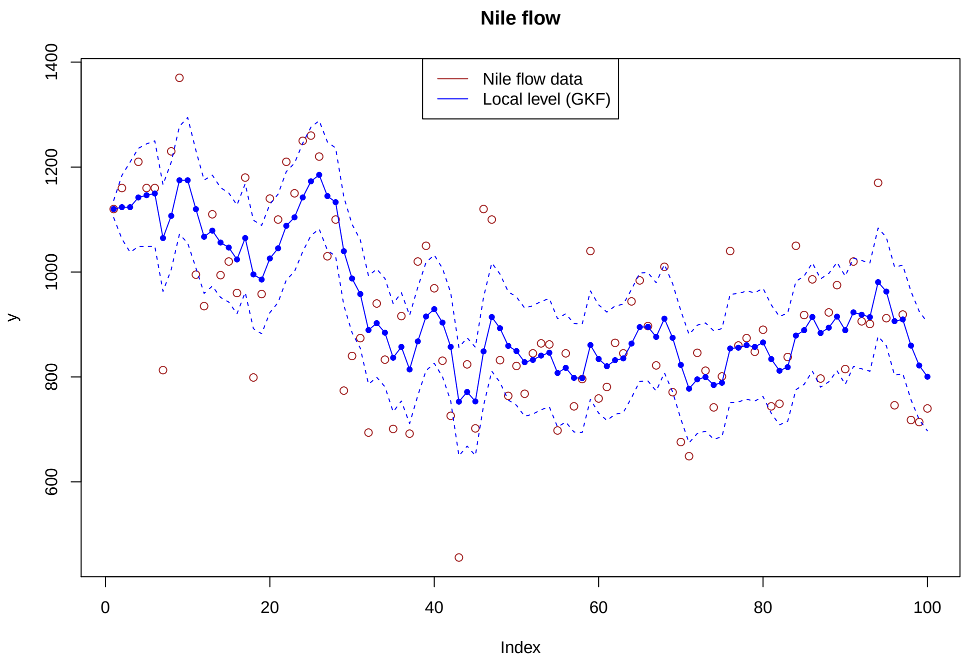

Figure 1: Nile data with filtered level

We can also compute 95% confidence intervals for the filtering estimate as follows:

```
> v <- as.numeric(GKF.obj$Ptt)
> pl <- GKF.obj$att[1, ] + qnorm(0.05, sd = sqrt(v))
> pu <- GKF.obj$att[1, ] + qnorm(0.95, sd = sqrt(v))
> lines(pl, lty = 2, col = "blue")
> lines(pu, lty = 2, col = "blue")
```

In addition to filtering means and variances (of the state), and the log-likelihood of the conditional data, FKF returns some other quantities computed by the Kalman filter that will be used for the smoothing and sampling. See GKF documentation for more details.

4.1.2 Multivariate Linear Growth Model

The multivariate model considered here is obtained by assuming that the vector of observation $y_t = (y_{1,t}, \ldots, y_{d,t})$ can by seen as d independent series and study them (independently) by specifying a univariate model for each of them. For example, each series will have a state vector with a level and a slope component and a diagonal variance matrix. This means that the evolution of the level and slope is governed by independent random inputs and suggests describing the joint evolution of the state vectors by grouping together all the levels and then all the slopes in an overall state vector $\alpha_t = (\mu_{1,t}, \dots, \mu_{d,t}, \beta_{1,t}, \dots, \beta_{d,t})'$. The system error of the dynamics of this common state vector will then be characterized by a block-diagonal variance matrix having a first $d \times d$ block accounting for the correlation among levels and a second $d \times d$ block accounting for the correlation among slopes. To be specific, suppose one has $d = 2$ series. Then $\alpha_t = (\mu_{1,t}, \mu_{2,t}, \beta_{1,t}, \beta_{2,t})'$ and the system equation is

$$\begin{pmatrix} \mu_{1,t+1} \\ \mu_{2,t+1} \\ \beta_{1,t+1} \\ \beta_{2,t+1} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} \mu_{1,t} \\ \mu_{2,t} \\ \beta_{1,t} \\ \beta_{2,t} \end{pmatrix} + \begin{pmatrix} \eta_{1,t} \\ \eta_{2,t} \\ \eta_{3,t} \\ \eta_{4,t} \end{pmatrix} \tag{10}$$

where $(\eta_{1,t}, \eta_{2,t}, \eta_{3,t}, \eta_{4,t})' \sim \mathcal{N}(0, Q)$ and

$$Q = \left[\begin{array}{c|c} Q_\mu & 0 \\ \hline 0 & Q_\beta \end{array} \right],$$

while the measurement equation is given by

$$\begin{pmatrix} y_{1,t} \\ y_{2,t} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \alpha_t + \begin{pmatrix} \epsilon_{1,t} \\ \epsilon_{2,t} \end{pmatrix}, \tag{11}$$

with $(\epsilon_{1,t}, \epsilon_{2,t})' \sim \mathcal{N}(0, H)$, where H is allowed to be non-diagonal.

In order to illustrate the previous example, we use data on the annual investment in Denmark and Spain from 1960 to 2000 (described in Petris et al., 2009, chap. 3.3.2). This dataset was collected from Petris et al. (2009) book website [1] and is shipped with the GKF package.

For this model we compare results from packages dlm (Petris (2010)) and GKF. We start by loading the data:

```
> data(GKF_data)
> invest <- GKF.data$invest
```

After that, we set up the model in a 'dlm' fashion

```
> library(dlm)
> mod <- dlm::dlmModPoly(2)
> mod$FF <- mod$FF %x% diag(2)
> mod$GG <- mod$GG %x% diag(2)
> W1 <- matrix(c(0.5, 0, 0, 0.5), 2, 2)
> W2 <- diag(c(49, 437266))
> W2[1, 2] <- W2[2, 1] <- 155
> mod$W <- bdiag(W1, W2)
> V <- diag(c(72, 14353))
> V[1, 2] <- V[2, 1] <- 1018
> mod$V <- V
> mod$m0 <- rep(0, 4)
> mod$C0 <- diag(4) * 1e4
```

The filtering in dlm is simply made by a call to the dlmFilter function, which only requires the data and the model object. After some minor transformation, we can easly obtain the filtered signal:

```
> filtered <- dlm::dlmFilter(invest, mod)
> alpha.filtered <- dlm::dropFirst(filtered$m)
> theta.filtered.dlm <- t(mod$FF %*% t(alpha.filtered))
```

The system object used by package GKF is deduced from the dlm object as follows:

[1] http://definetti.uark.edu/~gpetris/dlm/

```
> a0 = mod$m0
> P0 = mod$C0
> dt = rep(0, 4)
> ct = rep(0, 2)
> Tt = mod$GG
> Zt = mod$FF
> Qt = mod$W
> Ht = mod$V
> yt = t(as.matrix(invest))
```

Filtering is done as before in GKF by a call to FKF. We decided to set checkInputs to TRUE to replicate the system object accordingly, if needed.

```
> kf.GKF <- FKF(a0 = a0, P0 = P0, dt = dt, ct = ct, Tt = Tt, Zt = Zt, Qt = Qt,
+               Ht = Ht, yt = yt, checkInputs = TRUE)
> theta.filtered.GKF <- Zt %*% kf.GKF$att
```

Figure 2 (page 8) shows Denmark annual investment data together with filtered signal values from dlm and GKF. We notice that both graphs agree except in the initial starting points due to the Bayesian approach adopted by dlm, i.e, the effect of the prior distribution disapears after some time.

```
> i = 1
> plot(yt[i,],main = "Annual Denmark investments",col = "brown",xlab = NA, ylab = NA)
> lines(theta.filtered.dlm[,i],type = "l",lty = 6,pch = 20, col = "green")
> lines(theta.filtered.GKF[i,],type = "o",pch = 20, col = "blue")
```

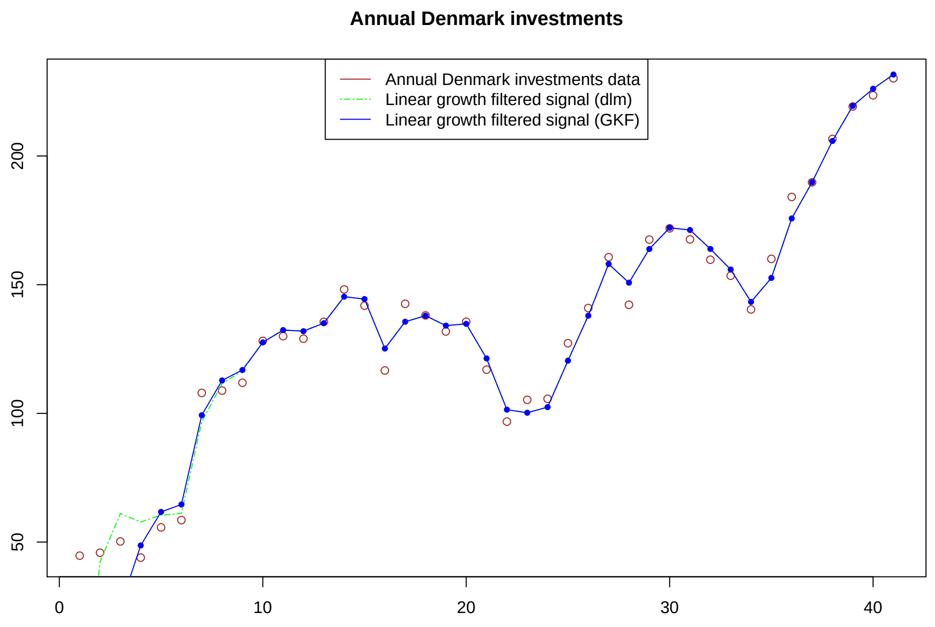

Figure 2: Denmark investment with filtered signal from dlm and GKF

4.2 Smoothing

The Smoothing distribution at time t is the conditional distribution of α_t (state smoothing) or θ_t(signal smoothing) given the entire observed data i.e, $y = (y_1', \dots, y_n')'$. This distribution is also Gaussian, and its characteristic elements can be computed by a (forward) recursive algorithm (see Durbin and Koopman, 2012, chap. 4.5.3). Note that `GKF` only computes signal smoothing i.e, $\hat{\theta} = \mathsf{E}[\theta_t|y]$ (the mode of $p(\theta_t|y)$) and $\text{var}[\theta_t|y]$ by adapting the disturbance smoothing recursions using the fact that $\theta_t = y_t - \epsilon_t$ (hence $\hat{\theta} = y_t - \hat{\epsilon}$) and $\text{var}(\theta_t|y) = \text{var}(\epsilon_t|y)$. The smoothed state can be easily obtained by applying the appropriate linear transformation. Again, the technical result recalled in *Lemma 1*, Appendix A is the main ingredient of the derivation.

We illustrate now how to obtain the smoothed signal for the (multivariate) linear growth example discussed earlier using `GKF`. Figure 3 (page 9) compares Spain annual investment data with filtered and smoothed signal values.

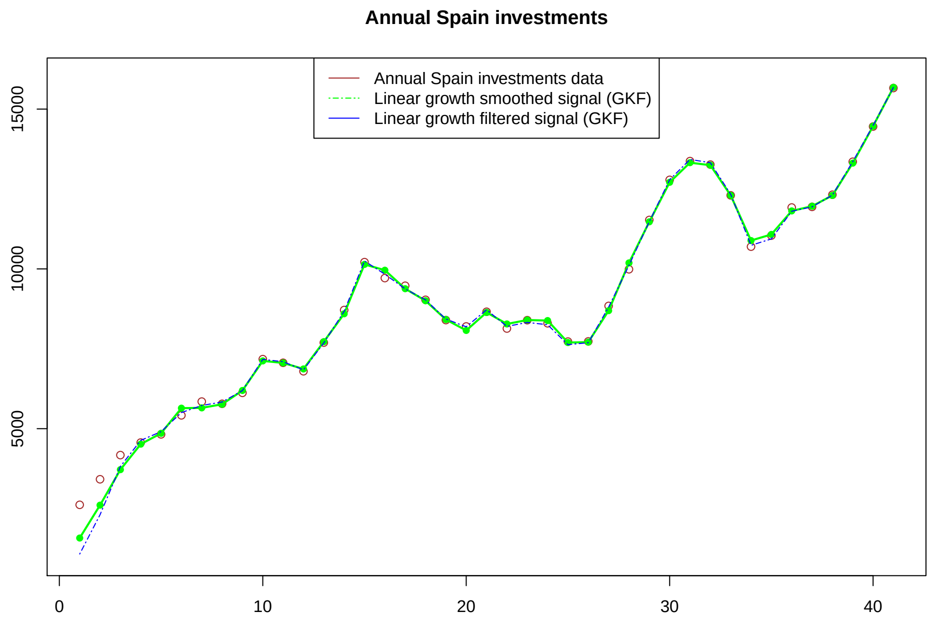

Figure 3: Spain investment with filtered and smoothed signal from `GKF`

We notice a small difference at the first time points which disapears quickly at the end of the observation window as higlighted by the following result:

```
> i = 2
> mat <- matrix(t(rbind(theta.smoothed.GKF[i, ], theta.filtered.GKF[i, ])), ncol = 2)
> colnames(mat) <- c("Smoothed", "Filtered")
> head(mat)

     Smoothed Filtered
[1,] 1577.616 1077.298
[2,] 2606.264 2306.147
[3,] 3717.433 3817.201
[4,] 4516.029 4632.766
[5,] 4855.204 4906.939
[6,] 5641.648 5503.007

> tail(mat)

      Smoothed Filtered
[36,] 11814.23 11804.01
```

```
[37,] 11960.81 11941.68
[38,] 12299.36 12309.66
[39,] 13309.56 13356.06
[40,] 14466.83 14490.67
[41,] 15681.27 15681.27
```

5 Sampling Smoother with GKF

In many simulation based techniques such as importance sampling or Markov Chain Monte Carlo (MCMC), it is imperative to be able to sample from the state/signal (or the disturbance/innovation) given the observed data that is for example drawing a sample $\theta = (\theta_1', \ldots, \theta_n')'$ given $y = (y_1', \ldots, y_n')'$. The first algorithms for the linear Gaussian model has been suggested as early as 1994 by Frühwirth-Schnatter (1994) followed by De Jong and Shephard (1995). A simpler and computationally more efficient algorithm inspired from the last cited paper was suggested by Durbin and Koopman (2002) and recalled in Durbin and Koopman (2012, chap. 4.9). This last technique also known as simulation smoothing by mean corrections is the one implemented in GKF. Note that only simulation smoother from the signal conditional distribution is available in GKF version 1.7.0.

Another family of simulation based algorithms known as Forward Filtering Backward Sampling (FFBS) has been developed independently by Carter and Kohn (1994) and Shephard (1994). The algorithm consists essentially in a simulation version of the Kalman smoother. This alternative was implemented in dlm.

Before showing how to sample from the signal posterior distribution, it is interesting to get a sense of the difference between conditional and unconditional sampling. We will use the Nile river data to illustrate this point.

We first start by simulating unconditionally on the data using the state equation (signal=state in the local level model)

```
> set.seed <- 1
> n <- length(yt)
> Nile.theta.uncond <- yt[1]
> for (i in 2:n) Nile.theta.uncond[i] <- Nile.theta.uncond[i-1] + rnorm(1,0,sqrt(Qt))
```

Then we use GKF functions to produce and sample from the signal posterior distribution.

```
> Nile.theta.Smooth <- GaussianSignalSmoothing(
+     a0 = a0, P0 = P0, dt = dt, ct = ct, Tt = Tt, Zt = Zt, Ht = Ht, Qt = Qt, yt = yt
+               )$theta
> Nile.theta.cond <- GaussianthetaSampling(
+     a0 = a0, P0 = P0, dt = dt, ct = ct, Tt = Tt, Zt = Zt, Ht = Ht, Qt = Qt, yt = yt,
+          M = 1)$theta
```

Figure 4 (page 11) shows just how badly the unconditional sample describes the data. On the other hand, the conditional sample looks rather satisfactory.

We briefly describe now the main ideas behind the simulation smoothing by mean corrections. First, note that if one has a sample from the posterior disturbance $\tilde{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)'$ then a sample from the posterior signal is simply obtained by $\tilde{\theta} = y - \epsilon$. The algorithm to simulate from the posterior disturbance and innovation can be done in three major steps as follows:

- Define $\omega = (\epsilon', \eta')'$. One starts by drawing a sample ω^+ from $p(\omega)$ where $p(\omega) \sim \mathcal{N}(0, \Omega)$, $\Omega = \mathrm{diag}(H_1, \ldots, H_n, Q_1, \ldots, Q_n)$. This sampe is then used to generate a sample y^+ from the observation by means of a recursive application of the measurement and transition equations (2) and (3), where the recursion is initialized by the draw $\alpha_1^+ \sim (a_1, P_1)$.
- Use the Kalman filter forward recursion together with the smoothing backward recursion to compute $\hat{\omega} = (\hat{\epsilon}', \hat{\eta}')' = \mathsf{E}[\omega|y]$ and $\hat{\omega}^+ = (\hat{\epsilon}^{+\prime}, \hat{\eta}^{+\prime})' = \mathsf{E}[\omega^+|y^+]$.
- take the sample $\tilde{\omega} = \hat{\omega} - \hat{\omega}^+ + \omega^+$.

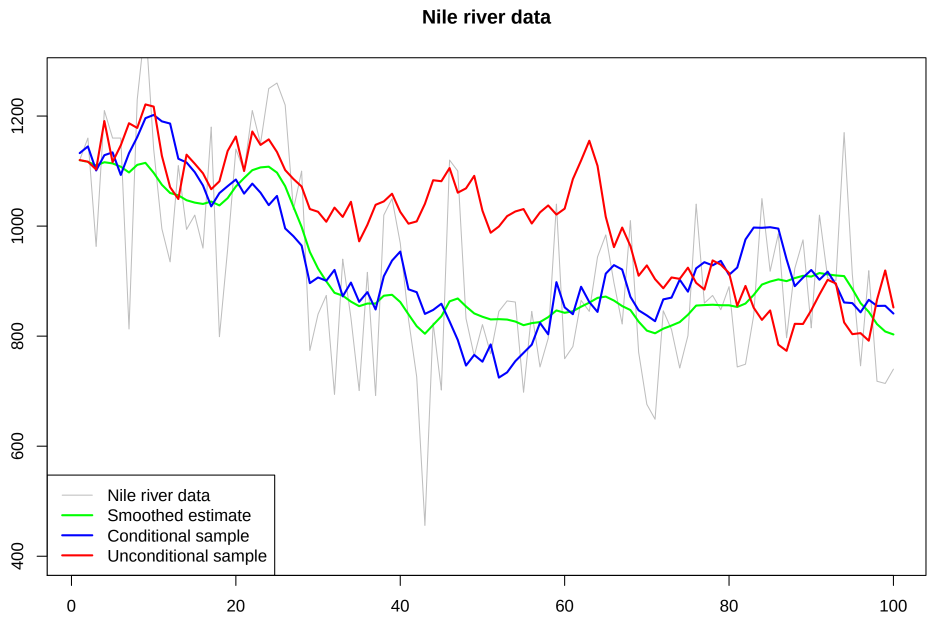

Figure 4: Comparison of the conditional and unconditional signal vector simulated sample.

Finally we show how to sample from `GKF` and `dlm`. `GKF` provides the function `GaussianthetaSampling` which allows to sample `M` times from the posterior signal distribution. It requires the system objects as usual plus some optional inputs such as the number of samples. We use the linear growth model to show its use.

```
> data(GKF_data)
> invest <- GKF.data$invest
> mod <- dlm::dlmModPoly(2)
> mod$FF <- mod$FF %x% diag(2)
> mod$GG <- mod$GG %x% diag(2)
> W1 <- matrix(c(500,0,0,500), 2, 2)
> W2 <- diag(c(49, 437266))
> W2[1, 2] <- W2[2, 1] <- 155
> mod$W <- bdiag(W1, W2)
> V <- diag(c(72, 14353))
> V[1, 2] <- V[2, 1] <- 1018
> mod$V <- V
> mod$m0 <- rep(0, 4)
> mod$C0 <- diag(4) * 1e4
> ## Rcpp
> a0 = mod$m0; P0 = mod$C0; dt = rep(0, 4); ct = rep(0, 2)
> Tt = mod$GG; Zt = mod$FF; Qt = mod$W; Ht = mod$V
> yt = t(as.matrix(invest))
```

`dlm` provides the function `dlm::dlmBSample` to sample from the state posterior distribution based on the FFBS algorithm. It returns only one sample at a time and hence needs to be called in a loop if many samples are desired. It only requires the filtered object (output of `dlm::dlmFilter`).

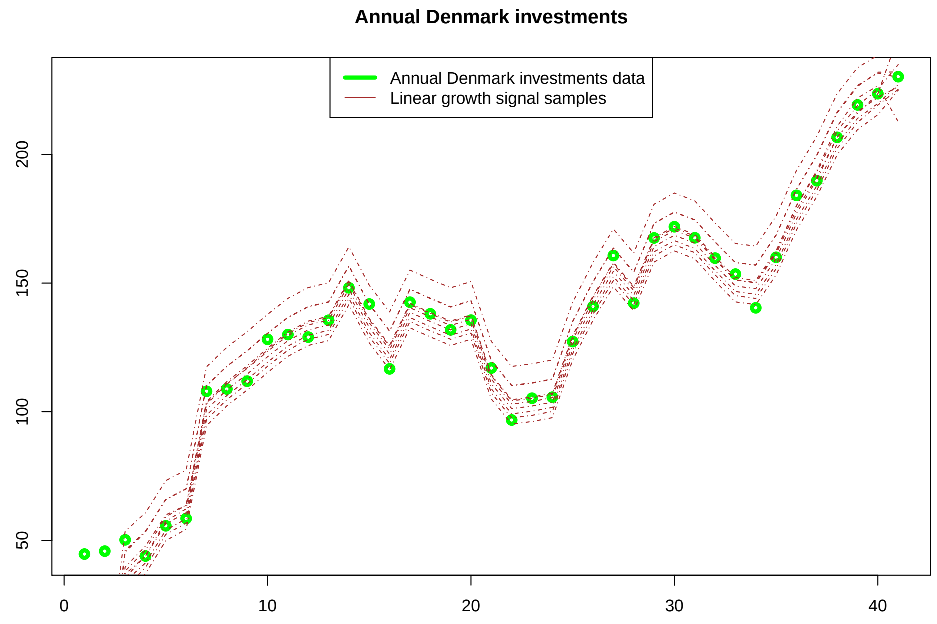

Figure 5: Spain investment data with 10 sampled signal from `GKF`

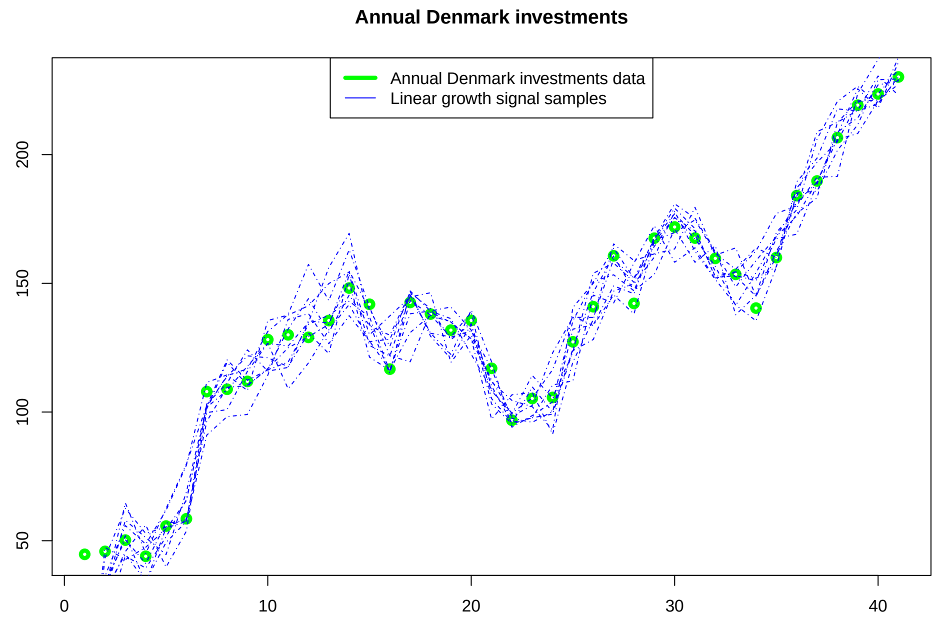

Figure 6: Spain investment data with 10 sampled signal from `dlm`

6 Additional multivariate example

In this section we discuss the full treatment of an additional multivariate example.

6.1 A time varying linear Gaussian model: the Dynamic Capital Asset Pricing model

The capital asset pricing model is used to calculate the required rate of return for any risky asset (relative to a risk-free asset). A simplified version of the CAPM assumes that the excess return on a risky asset in any time period is proportional, on average, to the excess return on a portfolio representing the entire market. The proportionality constants are known as *betas*. Considering several assets at once, and allowing the betas to vary over time, one can set up the following multivariate dynamic regression model

$$\begin{aligned} (r_{a,t} - r_{rf,t}) &= \beta_t(r_{m,t} - r_{rf,t}) + \epsilon_t & \epsilon_t \sim \mathcal{N}(0, \Sigma_\epsilon) \\ \beta_{t+1} &= \beta_t + \eta_t & \eta_t \sim \mathcal{N}(0, \Sigma_\eta) \end{aligned} \tag{12}$$

where $r_{a,t}$ is the return of the risky asset a, $r_{rf,t}$ is the return of the risk free asset (the reference) and $r_{m,t}$ is the return of the market (an index for example) at time $t, t = 1, \ldots, n$. Note that Σ_ϵ and Σ_η are variance matrices accounting for correlated observation errors and correlated changes of the betas, respectively. The data we will use for this example are monthly returns on four stocks (Mobil, IBM, Weyer, and Citicorp) from January 1978 to December 1987, together with the 30-day Treasury Bill as a proxy for the risk-free asset. The value-weighted average returns for all the stocks listed at the New York and American Stock Exchanges will be used as a proxy for the market returns. The data, originally used in Berndt (1991), was collected online [2] and are shipped with package `GKF`.

In state space notations, model (12) can be summarized in the following system variables:

$$\begin{aligned} y_t = (r_{a,t} - r_{rf,t}) \quad & c_t = 0 \quad & Z_t = (r_{m,t} - r_{rf,t}) I_d \quad & H_t = \Sigma_\epsilon \\ \alpha_t = \beta_t \quad & d_t = 0 \quad & T_t = I_m \quad & Q_t = \Sigma_\eta \end{aligned} \tag{13}$$

where $m = d = 4$ is the number of assets considered. We prepare the data for the analysis first:

```
> data(GKF_data)
> tmp <- GKF.data$CAPMData
> pkDat <- GKF.data$CAPMData
> y <- tmp[, 1:4] - tmp[, "RKFREE"]
> colnames(y) <- colnames(tmp)[1:4]
> market <- tmp[, "MARKET"] - tmp[, "RKFREE"]
> rm("tmp")
> m <- ncol(y)
```

One can start by estimating the observation and transition variances by maximum likelihood. We will take advantage of the symmetric property of the matrices by parameterizing each one in terms of its log Choleski decomposition. Note that the number of unknown in each matrix is $k = m(m+1)/2 = 10$.

```
> k <- m*(m+1) * 0.5
> n <- length(market)
> ## Estimate the variances
> yt <- t(y)
> dt = rep(0, m)
> ct = rep(0, m)
> Zt <- sapply(seq_along(market), function(i) market[i] %x% diag(m))
> dim(Zt) <- c(m, m, n)
> Tt = diag(nr = m)
> a0 = rep(0, m)
> P0 = diag(1e7, nr =  m)
> VarLik <- function(theta) {
+     a <- diag(exp(0.5 * theta[1:m]), nr = m)
+     a[upper.tri(a)] <- theta[(m + 1):k]
+     Ht <- crossprod(a)
```

[2] http://definetti.uark.edu/~gpetris/dlm/

```
+     a <- diag(exp(0.5 * theta[1:m + k]), nr = m)
+     a[upper.tri(a)] <- theta[-(1:(k + m))]
+     Qt <- crossprod(a)
+     -FKF(yt = yt, ct = ct, dt = dt, Zt = Zt, Tt = Tt,
+          Ht = Ht, Qt = Qt, a0 = a0, P0 = P0)$logLik
+ }
> fit <- optim(par = rep(0, 2 * k),fn = VarLik, method = "BFGS",
+              control = list(maxit = 500))
> theta <- fit$par
> a <- diag(exp(0.5 * theta[1:m]), nr = m)
> a[upper.tri(a)] <- theta[(m+1):k]
> Ht <- crossprod(a)
> a <- diag(exp(0.5 * theta[1:m + k]), nr = m)
> a[upper.tri(a)] <- theta[-(1:(k + m))]
> Qt <- crossprod(a)
```

The next step is to compute the betas from the smoothed signal in the usual way by a call to the `GaussianSignalSmoothing` function

```
> smoothCAPM <- GaussianSignalSmoothing(yt = yt, ct = ct, dt = dt, Zt = Zt, Tt = Tt,
+                                        Ht = Ht, Qt = Qt, a0 = a0, P0 = P0)
> thetas <- t(smoothCAPM$theta)
> betas <- matrix(0, n, m)
> for (i in 1:length(market))  betas[i, ] <- t(solve(Zt[, , i])%*%thetas[i, ])
> betas <- ts(betas, start = start(market), freq = frequency(market))
```

Except for the Mobil stock, Figure 7 (page 14) shows that all the other betas changes considerably over time which justifies the dynamic assumption.

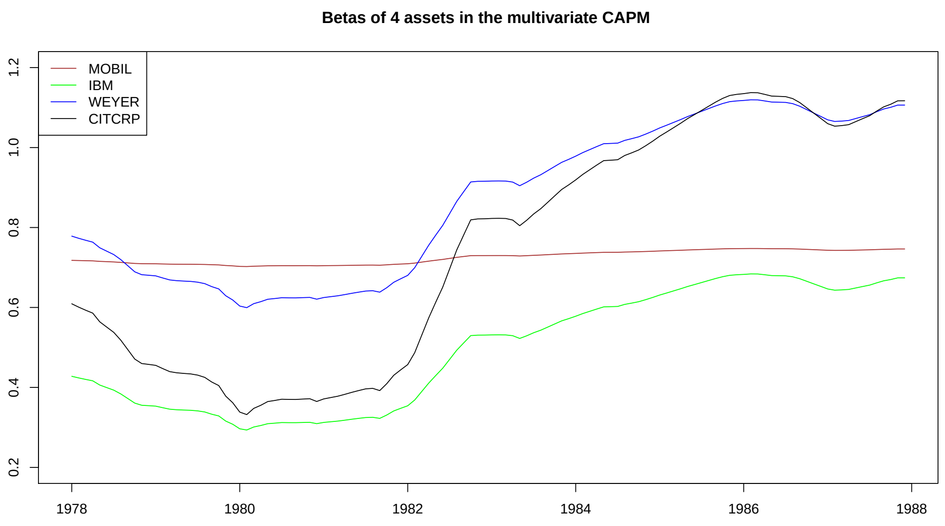

Figure 7: Multivariate dynamic CAPM : betas visualization

If a sample from the posterior distribution of the betas is desired, one needs to call `GaussianthetaSampling` with the usual inputs. Figure 8 (page 15) illustrates this task for the IBM stock:

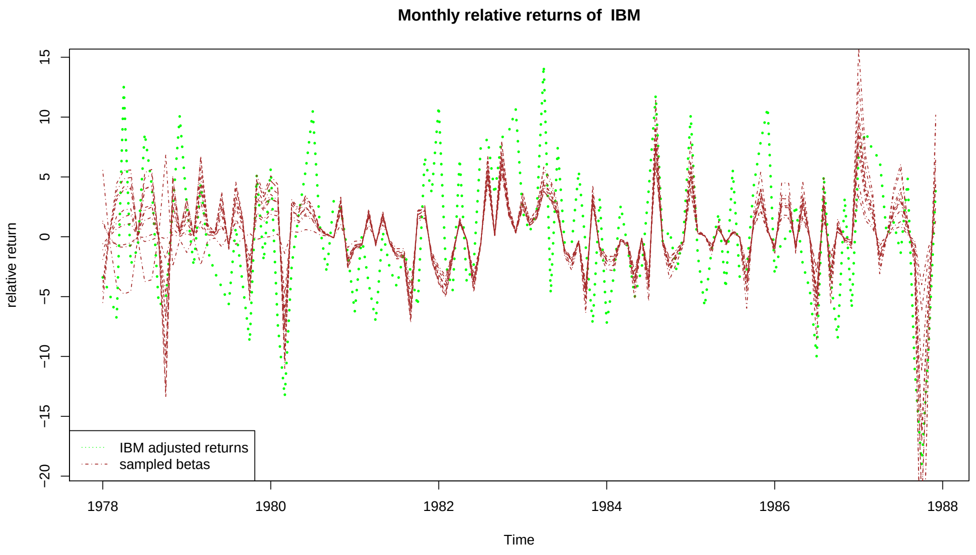

Figure 8: 10 samples from the posterior signal distribution of IBM returns together with the real data.

7 A nonLinear non-Gaussian example: the Poisson scoring model

In the nonlinear non-Gaussian case, the simplfying Gaussian assumption is dropped and as a consequence the computations are slightly more complicated. The model considered here is the transition Equation (3) together with the observation Equation (1).

7.1 Computing the posterior mode

We first note that the filtering and smoothing operations described before are equivalent to the computation of the posterior mode of $p(\theta_t|y_1,\dots,y_t)$ and $p(\theta_t|y_1,\dots,y_n)$ respectively (together with the associated Hessian matrix). For non-Gaussian models, these computations are usually not doable in analytical form but can be obtained by maximization of $p(\theta_*|y)$ using for example the Newton-Raphson algorithm. This approach was suggested by Jungbacker and Koopman (2007) and is the one implemented in `GKF`.

For a given guess g of the optimum $\hat{\theta}$, the Newton-Raphson method proposes the updated guess g^+ defined by

$$g^+ = g - \{\ddot{p}(\theta|y)|_{\theta=g}\}^{-1}\dot{p}(\theta|y)|_{\theta=g},$$

where $\dot{p}(.|.) = \frac{\partial \log p(.|.)}{\partial\theta}$ and $\ddot{p}(.|.) = \frac{\partial^2 \log p(.|.)}{\partial\theta\partial\theta'}$. It can be shown that the updating step can be written as

$$\begin{aligned} g^+ &= g - \{\ddot{p}(y|\theta)|_{\theta=g} - \Psi^{-1}\}^{-1}\{\dot{p}(y|\theta)_{\theta=g} - \Psi^{-1}(g-\mu)\} \\ &= (\Psi^{-1} + A^{-1})^{-1}(A^{-1}x + \Psi^{-1}\mu), \end{aligned} \quad (14)$$

where $A = -\{\ddot{p}(y|\theta)|_{\theta=g}\}^{-1}$ and $x = g + A\dot{p}(y|\theta)_{\theta=g}$, Ψ and μ defined in (7).

If we can assure that $\ddot{p}(y|\theta)|_{\theta=g}$ is negative definite for all values of θ, the Kalman filter and smoother recursions can be used to compute g^+ by taking model (3)-(2) with $y = x$ and $H = A$. Note that the Hessian is simply given by $G = \{\ddot{p}(\theta|y)|_{\theta=\hat{\theta}}\}^{-1} = -\Psi^{-1} - A^{-1}$. However, if this assumption cannot be made, Jungbacker and Koopman (2007, Theorem 1) state that a Kalman filter similar approach could be used to compute g^+. The suggested algorithm is summarized below:

- First, compute the Kalman filter forward recursions with $y_t = x_t$ and $H_t = A_t$ (where x and A are defined by Equation (14))

$$\begin{aligned} v_t &= x_t - c_t - Z_t\alpha_t, & F_t &= A_t + Z_tP_tZ_t', & \\ & & K_t &= T_tP_tZ_t'F_t^{-1}, & t = 1,\dots,n, \\ \alpha_{t+1} &= d_t + T_t\alpha_t + K_tv_t, & P_{t+1} &= T_tP_tT_t' - K_tF_tK_t' + Q_t, & \end{aligned} \quad (15)$$

- Then, carry out the following backward recursion using quantities previously computed.

$$e_t = F_t^{-1} v_t - K_t' s_t, \qquad s_{t-1} = Z_t' e_t + T_t' s_t, \qquad g_t^{+} = x_t - A_t e_t \tag{16}$$

  for $t = n, \dots, 1$ with initialization $s_n = 0$.

Function `NRUpdatingStep` implements this procedure and is called iteratively until convergence in function `PosteriorSignalMode` to obtain the posterior mode $\hat{\theta}$.

7.2 Sampling from the proposal density

In the nonlinear non-Gaussian case, it is usually not feasible to sample from $p(\theta|y,\psi)$. Hence one needs to find a density $f(\theta|y,\psi)$ that is similar to $p(\theta|y,\psi)$ and from which samples can be obtained efficiently. This density is known as the proposal density. `GKF` implements a Gaussian proposal density with the same mode as the target and with the same curvature around the mode that is

$$f(\theta|y,\psi) = \mathcal{N}(\hat{\theta}, V), \quad V = -G^{-1}, \quad G = \{\ddot{p}(\theta|y)|_{\theta=\hat{\theta}}\}^{-1} = -\Psi^{-1} - A^{-1} \tag{17}$$

where A is defined as before and evaluated at $\theta = \hat{\theta}$.

Again, if we can assure that A is positive definite, the linear Gaussian sampling method of Durbin and Koopman (2002) (`GKF` function `GaussianthetaSampling`) can be called with $y = \hat{\theta} + A\dot{p}(y|\theta)|_{\theta=\hat{\theta}}$ and $H = A$. If this assumption cannot be made, one can use Jungbacker and Koopman (2007, Theorem 2) which only requires $\ddot{p}(\theta|y)$ to be invertible. If we define A as in (17) and $x = \hat{\theta} + A\dot{p}(y|\theta)|_{\theta=\hat{\theta}}$, a sample θ^i from the proposal density $f(\theta|y,\psi)$ defined in (17) is generated by first computing (15) at the x points along with the following recursions

$$\begin{aligned} C_t &= A_t^{-1} - F_t^{-1} - K_t' N_t K_t, & R_t &= C_t^{-1}(A_t^{-1} Z_t - K_t' N_t T_t), \\ \omega_t &\sim \mathcal{N}(0, C_t), & u_t &= A_t = (\omega_t + F_t^{-1} v_t - K_t' r_t), \\ r_{t-1} &= Z_t' A_t^{-1} u_t - R_t' \omega_t + T_t' r_t, & R_{t-1} &= R_t' C_t R_t - Z_t' A_t^{-1} Z_t + T_t' N_t T_t, \end{aligned} \tag{18}$$

for $t = n, \dots, 1$ with usual initialization $r_n = 0$ and $N_n = 0$. The simulation θ^i from $f(\theta|y,\psi)$ is obtained by $\hat{\theta} + (u_1', \dots, u_n')'$. This method is implemented in `GKF` (function `PosteriorSignalSampling`) and allows users to generate more than one sample. It is recommended to set the seed before sampling especially if the samples will be used in importance sampling (or any similar simulation based estimation method).

Assume a sample θ^i is obtained from $f(\theta|y,\psi) = \mathcal{N}(\hat{\theta}, V)$ using (18). The log-density $\log(f(\theta^i|y,\psi))$ is given by

$$\log(f(\theta^i|y,\psi)) = -\frac{dn}{2}\log 2\pi - \sum_{t=1}^{n} \log|\det A_t| - \sum_{t=1}^{n} \log(\det B_t) - \frac{1}{2}\sum_{t=1}^{n} b_t^{i\prime} b_t^i, \tag{19}$$

where b_t^i and B_t implicitely defined as $\omega_t^i = B_t b_t^i$ and $C_t = B_t B_t'$, respectively. Jungbacker and Koopman (2007) proved that the matrix C_t is definite semi-positive however, numerical instabilities may appear and abort the computation if C_t is not carefully checked before using it, see Section B for more details.

7.3 Likelihood evaluation

We noted previously that some system vectors and matrices can depend on a vector of parameter ψ. Following Durbin and Koopman (2012), we opt for the method of maximum likelihood to estimate the parameters collected in ψ which produces optimal properties in large samples.

The likelihood function for the vector of observation y is based on the observation density and the time independence assumption (1) and is given by

$$L(\psi) = p(y|\psi) = \int p(y,\theta|\psi)d\theta = \int p(y|\theta,\psi)p(\theta|\psi)dz \tag{20}$$

which needs to be evaluated at different values of the parameter vector ψ in reasonable computing times as it will be called several times by the optimization routine. An analytical solution to evaluate the previous integral is not available and hence numerical evaluation will be considered. It is well established that numerical integration in multi-dimensional space becomes quickly infeasible when the dimension increases. Consequently, we will adopt simulation based techniques. Given that we know how to draw samples from $p(\theta|\psi)$ and have usually explicit expression for $p(y|\theta, \psi)$, one may think that the problem can be solved by 'direct' Monte Carlo estimate as follows:

- draw M independent samples $\theta^{(k)}$ from $p(\theta|\psi)$
- get the estimate $\hat{L}(\psi)$ by avereging $p(y|\theta^{(k)}, \psi)$, i.e, $\hat{L}(\psi) = \sum_{k=1}^{M} \frac{1}{M} p(y|\theta^{(k)}, \psi)$

However, as highlighted in figure 4, sampling unconditionnally on the data is not efficient and hence leads to poor estimates. A more effective approach is based on impotance sampling as advocated by Durbin and Koopman (see 2012, chap. 11). Starting from (20), we can modify the likelihood

G: (1) Tarak, you may wish to use more environment align or aligned. I have changed it at a couple of places as an example. Here I used "aligned" to mimic your layout but it may be better to use "align" and number only the last row. To leave more space between lines, use the option argumnent of \\, as in the last line below.

(2) It is more usual to write $E_p(...)$ rather than $E(...)_p$

$$\begin{aligned} L(\psi) = p(y|\psi) &= \int p(y, \theta|\psi) d\theta \\ &= \int p(y|\theta, \psi) p(\theta|\psi) dz \\ &= \int \frac{p(y|\theta, \psi) p(\theta|\psi)}{p(\theta|y, \psi)} p(\theta|y, \psi) dz \\ &= \mathsf{E}\left(\frac{p(y|\theta, \psi) p(\theta|\psi)}{p(\theta|y, \psi)}\right)_p \end{aligned} \tag{21}$$

where $\mathsf{E}()_p$ denotes expectation with respect to density $p(\theta|y, \psi)$. This expectation can be evaluated in the usual way by sampling from $p(\theta|y, \psi)$ and averaging the importance weights $q = \frac{p(y|\theta,\psi)p(\theta|\psi)}{p(\theta|y,\psi)}$. To summarize, the importance sampling estimate is given by

$$\hat{L}(\psi) = \tfrac{1}{M} \sum_{i=1}^{M} q_i, \quad q_i = \tfrac{p(y|\theta^i,\psi)p(\theta^i|\psi)}{p(\theta^i|y,\psi)}, \quad \theta^i \sim p(\theta|y, \psi), \tag{22}$$

where q_i is referred to as an important weight, which can be computed using (1), (19) and ideas discussed in Appendix C. The samples θ^i are obtained from the sampling smoother algorithm described in 7.2. For purpose of likelihood maximisation with respect to ψ, it is preferred to work with the log-likelihood function. However, taking the logarithm in (22) introduces bias that can be accounted for as explained in Appendix D.

7.4 A nonlinear non-Gaussian example

7.4.1 The model

The example selected is a simple model to forecast the number of goals scored (build a probabilty table for all possible outcomes) by a football team based on its past performance. We assume that the number of goals scored by a team can be modelled by a Poisson process where the (log)intensity follows a Markovian process

$$\begin{aligned} \mathsf{P}(y_t = x|\lambda_t) &= \text{Poiss}(\lambda_t) \\ \lambda_t &= \exp(\alpha_t) \\ \alpha_{t+1} &= \phi_\alpha \alpha_t + \eta_{\alpha,t} \end{aligned} \tag{23}$$

Model (23) considers only one team scoring process. If more than one team (k teams) is of interest, the associated multivariate model can be obtained as follows. Let $y_t = (y_{1,t}, \dots, y_{k,t})$ be the number of goals scored by the k teams at time t and define the intensity λ_t and state (equal to the signal in this case) vectors similarly. Assuming independence between goals [3] scored by each team at time t, one gets

$$\begin{aligned} \mathsf{P}(y_t|\alpha_t) &= \prod_{i=1}^{k} \text{Poiss}(\lambda_{t,i}) \\ \lambda_{t,i} &= \exp(\alpha_{t,i}),\ \ i = 1, \dots, k \\ \alpha_{t+1} &= \Phi_\alpha \alpha_t + \eta_\alpha \end{aligned} \tag{24}$$

where $\Phi_\alpha = \text{diag}(\phi_\alpha)$ and $\eta_\alpha \sim (0, Q),\ Q = \text{diag}(\sigma^2)$. Assuming independence between the n time observations and defining $y = (y'_1, \dots, y'_n)'$ and similarly $\alpha = (\alpha'_1, \dots, \alpha'_n)'$, we get : $\mathsf{P}(y|\alpha;\psi) = \prod_{t=1}^{n} \mathsf{P}(y_t|\alpha_t;\psi)$ where ψ is the vector of parameters defined as $\psi = (\phi, \sigma^2)$ which can be easily computed in R by

```
> dP <- function(theta,y){
+     lambda <- exp(theta)
+     sum(dpois(as.numeric(y), as.numeric(lambda), T), na.rm = TRUE)
+ }
```

Note that the function should accept NA values and only return numeric which was done in dP by setting na.rm=TRUE in dpois.

7.4.2 The data

The data collects the goals scored in games between 10 selected English Premier League football teams [4] from season 2005-2006 to season 2012-2013 which gave us 180 ($10 \times 9 \times 2$) observation per season. The data was divided in 2 sets. A short one with only 2 seasons (2011-2012 / 2012-2013) called shortGoals and the longer one with the entire dataset called longGoals [5].

7.4.3 Study a nonlinear non-Gaussian model with GKF

As discussed earlier, in order to fit a nonlinear non-Gaussian model one needs to provide the first and second derivatives of the conditional log-density $p(y|\theta) = p(y|\alpha)$. Those derivatives are given in this example by:

$$\dot{p}(y_t|\theta_t) = y_t - \exp(\theta_t)\ ,\ \ddot{p}(y_t|\theta_t) = \text{diag}(-\exp(\theta_t))$$

and can be defined in R as follows:

```
> jac_ <-  function(theta, y, Pars = list()) {
+     res <- y - exp(theta)
+     res[is.na(res)] = 0
+     res
+ }
> hess_ = function(theta, y, Pars = list()) diag(-exp(theta))
```

Note the extra parameter Pars in these functions. Pars is used to pass extra inputs, when needed, to compute the derivatives. Morever, both routines need to handle missing values in y. We recommand to set the derivative component related to a missing value $y_{t,k}$ to 0. In the previous example, only the first derivative deals with missing values because the hessian expression is independent of y.

The model is then built up, the data-set selected and the parameters $\phi = 0.9975$ and $\sigma^2 = 0.000205$ are specified.

[3]We are fully aware that this assumption is not verified in reality. In fact, if team A plays team B, the goals scored by team A is highly influenced by the goals scored by team B. However, the aim of this example is illustrative and hence we assumed independence to keep things simple.

[4]Arsenal, Aston Villa, Chelsea, Everton, Fulham, Liverpool, Man City, Man United, Tottenham, Wigan.

[5]the data is available at http://www.football-data.co.uk/

```
> data(GKF_data)
> yt.long <- GKF.data$longGoals
> yt.short <- GKF.data$shortGoals
> d <- nrow(yt)
> n <- ncol(yt)
> m <- d
> phi <- 0.9975
> sig2 <- 0.000205
> P0 <- diag(sig2, m, m)
> a0 <- numeric(m)
> Zt <- diag(1, d, m)
> dt <- numeric(m)
> ct <- numeric(d)
> Tt <- diag(phi, m, m)
> Qt <- diag(sig2, m, m)
```

A more interesting example considers the vector of parameters ψ unknown and estimates it by maximum likelihood based on importance sampling as described in Section 7.3. The requirements for this method are the following

1. Provide a routine to compute the log-likelihood. The first argument of this function should be the parameter `psi`. Other inputs are the seed used to generate the importance sample, the observation matrix and the number of sample used. Note that the seed is required to be the same for the generation of the `M` simulated paths of θ in order to obtain a smooth multi-dimensional likelihood surface in ψ for its maximization (Jungbacker and Koopman, 2007). Although small, the bias correction needs to be included if the logarithm of the importance weights is used directly in the estimator, see Appendix D.

```
> computeLogLik <- function(psi, seed=345, yt=yt.long, M=50){
+     set.seed(seed)
+     phi <- psi[1]
+     sig2 <- psi[2]
+
+     ## dim
+     d <- nrow(yt)
+     n <- ncol(yt)
+     m <- d
+
+     ## dlm objects
+     P0 <- diag(sig2, m, m)
+     a0 <- numeric(m)
+     Zt <- diag(1, d, m)
+     dt <- numeric(m)
+     ct <- numeric(d)
+     Tt <- diag(phi, m, m)
+     Qt <- diag(sig2, m, m)
+
+     thetaSample <- PosteriorSignalSampling(a0, P0, dt, ct, Tt, Zt, Qt, yt,
+                                            SpecificPars = list(jac = jac_, hess = hess_),
+                                            M = M, type = "NLnonGaussian")
+     loglikV <- numeric(M)
+     for (i in 1:M){
+         thSample <- thetaSample$theta[, , i]
+         loglikV[i] <- dP(thSample, yt) + thetaSample$UncondthetaLogLik[i] -
+                                           thetaSample$CondthetaLogLik[i]
+     }
+     bias.corr <- log(1 + var(loglikV) / (2*M))
+     return(- mean(loglikV) - bias.corr )
+ }
```

2. Choose an optimization routine together with an initial guess for ψ. Following standard procedure, we minimize the inverse log-liklihood instead of maximizing the (log-)likelihood. We have tried several optimization algorithms (using the `optimx` wrapper (Nash, 2014, see)). We found that the `R` box-constrained optimization using PORT routines `mlminb` works best (smallest objective value, minimum number of iterations and shortest execution time).

   ```
   > phi <- 0.5
   > sig2 <- 1
   > Par <- nlminb(start = c(phi, sig2),  objective = computeLogLik,
   +               lower = c(1e-3, 1e-3), upper = c(1, 4))
   > MLParams <- Par$par
   > phi <- MLParams[1]
   > sig2 <- MLParams[2]
   ```

To illustrate the result, we can use the estimated parameters to construct the posterior signal mode (best estimate of the signal), convert it to scoring intensity (just apply the exponential) and then plot the scoring intensity for a given team (by setting the `team_index` to a number between 1 and 10) together with the observed scored goals. Recall that the scoring intensity in a Poisson model coincides with its expected value (the mean).

```
> thetaHat <- PosteriorSignalMode(a0, P0, dt, ct, Tt, Zt, Qt, yt,
+                                 jac_, hess_, list(), 0)$theta_hat
> lambdaHat <- exp(thetaHat)
> teams <- rownames(lambdaHat)
```

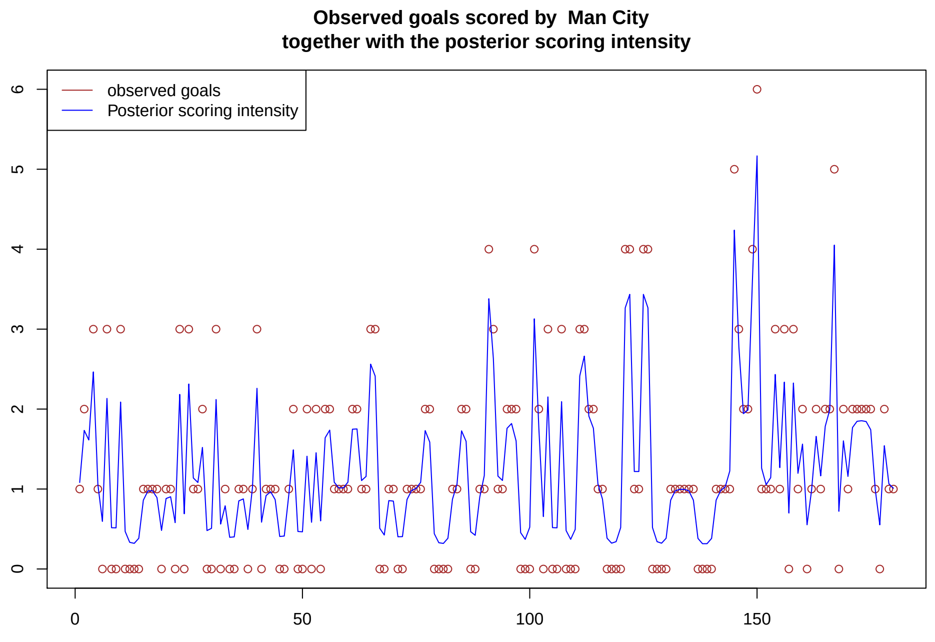

Figure 9: Manchester City observed scored goals plotted against its AR-Poisson scoring intensity

Figure 9 shows that the scored goals highly fluctuates over time whcih makes a constant scoring intensity unrealistic. Althought simple, the dynamic AR-Poisson model renders better the evolution of the scoring process and can be considered as a good improvement to the static Poisson counting process. Readers interested in a more elaborated model for forecasting football match results are referred to Koopman and Lit (2014).

8 Conclusion

In this paper we introduce the different functionalities available in package `GKF`. To our knowledge, `GKF` is the first `R` package to offer facilities to fit non-linear non-Gaussian models when the observations are not assumed to come from a log-concave density. Most of the computation in `GKF` are done in `C++`, which makes it fast enough to be able to fit high-dimensional state space models to large datasets. We showed in this paper different examples and how `GKF` can be used to fit them. Although simple, these examples can be used as templates for more elaborated application. The package is also shipped with several datasets to ease the reproduction of the presented computations.

References

Berndt, E. R. (1991). *The practice of econometrics: classic and contemporary.* Addison-Wesley Reading, MA.

Borsdorf, R. and Higham, N. J. (2010). A preconditioned newton algorithm for the nearest correlation matrix. *IMA Journal of Numerical Analysis*, 30(1):94–107.

Boyle, J. P. and Dykstra, R. L. (1986). A method for finding projections onto the intersection of convex sets in hilbert spaces. In *Advances in order restricted statistical inference*, pages 28–47. Springer.

Carter, C. K. and Kohn, R. (1994). On gibbs sampling for state space models. *Biometrika*, 81(3):541–553.

De Jong, P. and Shephard, N. (1995). The simulation smoother for time series models. *Biometrika*, 82(2):339–350.

Durbin, J. and Koopman, S. J. (2002). A simple and efficient simulation smoother for state space time series analysis. *Biometrika*, 89(3):603–616.

Durbin, J. and Koopman, S. J. (2012). *Time series analysis by state space methods. Second edition.* Oxford University Press.

Frühwirth-Schnatter, S. (1994). Data augmentation and dynamic linear models. *Journal of time series analysis*, 15(2):183–202.

Helske, J. (2017). KFAS: Exponential family state space models in R. *Journal of Statistical Software*, 78(10):1–39.

Higham, N. J. (2002). Computing the nearest correlation matrix — a problem from finance. *IMA journal of Numerical Analysis*, 22(3):329–343.

Jungbacker, B. and Koopman, S. J. (2007). Monte carlo estimation for nonlinear non-gaussian state space models. *Biometrika*, 94(4):827–839.

Koopman, S. J. and Lit, R. (2014). A dynamic bivariate poisson model for analysing and forecasting match results in the english premier league. *Journal of the Royal Statistical Society: Series A (Statistics in Society).*

Li, Q. and Qi, H.-d. (2011). A sequential semismooth newton method for the nearest low-rank correlation matrix problem. *SIAM Journal on Optimization*, 21(4):1641–1666.

Luethi, D., Erb, P., and Otziger, S. (2014). *FKF: Fast Kalman Filter.* R package version 0.1.3.

Maechler, M. and Bates, D. (2006). 2nd introduction to the matrix package.

Nash, J. C. (2014). On best practice optimization methods in R. *Journal of Statistical Software*, 60(2):1–14.

Petris, G. (2010). An R package for dynamic linear models. *Journal of Statistical Software*, 36(12):1–16.

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

Sanderson, C. et al. (2010). Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. *Report Version*, 2.

Shephard, N. (1994). Partial non-gaussian state space. *Biometrika*, 81(1):115–131.

A Multivariate regression theory: Basic result

We recall a basic result that can be seen as representing the regression of a vector x on another vector y when their joint distribution is multivariate normal i.e, we would like to characterise the distribution of $x|y$ assuming

$$\mathsf{E}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \quad \text{var}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma'_{xy} & \Sigma_{yy} \end{bmatrix} \tag{25}$$

where Σ_{yy} is supposed to be a nonsingular matrix.

Lemma A.1. *The conditional distribution of x given y is normal with mean vector*

$$\mathsf{E}\left(x|y\right) = \mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y),$$

and variance matrix

$$\text{var}\left(x|y\right) = \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma'_{xy}.$$

Proof. Let $z = x - \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y)$. Since the transformation $(x, y) \to (z, y)$ is linear and (x, y) is normally distributed, (z, y) is also normally distributed. Moreover, we have,

$$\begin{aligned} \mathsf{E}(z) &= \mu_x \\ \text{var}(z) &= \mathsf{E}[(z - \mu_x)(z - \mu_x)'] \\ &= \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma'_{xy}, \\ \text{cov}(y, z) &= \mathsf{E}[y(z - \mu_x)'] \\ &= \mathsf{E}[y(x - \mu_x)' - y(y - \mu_y)'\Sigma_{yy}^{-1}\Sigma'_{xy}] \\ &= 0 \end{aligned} \tag{26}$$

Hence z and y are uncorrelated i.e, independent (due to the Gaussian assumption). Therefore, the conditional distribution of z knowing y is the same at its unconditional distribution given by (26). Since $x = z + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y)$, it follows that

$$\begin{aligned} \mathsf{E}(x|y) &= \mathsf{E}(z|y) + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y) \\ &= \mathsf{E}(z) + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y) \\ &= \mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y) \\ \text{var}(x|y) &= \text{var}(z|y) \\ &= \text{var}(z) \\ &= \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma'_{xy} \end{aligned} \tag{27}$$

which is the claimed result. □

Define now $v_t = y_t - \mathsf{E}(y_t|Y_{t-1}) = y_t - \mathsf{E}(Z_t\alpha_t + \epsilon_t|Y_{t-1}) = y_t - Z_t a_t$ which can be seen as the one-step ahead forecast error of y_t given Y_{t-1} and where $a_t = \mathsf{E}(\alpha_t|Y_{t-1})$ the one-step ahead forecast of the state at time t and its variance $F_t = \text{var}(v_t|Y_t - 1)$. Note that when Y_{t-1} and v_t are fixed then Y_t is fixed and vice versa which implies that $\mathsf{E}(\alpha_t|Y_t) = \mathsf{E}(\alpha_t|Y_{t-1}, v_t)$. The derivation of the Kalman filter recursion is then obtained inductively by first studying the distibution of v_t, noting that $\text{cov}(y_j, v_t) = 0$ for any $j = 1, \ldots, t-1$

then applying *Lemma 1* to the conditional joint distribution of α_t and v_t given Y_{t-1} to obtain recursions for $a_{t|t}$ and $P_{t|t}$. Recursions for a_{t+1} and $P_{t+1} = \text{var}(\alpha_{t+1}|Y_t)$ are simply obtained by applying the conditional expectation to the state updating equation (3). The resulting system of equations forms the Kalman filter (forward) recursion and is collected in (15). Readers interested in the detail of the derivation are referred to Durbin and Koopman (2012, chap. 4).

B The nearest covariance matrix to a given symmetric matrix

Jungbacker and Koopman (2007) proved that matrix C_t defined in (18) is definte positive (See their technical report). However, due to rounding errors, small negative eigenvalues manifest; this will abort the entire sampling algorithm as it prohibits sampling from $\mathcal{N}(0, C_t)$. Moreover, the definition of B_t requires the computation of the square root (Cholesky) decomposition of C_t which is known to be difficult when the matrix has small eigenvalues. In such cases, the direct use of C_t is not possible (numerically), and one needs to find a relevant approximation that is the nearest symmetric positive definite matrix to C_t with not "too-small" (positive) eigenvalues.

The question of finding the nearest correlation matrix has been studied in the finance literature (Higham (2002), Li and Qi (2011), Borsdorf and Higham (2010)). This problem is very similar to the problem in hand and the solutions suggested can be easily adapted to our purpose. In `GKF`, we retained the alternating projections method described by Higham (2002) and modifided it to obtain a positive definte matrix with "large enough" eigenvalues.

The problem considered in Higham (2002) can be formulated as follows: for arbitrary symmetric matrix $A \in \mathbb{R}^{n\times n}$, find the matrix X that minimizes

$$\gamma(A) = \min\big\{||A - X|| : X \text{ is symmetric positive definite with unit diagonal}\big\} \tag{28}$$

the norm considered in `GKF` is the weighted Frobenuis norm [6] and is defined as

$$||A||_W = ||W^{1/2}AW^{1/2}||_F,$$

where W is a symmetric matrix of positive weights. The use of weights allows us to express our confidence in the different elements of A. Package `GKF` only allows diagonal W (the most natural choice) and the default for W is $W = I$ which corresponds to the Forbenius norm.

Let S be the set of symmetric positive definite matrices and U the set of symmetric matrices with unit diagonal. The solution of problem (28) is in the intersection of S and U and is the closest to A (in the weighted Frobeinus norm). The idea of the alternating projections method is to construct the solution iteratively by projecting the iterated guess on S first and then on U. Given that boths sets are closed and convex, so is their intersection. It thus follows from standard results in approximation theory that the minimum of (28) is achieved at a unique matrix X. The algorithm steps are summarized below

```
• $\Delta S_0 = 0,\ Y_0 = A$
• for $k = 1, 2, \ldots$
    – $R_k = Y_{k-1} - \Delta S_{k-1}$ // $\Delta S_{k-1}$ is the Dykstra's correction
    – $X_k = P_S(R_k)$
    – $\Delta S_k = X_k - R_k$
    – $Y_k = P_U(X_k)$
• end
```

The convergence of X_k and Y_K to the desired correlation matrix as $k \to \infty$ is achieved at linear rate at best (Boyle and Dykstra, 1986, see) for more details about the correction and the convergence rate).

[6]The Frobenius norm of matrix A is given by,$||A||_F^2 = \sum_{i,j} a_{ij}^2$.

Given that we restrict our attention to diagonal weight matrices W and that we expect to have some small eigenvalues, we can construct the P_S projection in an efficient way following Higham (2002, Theorem 3.2-Corollary 3.5):

$$\begin{aligned} P_S(R_k) &= W^{-1/2}\left((W^{1/2}R_kW^{1/2})_+\right)W^{-1/2}, \\ (W^{1/2}R_kW^{1/2})_+ &= \sum_{\lambda_i>\epsilon} \lambda_i q_i q_i' \end{aligned} \tag{29}$$

where λ_i are the "not too small" eigenvalues of $W^{1/2}R_kW^{1/2}$. The "not too small" concept is quantified by a threshold $\epsilon = \max_{i=1,\dots,n}(\lambda_i)eig\_tol$ where `eig_tol` is a sensibility selected by the users (default = `1e-6`). In other words, the P_S projection (when W is diagonal) is equivalent to rebuilding $W^{1/2}R_kW^{1/2}$ from its "not too small" eigenvalues and associated eigenvectors and then adjust it by inverse squared weights. The projection S_U when W is diagonal resumes to simply setting the diagonal elements of $P_S(R_k)$ to 1.

When one is only interested in the nearest covariance matrix with "not too small" eigenvalues (hence invertible), the projection S_U can be escaped. The resulting matrix of the Higham's iterations is then rebuilt from its set of selected "not too small" eigenvalues and associated eigenvectors (although the threshold is now constructed based on `posd_tol`, default=`1e-6`). Note that a similar solution was proposed in `R` by the authors of package `Matrix` (Maechler and Bates (2006), function `nearPD`). However, the `GKF` function `nearPD` is faster, closer to the Higham's algorithm and provides an approximate square-root (Cholesky) decomposition of the solution (that will be used in the computation of B_t when drawing the posterior simulated sample, see Section 7.2).

C An efficient way to compute the signal log-likelihood

We start by recalling the definition of the signal covariance matrix Ψ and suggest a recursive efficient way to compute its inverse and the associated Gaussian log-likelihood.

The all times signal vector θ was defined in Eq (7). With slightly adapted notations we define the up-to-time t signal vector as

$$\Theta_t = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_t \end{bmatrix} \quad \Theta_t \sim \mathcal{N}(\mu_t, \Psi_t) \tag{30}$$

where $\mu_t = (\nu_1', \nu_2', \dots, \nu_t')'$ and $\nu_i = c_i + Z_i\alpha_i, i = 1, \dots, t$. the up-to-time t signal covarianvce matrix is similarly given by $\Psi_t = \mathcal{Z}_t \Omega_t \mathcal{Z}_t'$ where $\mathcal{Z}_t = \text{diag}(Z_1, Z_2, \dots, Z_t)$, $\Omega_t = L_t\mathcal{D}_tL_t'$, $\mathcal{D}_t = \text{diag}(P_1, Q_2, \dots, Q_{t-1})$ and L_t is the up-to-time t part of matrix T defined in (5). An interesting property in matrix L_t can be visualized below

$$L_t = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 \\ T_1 & I & 0 & 0 & 0 & 0 & 0 \\ T_2T_1 & T_2 & I & 0 & 0 & 0 & 0 \\ T_3T_2T_1 & T_3T_2 & T_3 & I & 0 & 0 & 0 \\ & & & & \ddots & & \vdots \\ T_{t-2}\times T_1 & T_{t-2}\times T_2 & T_{t-2}\times T_3 & T_{t-2}\times T_4 & T_{t-2} & I & 0 \\ T_{t-1}\times T_1 & T_{t-1}\times T_2 & T_{t-1}\times T_3 & T_{t-1}\times T_4 & T_{t-1}T_{t-2} & T_{t-1} & I \end{bmatrix} = \begin{bmatrix} L_{t-1} & 0 \\ l_t' & I \end{bmatrix} \tag{31}$$

Note that l_t can be easily derived from l_{t-1} using the following relationship

$$l_t' = [T_{t-1}l_{t-1}', T_{t-1}]$$

Similar structure can also be derived for $\mathcal{D}_t$

$$\mathcal{D}_t = \begin{bmatrix} \mathcal{D}_{t-1} & 0 \\ 0 & d_t \end{bmatrix},$$

where $d_t = Q_{t-1}$ which is known or partially depending on the vector of parameter ψ. Note that 0 denotes here a matrix of zeros of appropriate dimension. This lower nested structure of matrix Ω_t will be used to compute its inverse recursively in an efficient way as well as in the computation of the signal log-likelihood.

C.1 Recursive computation of the observation covariance matrix inverse

As seen before, Ω_t is defined by $\Omega_t = L_t \mathcal{D}_t L_t'$. Hence, its inverse also has a similar structure given by $\Omega_t^{-1} = L_t^{\ddagger} \mathcal{D}_t^{-1} L_t^{-1}$ where $L_t^{\ddagger}$ is the inverse transpose of L_t. Using the fact that L_t^{-1} is also lower triangular and writing the definition $L_t^{-1} L_t = I$ leads to the following recursion for L_t^{-1}

$$L_{t+1}^{-1} = \begin{bmatrix} L_t^{-1} & 0 \\ -\tilde{l}_{t+1}' & I \end{bmatrix}$$

where

$$\begin{aligned} \tilde{l}_{t+1}' &= l_{t+1}' L_t^{-1} \\ &= [T_t l_t', T_t] \begin{bmatrix} L_{t-1}^{-1} & 0 \\ -\tilde{l}_t' & I \end{bmatrix} \\ &= [T_t l_t' L_{t-1}^{-1} - T_t \tilde{l}_t', T_t] \\ \tilde{l}_{t+1}' &= [0, T_t] \end{aligned} \tag{32}$$

A similar recursion is also available to compute $\mathcal{D}_t^{-1}$ which is simply given by

$$\mathcal{D}_{t+1}^{-1} = \begin{bmatrix} \mathcal{D}_t^{-1} & 0 \\ 0 & d_{t+1}^{-1} \end{bmatrix}$$

We conclude that in order to compute the up-to-time $t+1$ inverse of matrix Ω, assuming that the previous time step inverse is available, one has to just inverse the $m \times m$ (d_{t+1}) and make few matrix multiplications rather than inversing a $(t+1)m \times (t+1)m$ square matrix which is computationally demanding and can be fraught with numerical instabilities.

C.2 Recursive computation of the signal log-likelihood

The up-to-time $t+1$ signal vector Θ_{t+1} can be constructed from the time t step by $\Theta_{t+1} = [\Theta_t', \theta_{t+1}']'$. Hence, in thess notations, the all time signal vector is given by $\theta = \Theta_n$ which has a Gaussian destribution $\mathcal{N}(\mu_n, \Psi_n)$. The associated log-likelihood is given by (up to a constant and sign change)

$$l(\Theta_n) \quad \propto \quad \underbrace{\log |\Psi_n|}_{ld_n} + \underbrace{(\Theta_n - \nu_n)' \Psi_n^{-1} (\Theta_n - \nu_n)}_{\mathcal{Q}_n} \tag{33}$$

Computation of ld_n

Using the definition of Ψ_n in (7) and the fact that $\mathcal{Z}_n$ is block diagonal and L_n is unit block lower triangular matrices, the computation of the log-determinant of Ψ_n can be done in the following way

$$\begin{aligned} ld_n &= \log \Big(|\mathcal{Z}_n| |\Omega_n| |\mathcal{Z}_n'| \Big) \\ &= 2 \log |\mathcal{Z}_n| + \log |\Omega_n| \\ &= \sum_{t=1}^{n} \Big\{ 2 \log |Z_t| + \log |\mathcal{D}_t| \Big\} \end{aligned} \tag{34}$$

which suggest the following recursive algorithm

- For $t = 1$ initialize ld with $ld_1 = 2 \log |Z_1| + \log |P_1|$
- for $t = 2, \ldots, n$ update ld with $ld_t = ld_{t-1} + 2 \log |Z_t| + \log |Q_{t-1}|$

Computation of $\mathcal{Q}_n$

The computation of the quadratic form $\mathcal{Q}_n$ is more challenging since it involves the inversion of a large covarinace matrix. A naive approch would achieve a brute force inversion of Ψ_n which involves order n^3 flops and can be fraught with numerical instabilities, wherein small eigenvalues could manifest, due to rounding errors, as small negative numbers; this will abort the entire model-fiting routime. The approach adopted here only requires order n^2 flops and is numerically more stable.

First, let's re-write the quadratic form $\mathcal{Q}_n$ in a more convinient way

$$\begin{aligned}\mathcal{Q}_n &= (\Theta_n - \nu_n)'\Psi_n^{-1}(\Theta_n - \nu_n) \\ &= \underbrace{(\Theta_n - \nu_n)'\mathcal{Z}_n^{\ddagger}}_{\tilde{X}_n'}\underbrace{L_n^{\ddagger}\mathcal{D}_n^{-1}L_n^{-1}}_{\Omega_n^{-1}}\underbrace{\mathcal{Z}_n^{-1}(\Theta_n - \nu_n)}_{\tilde{X}_n}\end{aligned} \tag{35}$$

Note that when Z_t is not square, Z_t^{-1} is computed by the Moore-Penrose pseudo-inverse.
Define now the square root of a matrix A as a matrix $B = A^{1/2}$ such that $A = BB'$. The quadratic form (35) can be re-written now as

$$\begin{aligned}\mathcal{Q}_n &= \tilde{X}_n'L_n^{\ddagger}\mathcal{D}_n^{-\ddagger/2}\underbrace{\mathcal{D}_n^{-1/2}L_n^{-1}\tilde{X}_n}_{\mathcal{V}_n} \\ &= \mathcal{V}_n'\mathcal{V}_n \\ &= \sum_{t=1}^{n} v_t'v_t\end{aligned} \tag{36}$$

where $v_t = D_t^{-1/2}T_t^{-1}\tilde{X}_t, t = 1, \dots, n$ is the model residual. The idea of our alternative approach is then to compute $\mathcal{Q}_n$ recurslively by summing at each time step t the squared residuals $v_t'v_t$.

Now, we will focus on how to update the residuals. Assuming v_t is known, we have

$$\begin{aligned}v_{t+1} &= D_{t+1}^{-1/2}T_{t+1}^{-1}\tilde{X}_{t+1} \\ &= \begin{bmatrix} D_t^{-1/2} & 0 \\ 0 & d_{t+1}^{-1/2}\end{bmatrix}\begin{bmatrix} T_t^{-1} & 0 \\ -\tilde{l}_{t+1}' & I\end{bmatrix}\begin{bmatrix}\tilde{X}_t \\ \tilde{x}_{t+1}\end{bmatrix} \\ &= \begin{bmatrix} D_t^{-1/2}T_t^{-1} & 0 \\ -d_{t+1}^{-1/2}\tilde{l}_{t+1}' & d_{t+1}^{-1/2}\end{bmatrix}\begin{bmatrix}\tilde{X}_t \\ \tilde{x}_{t+1}\end{bmatrix} \\ &= \begin{bmatrix}\tilde{v}_t \\ d_{t+1}^{-1/2}\left(\tilde{x}_{t+1} - \tilde{l}_{t+1}'\tilde{X}_t\right)\end{bmatrix} \\ &= \begin{bmatrix}\tilde{v}_t \\ d_{t+1}^{-1/2}\left(\tilde{x}_{t+1} - T_t\tilde{x}_t\right)\end{bmatrix}\end{aligned} \tag{37}$$

Hence, the recursive computation of $\mathcal{Q}_n$ can be summarized in the following algorithm

- For $t = 1$, initialize the procedure by
  - $\nu_1 = c_1 + Z_1 d_1$
  - $\tilde{x}_1 = Z_1^{-1}(\theta_1 - \nu_1)$
  - $Q_1 = \tilde{x}_1' L_1^{\ddagger} P_1^{-1} L_1^{-1}\tilde{x}_1$
- for $t = 2, \dots, n$, update $\mathcal{Q}_{\sqcup}$ by
  - $\nu_t = c_t + Z_t d_t$
  - $\tilde{x}_t = Z_t^{-1}(\theta_t - \nu_t)$

– $Q_t = Q_{t-1} + \Big(\tilde{x}_t - T_{t-1}\tilde{x}_{t-1}\Big)' d_t^{-1}\Big(\tilde{x}_t - T_{t-1}\tilde{x}_{t-1}\Big)$

Here again, at each time step, we only require the inversion of two relatively small matrices (Z_t and d_t).

The log-likelihood of the signal vector is simply obtained by summing the results of the two algorithms described above. This method was implemented in function `FKF` where the signal log-likelihood is returned if `ComputeThetaLik` is turned to `TRUE` (default `FALSE`) and the signal vector (where the likelihood should be computed) is provided (`ThetaVal`). See the package documentation for more details about the inputs required classes.

D Bias corection in the log-likelihood evaluation

The likelihood (22) is more easily computed using the logarithm of the importance weights q. In fact, the different densities involved in the computation of q are more manageable if the logarithm is applied to them before combining them together (which will imply summation instead of multiplication). Besides, it is usally not feasible to retrieve the importance weights q from its logarithm counterparts (applying the exponentiel) for numerical reasons. Therefore, it will be useful to estimate the likelihood $l(\psi) = \log L(\psi) = \log \mathsf{E}\Big(q(\theta)\Big)_p$ using the $\log\Big(q(\theta^i)\Big), i = 1, \ldots, M$; a natural candidate estimator would be $\frac{1}{M}\sum_{i=1}^{M}\log(q(\theta^i))$. However, $\mathsf{E}\Big(\frac{1}{M}\sum_{i=1}^{M}\log(q(\theta^i))\Big) = \mathsf{E}\Big(\log(q(\theta))\Big) \neq \log \mathsf{E}\Big(q(\theta)\Big)$, so a small bias will be introduced. In order to correct this bias, one needs to study the behavior of $\log \mathsf{E}\Big(q(\theta)\Big)$ in the neighberhood of $\mathsf{E}\Big(\log q(\theta)\Big)$. Define the following useful quantities

$$\begin{aligned} X &= \log(q(\theta)) \\ \mu_X &= \mathsf{E}\Big(X\Big) \approx \tfrac{1}{M}\sum_{i=1}^{M}\log(q(\theta^i)) \\ \sigma_X^2 &= \mathrm{var}(X) \approx \tfrac{1}{M-1}\sum_{i=1}^{M}(\log(q(\theta^i)) - \mu_X)^2 \end{aligned} \tag{38}$$

where the $\approx$ sign means estimated by. The Taylor expansion of e^X in the neighberhood of μ_X is given by

$$e^X = e^{\mu_X}\left(1 + (X-\mu_X) + \frac{1}{2}(X-\mu_X)^2\right) + \mathcal{O}(N^{-\frac{3}{2}})$$

so,

$$\begin{aligned} \mathsf{E}\Big(e^X\Big) &= e^{\mu_X}\Big(1 + \underbrace{\mathsf{E}(X-\mu_X)}_{0} + \tfrac{1}{2}\,\mathsf{E}\Big((X-\mu_X)^2\Big)\Big) && +\mathcal{O}(N^{-\frac{3}{2}}) \\ &= e^{\mu_X}(1 + \tfrac{1}{2}\sigma_X^2) && +\mathcal{O}(N^{-\frac{3}{2}}) \end{aligned} \tag{39}$$

We conclude that the log-likelihood $\log l(\psi)$ can be approximated by

$$\begin{aligned} \log \mathsf{E}\Big(q(\theta)\Big) &\approx \mu_X + \underbrace{\log(1 + \frac{\sigma_X^2}{2})}_{\text{bias correction}} \\ &\approx \underbrace{\frac{1}{M}\sum_{i=1}^{M}\log(q(\theta^i))}_{\text{biased estimator}} + \log(1 + \tfrac{s_X^2}{2M}) \end{aligned} \tag{40}$$

where s_X^2 is the sample covariance estimator of σ_X^2 defined in (38).