KFAS: Exponential family state space models in R

Jouni Helske
University of Jyväskylä

Abstract

State space modelling is an efficient and flexible method for statistical inference of broad class of time series and other data. This paper describes an R package **KFAS** for modelling state space models with the observations from exponential family, namely Gaussian, Poisson, binomial, negative binomial and gamma distributions. After introducing the basic theory behind the state space modelling, an illustrative example is provided, and finally a short comparison to alternative modelling framework is presented.

Keywords: state space models, time series, dynamic linear models, **KFAS**, R, exponential family.

Modification of manuscript submitted to *Journal of Statistical Software*

1. Introduction

State space models offer an unified framework for modelling several types of time series and other data. Structural time series, ARIMA models, simple regression, generalized linear mixed models, and cubic spline smoothing are just some examples of the statistical models which can be represented as a state space model. One of the simplest classes of state space models are linear Gaussian state space models (also known as dynamic linear models), which are analytically tractable, and are therefore often used in many fields of science.

Petris and Petrone (2011) and Tusell (2011) introduce and review some of the contributed packages available at Comprehensive R Archive Network (CRAN) for R (R Core Team 2014) for Gaussian state space modelling. Since then, several new additions have emerged in CRAN. Most of these packages use one or multiple packages reviewed in Tusell (2011) for filtering and smoothing, and add new user interface and functionality for certain type of models. For example, package **rucm** (Chowdhury 2014) is focused on structural time series, while **dlmodeler** (Szymanski 2014) provides unified interface compatible with multiple packages, and **MARSS** (Holmes, Ward, and Wills 2013, 2012) provides functions for a maximum likelihood estimation of large class of Gaussian state space models via the EM-algorithm.

One of the packages reviewed in the aforementioned papers is **KFAS** (Kalman Filtering And Smoothing) which, in addition, of modelling the general linear Gaussian state space models, it can also be used in cases where the observations are from other exponential family models, namely binomial, Poisson, negative binomial, and Gamma models. To my knowledge, **KFAS** is currently the only package which supports non-Gaussian models in a classical state space modelling framework. Package **sspir** (Dethlefsen, Lundbye-Christensen, and Christensen 2012) included similar capabilities, but it was removed from CRAN in 2013. In

addition to **KFAS**, there are at least two packages which are designed for more broader class of problems, but can also be used for exponential family state space modelling. Package **pomp** (King, Ionides, Bretó, Ellner, Ferrari, Kendall, Lavine, Nguyen, Reuman, Wearing, and Wood 2014) offers functions for inference of state space models with non-Gaussian and non-linear observation and state equations by particle filtering and related methods. Another package suitable for state space modelling is **INLA** (Rue, Martino, Lindgren, Simpson, Riebler, and Krainski 2015) (not available on CRAN), which can be used for Bayesian analysis via integrated nested Laplace approximation technique. Although it is often used in a spatial modelling via Gaussian random fields, it can also be used for certain temporal state space models where the state transitions are Gaussian.

After the papers by Petris and Petrone (2011) and Tusell (2011), **KFAS** has been completely rewritten. Package is now much more user friendly due to the use of R's symbolic formulas in model definition. The non-Gaussian modelling, which was somewhat experimental in old versions of **KFAS**, is now fully functional supporting multivariate models with different distributions. Many other features have also been added (such as methods for computing model residuals), performance of the main functions have improved and in the process several bugs have been also fixed.

In this paper I first introduce the basic theory related to state space modelling, and then proceed to show main aspects of **KFAS** in more detail, illustrate its functionality by applying it to real life dataset ~~and finally make short comparison between **INLA** and **KFAS**~~ (comparison to **INLA** is removed from vignette as **INLA** is not available at CRAN).

2. Gaussian state space model

The theory behind **KFAS** is mostly based on Durbin and Koopman (2012) and related articles by the same authors, and therefore the basic notation is nearly identical with the one used by Durbin and Koopman. For linear Gaussian state space model with continuous states and discrete time intervals $t = 1, \ldots, n$ we have

$$\begin{aligned} y_t &= Z_t\alpha_t + \epsilon_t, \quad \text{(observation equation)} \\ \alpha_{t+1} &= T_t\alpha_t + R_t\eta_t, \quad \text{(state equation)} \end{aligned} \tag{1}$$

where $\epsilon_t \sim N(0, H_t), \eta_t \sim N(0, Q_t)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other. We assume that y_t is $p \times 1$, α_{t+1} is $m \times 1$ and η_t is $k \times 1$ vector. We also denote $\alpha = (\alpha_1^\top, \ldots, \alpha_n^\top)^\top$ and similarly $y = (y_1^\top, \ldots, y_n^\top)^\top$.

Here y_t contains the observations at time t, whereas α_t is a vector of latent state process at time point t. The system matrices Z_t, T_t, and R_t, together with the covariance matrices H_t and Q_t depend on the particular model definition, and are often time invariant, i.e., do not depend on t. Usually at least some of these matrices contain unknown parameters which need to be estimated. In **KFAS** one defines the model with the function `SSModel`. Function `SSModel` only builds the model, and does not perform estimation of unknown parameters, which differs from functions like `lm` which builds and estimates the model with one command.

The main goal of the state space modelling is to gain knowledge of the latent states α given the observations y. This is achieved by using two important recursive algorithms, Kalman filtering and smoothing. From Kalman filtering algorithm we obtain the one step ahead predictions and the prediction errors

$$a_{t+1} = \mathsf{E}(\alpha_{t+1}|y_t, \ldots, y_1),$$
$$v_t = y_t - Z_t a_t,$$

and the related covariance matrices

$$P_{t+1} = \mathsf{VAR}(\alpha_{t+1}|y_t, \ldots, y_1),$$
$$F_t = \mathsf{VAR}(v_t) = Z_t P_t Z_t^\top + H_t.$$

Using the results of the Kalman filtering, we establish the state smoothing equations running backwards in time and yielding

$$\hat{\alpha}_t = \mathsf{E}(\alpha_t|y_n, \ldots, y_1),$$
$$V_t = \mathsf{VAR}(\alpha_t|y_n, \ldots, y_1).$$

Similar smoothed estimates can also be computed for the disturbance terms ϵ_t and η_t, and straightforwardly for the mean signal $\theta_t = Z_t\alpha_t$. For details on these algoritms, see Appendix A and Durbin and Koopman (2012).

A prior distribution of the initial state vector α_1 can be defined as a multivariate Gaussian distribution with mean a_1 and covariance matrix P_1. For uninformative diffuse prior, one typically sets $P_1 = \kappa\mathrm{I}$, where κ is for example 10^7. However, this method can be numerically unstable due to cumulative roundoff errors. To solve this issue Koopman and Durbin (2003) present the exact diffuse initialization method, where the diffuse elements in a_1 are set to zero and P_1 is decomposed as $\kappa P_{\infty,1} + P_{*,1}$, where $\kappa \to \infty$. Here $P_{\infty,1}$ is a diagonal matrix with ones on those diagonal elements which relate to the nonstationary elements of α_1, and $P_{*,1}$ contains the covariances of the stationary elements of α_1 (and zeros elsewhere). At the start of the Kalman filtering (and at the end of backward smoothing) we use so called exact diffuse initialisation formulas until $P_{\infty,t}$ becomes zero matrix, and then continue with usual Kalman filtering equations. This exact method should be less prone to numerical errors, although they can still occur especially in the smoothing phase, if we have for example high collinearity between the explanatory variables of the model. Note that given all the parameters in the system matrices, results from the Kalman filter and smoother are equivalent with Bayesian analysis given the same prior distribution for α_1.

When we have multivariate observations, it is possible that in the diffuse phase, the matrix F_t is not invertible, and the computation of a_{t+1} and P_{t+1} becomes impossible. On the other hand, even if F_t is invertible, the computations can become slow when dimensionality of F_t, i.e., the number of series increases. Also in case of multivariate observations, the formulas relating to the diffuse initialization become cumbersome. Based on the ideas of Anderson and Moore (1979), a complete univariate approach for filtering and smoothing was introduced by Koopman and Durbin (2000) (known as sequential processing by Anderson and Moore). The univariate approach is based on the alternative representation of the model (1), namely

$$\begin{aligned} y_{t,i} &= Z_{t,i}\alpha_{t,i} + \epsilon_{t,i}, \quad i = 1, \ldots, p_t, \quad t = 1, \ldots, n, \\ \alpha_{t,i+1} &= \alpha_{t,i}, \quad i = 1, \ldots, p_t - 1, \\ \alpha_{t+1,1} &= T_t\alpha_{t,p_t} + R_t\eta_t, \quad t = 1, \ldots, n, \end{aligned}$$

and $a_{1,1} \sim N(a_1, P_1)$, with an assumption that H_t is diagonal for all t. Here the dimension of the observation vector y_t can vary over time and therefore missing observations are handled

straightforwardly by adjusting the dimensionality of y_t. In case of non-diagonal H_t, the original model can be transformed either by taking the LDL decomposition of H_t, and multiplying the observation equation with the L_t^{-1}, so $\epsilon_t^* \sim N(0, D_t)$, or by augmenting the state vector with ϵ, when Q_t becomes block diagonal with blocks Q_t and H_t. The augmenting can also be used for introducing correlation between ϵ and η. Both LDL decomposition and state vector augmentation is supported in **KFAS**.

In theory when using the univariate approach, the computational costs of filtering and smoothing decrease, as the number of matrix multiplications decrease, and there is no need for solving the system of equations (Durbin and Koopman 2012, p. 159). As noted in (Tusell 2011), these gains can somewhat cancel out as more calls to linear algebra functions are needed and the memory management might not be as effective as working with larger objects at once. Nevertheless as noted previously, the sequential processing has also other clear benefits especially with diffuse initialization where the univariate approach simplifies the recursions considerably (Durbin and Koopman 2012).

KFAS uses this univariate approach in all cases. Although $K_t = P_t Z_t^\top = \mathsf{COV}(a_t, y_t | y_t - 1, \ldots, y_1)$, v_t, and F_t differ from the standard multivariate versions, we get $a_t = a_{t,1}$ and $P_t = P_{t,1}$ by using the univariate approach. If standard multivariate matrices F_t and K_t are needed for inference, they can be computed later from the results of the univariate filter. As $F_{*,i,t}$, $K_{*,i,t}$, and $P_{*,t}$ coincide with the nondiffuse counterparts if $F_{\infty,i,t} = 0$, the asterisk is dropped from the variable names in **KFAS**, and for example variable `F` is a $n \times p$ array containing $F_{*,i,t}$ and $F_{i,t}$, whereas `Finf` is a $n \times d$, where d is the last time point before the diffuse phase ended.

2.1. Log-likelihood of the Gaussian state space model

The Kalman filter equations can be used for computing the log-likelihood, which in its standard form is

$$\log L = \frac{np}{2} \log 2\pi - \frac{1}{2} \sum_{t=1}^{n} (\log |F_t| + v_t' F_t^{-1} v_t).$$

In case of the univariate treatment and diffuse initialization, the diffuse log-likelihood can be written as

$$\log L_d = -\frac{1}{2} \sum_{t=1}^{n} \sum_{i=1}^{p_t} w_{i,t},$$

where

$$w_{i,t} = \begin{cases} \log F_{\infty,i,t}, & \text{if } F_{\infty,i,t} > 0, \\ I(F_{i,t} > 0)(\log 2\pi + \log F_{i,t} + v_{i,t}^2 F_{i,t}^{-1}), & \text{if } F_{\infty,i,t} = 0. \end{cases}$$

See Durbin and Koopman (2012, Chapter 7) for details. Francke, Koopman, and De Vos (2010) show that there are cases where the above definition of diffuse log-likelihood is not optimal. Without going into the details, if system matrices Z_t or T_t contain unknown parameters in their diffuse parts, the diffuse likelihood is missing one term which depends on those unknown parameters. Francke *et al.* (2010, p.411–412) present a recursive formula for computing this extra term, which is also supported by **KFAS**.

3. State space models for exponential family

KFAS can also deal with observations which come from distributions of exponential family class other than Gaussian. We assume that the state equation as is in the Gaussian case, but the observation equation has the form

$$p(y_t|\theta_t) = p(y_t|Z_t\alpha_t),$$

where $\theta_t = Z_t\alpha_t$ is the signal and $p(y_t|\theta_t)$ is the observational density.

The signal θ_t is the linear predictor which is connected to the expected value $E(y_t) = \mu_t$ via a link function $l(\mu_t) = \theta_t$. In **KFAS**, the following distributions and links are available:

1. Gaussian distribution with mean μ_t and variance u_t with identity link $\theta_t = \mu_t$.
2. Poisson distribution with intensity λ_t and exposure u_t together with log-link $\theta_t = \log(\lambda_t)$. Thus we have $\mathsf{E}(y_t|\theta_t) = \mathsf{VAR}(y_t|\theta_t) = u_t e^{\theta_t}$.
3. Binomial distribution with size u_t and probability of success π_t. **KFAS** uses logit-link so $\theta_t = \text{logit}(\pi_t)$ resulting $\mathsf{E}(y_t|\theta_t) = u_t\pi_t$ and $\mathsf{VAR}(y_t|\theta_t) = u_t(\pi_t(1-\pi_t))$.
4. Gamma distribution with a shape parameter u_t and an expected value μ_t, again with log-link $\theta_t = \log(\mu)$, where Gamma distribution is defined as

$$p(y_t|\mu_t, u_t) = \frac{u_t^{u_t}}{\Gamma(u_t)}\mu_t^{-u_t}y_t^{u_t-1}e^{\frac{y_t u_t}{\mu_t}}.$$

   This gives us $\mathsf{E}(y_t|\theta_t) = e^{\theta_t}$ and $\mathsf{VAR}(y_t|\theta_t) = e^{2\theta_t}/u_t$.

5. Negative binomial distribution with a dispersion parameter u_t and an expected value μ_t with log-link $\theta_t = \log(\mu_t)$, where the negative binomial distribution is defined as

$$p(y_t|\mu_t, u_t) = \frac{\Gamma(y_t + u_t)}{\Gamma(u_t)y_t!}\frac{\mu_t^{y_t}u_t^{u_t}}{(\mu_t + u_t)^{u_t+y_t}},$$

   giving us $\mathsf{E}(y_t|\theta_t) = e^{\theta_t}$ and $\mathsf{VAR}(y_t|\theta_t) = e^{\theta_t} + e^{2\theta_t}/u_t$.

Note that variable u_t has a different meaning depending on the distribution it is linked to. In **KFAS** one defines the distribution for each time series via argument `distribution` and the additional known parameters u_t corresponding to each series as columns of matrix `u`.

In order to make inferences of the non-Gaussian models, we first find a Gaussian model which has the same conditional posterior mode as $p(\theta|y)$ (Durbin and Koopman 2000). This is done using an iterative process with Laplace approximation of $p(\theta|y)$, where the updated estimates for θ_t are computed via Kalman filtering and smoothing from the approximating Gaussian model. In approximating Gaussian model the observation equation is replaced by

$$\tilde{y}_t = Z_t\alpha_t + \epsilon_t, \quad \epsilon_t \sim N(0, H_t)$$

where the pseudo-observations $\tilde{y}_t$ variances H_t are based on first and second derivatives of $logp(y_t|\theta_t)$ with respect to θ_t (Durbin and Koopman 2000).

Final estimates $\hat{\theta}_t$ correspond to the mode of $p(\theta|y)$. In Gaussian case the mode is also the mean. In cases listed in (1)-(5) the difference between the mode and mean is often negligible.

Nevertheless, we are usually more interested in μ_t than in the linear predictor θ_t. As the link function is non-linear, direct transformation $\hat{\mu}_t = l^{-1}(\hat{\theta}_t)$ introduces some bias. To solve this problem **KFAS** also contains methods based on importance sampling, which allows us to correct these possible approximation errors. With importance sampling technique we can also compute the log-likelihood and the smoothed estimates for $f(\alpha)$, where f is an arbitrary function of states, $\exp(Z_t\alpha_t)$ being typical example.

In importance sampling scheme, we first find the approximating Gaussian model, simulate the states α^i from this Gaussian model and then compute the corresponding weights $w_i = p(y|\alpha^i)/g(y|\alpha^i)$, where $p(y|\alpha^i)$ represents the conditional non-Gaussian density of the original observations, and $g(y|\alpha^i)$ is the conditional Gaussian density of the pseudo-observations $\tilde{y}$. These weights are then used for computing

$$\mathsf{E}(f(\alpha)|y) = \frac{\sum_{i=1}^N f(\alpha^i)w_i}{\sum_{i=1}^N w_i}.$$

The simulation of Gaussian state space models in **KFAS** is based on simulation smoothing algorithm by Durbin and Koopman (2002). In order to improve the simulation efficiency, **KFAS** can use two antithetic variables in the simulation algorithms. See Durbin and Koopman (2012, p. 265-266) for details how these are constructed.

KFAS also provides means for filtering of non-Gaussian models. This is achieved by sequentially using the smoothing scheme for $(y_1, \ldots, y_t), t = 1 \ldots, n$ with y_t set as missing. This is relatively slow procedure for large models, as the importance sampling algorithms need to be performed n times, although the first steps are much faster than the one using whole data. The non-Gaussian filtering is mainly for computation of recursive residuals (see Section 4) and for illustrative purposes, where the computational efficiency is not that important. With large models or online-filtering problems, one is recommended to use proper particle filter approach which is out of the scope of this paper.

3.1. Log-likelihood of the non-Gaussian state space model

The log-likelihood function for the non-Gaussian model can be written as (Durbin and Koopman 2012, p. 272)

$$\begin{aligned}\log L(y) &= \log \int p(\alpha, y)\mathrm{d}\alpha \\ &= \log L_g(y) + \log E_g\left[\frac{p(y|\theta)}{g(y|\theta)}\right],\end{aligned}$$

where $L_g(y)$ is the log-likelihood of the Gaussian approximating model and the expectation is taken with respect to the Gaussian density $g(\alpha|y)$. The expectation can be approximated by

$$\log E_g\left[\frac{p(y|\theta)}{g(y|\theta)}\right] \approx \log \frac{1}{N}\sum_{i=1}^N w_i. \tag{2}$$

In many cases, good approximation of the log-likelihood can be computed without any simulation, by setting $N = 0$ and using the mode estimate $\hat{\theta}$ from the approximating model.

In practice (2) suffer from the fact that $w_i = p(y|\theta^i)/g(y|\theta^i)$ is numerically unstable; when number of observations is large, the discrete probability mass function $p(y|\theta^i)$ tends to zero,

even when the Gaussian density function $g(y|\alpha^i)$ does not. Therefore it is better to redefine the weights as

$$w_i^* = \frac{p(y|\theta^i)/p(y|\hat{\theta})}{g(y|\theta^i)/g(y|\hat{\theta})}.$$

The log-likelihood is then computed as

$$\log \hat{L}(y) = \log L_g(y) + \log \hat{w} + \log \frac{1}{N} \sum_{i=1}^{N} w_i^*,$$

where $\hat{w} = p(y|\hat{\theta})/g(y|\hat{\theta})$.

4. Residuals

For exponential family state space models, multiple types of residuals can be computed. Probably the most useful ones are standardized recursive residuals, which are based on the one-step ahead predictions from the Kalman filter. For univariate case these are defined as

$$\frac{y_t - \mathsf{E}(y_t|y_{t-1}, \ldots, y_1)}{\sqrt{\mathsf{VAR}(y_t|y_{t-1}, \ldots, y_1)}}, \quad t = d+1 \ldots, n,$$

where d is the last time point of diffuse phase, and the denominator can be decomposed as

$$\begin{aligned}\mathsf{VAR}(y_t|y_{t-1}, \ldots, y_1) &= \mathsf{VAR}(\mathsf{E}(y_t|\theta_t, y_{t-1}, \ldots, y_1)|y_{t-1}, \ldots, y_1) \\ &\quad + \mathsf{E}(\mathsf{VAR}(y_t|\theta_t, y_{t-1}, \ldots, y_1)|y_{t-1}, \ldots, y_1) \\ &= \mathsf{VAR}(\mathsf{E}(y_t|\theta_t)|y_{t-1}, \ldots, y_1) + \mathsf{E}(\mathsf{VAR}(y_t|\theta_t)|y_{t-1}, \ldots, y_1).\end{aligned}$$

In the Gaussian case this simplifies to $v_t F_t^{-\frac{1}{2}}$.

For multivariate observations we have several options on how to standardize the residuals. The most common one is a marginal standardization approach where each residual series is divided by its standard deviation, so we get standard normal distributed residual series which have no autocorrelation or cross-correlation with non-zero lags. Another option is to use for example Cholesky decomposition for the prediction error covariance matrix F_t and standardize the residuals by $L_t^{-1}(y_t - \hat{y}_t)$ where $L_t L_t^\top = F_t$. Now the whole residual series should look like a draw from standard normal distributed without any autocorrelation.

For computing the marginally standardized residuals, multivariate versions of F_t and v_t are needed, wheras the Cholesky standardized residuals can be computed directly from sequential Kalman filter as

$$v_{i,t} F_{i_t}^{-\frac{1}{2}}, \quad j = 1, \ldots, p, \quad t = d+1 \ldots, n.$$

These multivariate residuals depend on the ordering of the series, so if the residual diagnostics exhibit deviations from model assumptions, then the intepretation is somewhat more difficult than when using the marginal residuals. Therefore marginal residuals might be preferred. Note that if we want quadratic form residuals $(y_t - \hat{y}_t)F_t^{-1}(y_t - \hat{y}_t)$, then the ordering of the series does not matter.

The recursive residuals are defined just for the non-diffuse phase, which is problematic if the model contains long diffuse phase for example if a dummy variable with a diffuse prior is

incorporated to the model. This is because the diffuse phase cannot end before the dummy variable changes its value at least once. In order to circumvent this, one can set the a proper but highly non-informative prior distribution for the intervention variable when computing the residuals, which should have negligible effect on the visual inspection of the residual plots.

Other potentially useful residuals are auxiliary residuals which are based on smoothed values of states. For details, see Harvey and Koopman (1992) and Durbin and Koopman (2012, Chapter 7).

5. Functionality of KFAS

The state space model used with **KFAS** is built using function `SSModel`. The function uses R's formula object in a similar way as for example functions `lm` and `glm`. In order to define the different components of the state space model, auxiliary functions `SSMtrend`, `SSMseasonal`, `SSMcycle`, `SSMarima`, `SSMregression` are provided These functions can be used to define the structural, ARIMA, and regression components of the model. The function `SSMcustom` can be used for constructing an arbitrary component by directly defining the system matrices of model (1). More details on how to construct common state space models with **KFAS** are presented in Section 6.

The function `SSModel` returns an object of class `SSModel`, which contains the observations `y` as the `ts` object, system matrices `Z`,`H`,`T`,`R`,`Q` as arrays of appropriate dimensions, together with matrices `a1`, `P1`, and `P1inf` defining the initial state distribution. Additional components contains the system matrix `u` which is used in non-Gaussian models for additional parameters, character vector `distribution` which defines the distributions of the observations (multivariate series can have different distributions), and tolerance parameter `tol` which is used in diffuse phase for checking whether F_∞ is nonzero.

`SSModel` object also contains some attributes, namely integer valued attributes `p,m,k,n` which define the dimensions of the system matrices, and character vectors `state_types` and `eta_types` which define the elements of α_t and η_t. These attributes are used internally by **KFAS**, although user can carefully modify them if needed. For example, if the user wishes to redefine the error term η_t by changing the dimensions of `R` and `Q`, the attributes `k` and `eta_types` need to be updated accordingly.

The unknown model parameters can be estimated with `fitSSM`, which is a wrapper around the **R**'s `optim` function and the `logLik` method for **SSModel** object. For `fitSSM`, user gives the model object, initial values of unknown parameters and a function `updatefn` which is used to update the model given the parameters (the help page of `fitSSM` gives an example of `updatefn`). As the numerical optimization routines update the model and compute the likelihood thousands of times, the user is encouraged to build his own problem-specific model updating function for maximum efficiency. By default, `fitSSM` estimates the `NA` values in the time invariant covariance matrices H and Q, but no general estimation function is provided. Of course, user can also directly use `logLik` method for computing the likelihood and thus is free to choose a suitable optimization method for his problem.

Function `KFS` computes the filtered (one step ahead prediction) and smoothed estimates for states, signals, and the values of the inverse link function (expected value μ or probability π) in a non-Gaussian case. For Gaussian models, disturbance smoothing is also available.

With `simulateSSM` user can simulate the states, signals or disturbances of the Gaussian

state space models given the model and the observations. If the model contains missing observations, these can also be simulated by `simulateSSM` in similar way. It is also possible to simulate states from predictive distributions $p(\alpha_t|y_1, \ldots, y_{t-1})$, $t = 1, \ldots, n$. For these simulations, instead of using marginal distributions $N(a_t, P_t)$, **KFAS** uses a modification of Durbin and Koopman (2002), where smoothing is replaced by filtering.

For non-Gaussian models, `importanceSSM` returns the states or signals simulated from the approximating Gaussian model, and the corresponding weights w_i, which can then be used to compute arbitrary functions of the states or signals.

There are several `S3` methods available for `SSModel` and `KFS` objects. For both objects, simple `print` methods are provided, and for `SSModel` objects there is the `logLik` method. The `predict` method is for computation of the point predictions together with confidence or prediction intervals. Extraction operator `[` for extracting and replacing the subsets of model elements is also provided for class `SSModel`. Use of this method when modifying model is suggested instead of common list extractor `$`, as the latter can accidentally modify the dimensions of the corresponding model matrices.

For `KFS` object, the methods `residuals`, `rstandard` and `hatvalues` are provided. Also, function `signal` can be used for extracting subsets of signals from `KFS` objects, for example the part of $Z_t\alpha_t$ that corresponds to the regression part of the model.

6. Constructing common state space models with KFAS

In this section we present some typical models which can be presented in a state space form. More examples can be found on the main help page of **KFAS** by typing `?KFAS` after the package is loaded via `library(KFAS)`.

All the auxialiary functions used in formula argument of the function `SSModel` have some common arguments which are not directly related to the system matrices of the corresponding component. In complex multivariate models, important argument is `index`, which defines the series for which the corresponding component is constructed. For example, if we have four time series ($p = 4$), we may want to use certain regression component only for series 2 and 4. In this case we use argument `index=c(2,4)` when calling the appropriate `SSMregression` function. By default the index is `1:p` so the component is constructed for all series.

Another argument used in several auxiliary functions is `type`, which can take two possible values. Value `"distinct"` defines the component separately for each series defined by `index` (with covariance structure defined by argument `Q`), whereas value `"common"` constructs single component which applies to all series defined by `index`. For example we can define distinct local level components for all series together with covariance matrix which captures the dependencies of the different series, or we can define just a single local level component which is common to all series.

6.1. Structural time series

Structural time series refers to class of state space models where the observed time series is decomposed into several underlying components, such as trend and seasonal effects. The

basic structural time series model is of the form

$$\begin{aligned} y_t &= \mu_t + \gamma_t + c_t + \epsilon_t, \quad \epsilon_t \sim N(0, H_t), \\ \mu_{t+1} &= \mu_t + \nu_t + \xi_t, \quad \xi_t \sim N(0, Q_{\text{level},t}), \\ \nu_{t+1} &= \nu_t + \zeta_t, \quad \zeta_t \sim N(0, Q_{\text{slope},t}), \end{aligned} \tag{3}$$

where μ_t is the trend component, γ_t is the seasonal component and c_t is the cycle component. The seasonal component with period s can be defined in a dummy variable form

$$\gamma_{t+1} = -\sum_{j=1}^{s-1} \gamma_{t+1-j} + \omega_t, \quad \omega_t \sim N(0, Q_{\text{seasonal},t}),$$

or trigonometric form where

$$\begin{aligned} \gamma_t &= \sum_{j=1}^{\lfloor s/2 \rfloor} \gamma_{j,t}, \\ \gamma_{j,t+1} &= \gamma_{j,t} \cos \lambda_j + \gamma^*_{j,t} \sin \lambda_j + \omega_{j,t}, \\ \gamma^*_{j,t+1} &= -\gamma_{j,t} \sin \lambda_j + \gamma^*_{j,t} \cos \lambda_j + \omega^*_{j,t}, \quad j = 1, \ldots, \lfloor s/2 \rfloor, \end{aligned}$$

with $\omega_{j,t}$ and $\omega^*_{j,t}$ being independently distributed variables with $N(0, Q_{\text{seasonal},t})$ distribution and $\lambda_j = 2\pi j/s$.

Cycle component with period s is defined as

$$\begin{aligned} c_{t+1} &= c_t \cos \lambda_c + c^*_t \sin \lambda_c + \omega_t, \\ c^*_{t+1} &= -c_t \sin \lambda_c + c^*_t \cos \lambda_c + \omega^*_t, \end{aligned}$$

with ω_t and ω^*_t being independent variables from $N(0, Q_{\text{cycle},t})$ distribution and frequency $\lambda_c = 2\pi/s$.

For non-Gaussian models the observation equation of (3) is replaced by $p(y_t|\theta_t)$ where $\theta_t = \mu_t + \gamma_t + c_t$. Additional Gaussian noise term ϵ_t can also be included in θ_t using `SSMcustom` function (this is illustrated in Section 7)

`SSModel` contains three auxiliary functions, `SSMtrend`, `SSMcycle`, and `SSMseasonal`, for building structural time series. Argument `degree` of `SSMtrend` defines the degree of the polynomial component, where `1` corresponds to local level model and `2` to local linear trend model. Higher order polynomials can also be defined with larger values. Another important argument for `SSMtrend` is `Q` which defines the covariance structure of the trend component. This is typically a list of $p \times p$ matrices (with p being the number of series for which the component is defined), where the first matrix corresponds to level component (μ in (3)), second to slope component ν and so forth.

Function `SSMcycle` differs from `SSMtrend` only by one argument. `SSMcycle` does not have argument `degree`, but instead it has argument `period` which defines the length of the cycle c_t. Same argument is also used in function `SSMseasonal`, which contains also another important argument `sea.type`, which can be used to define whether user wants a `dummy` or `trigonometric` seasonal.

6.2. ARIMA models

ARIMA models are another typical time series modelling framework, which are also possible to define as a state space model. Auxiliary `SSMarima` defines ARIMA model using vectors `ar` and `ma`, which define the autoregressive and moving average coefficients respectively. Function assumes that all series defined by the `index` have the same coefficients. Argument `d` defines the degree of differencing, and logical argument `stationary` defines whether stationarity (after differencing) is assumed (if not, a diffuse initial states are used instead of stationary distribution). Univariate ARIMA(p,d,q) model can be written as

$$y_t^* = \phi_1 y_{t-1}^* + \ldots + \phi_p y_{t-p}^* + \xi_t + \theta_1 \xi_{t-1} + \ldots + \theta_q \xi_{t-q},$$

where $y_t^* = \Delta^d y_t$ and $\xi_t \sim \mathrm{N}(0, \sigma^2)$. Let $r = \max(p, q+1)$. **KFAS** defines the state space representation of ARIMA(p,d,q) model with stationary initial distribution as

$$Z_t^\top = \begin{pmatrix} 1_{d+1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}, T = \begin{pmatrix} U_d & 1_d^\top & 0 & \cdots & 0 \\ 0 & \phi_1 & 1 & & 0 \\ \vdots & & & \ddots & \\ \vdots & \phi_{r-1} & 0 & & 1 \\ 0 & \phi_r & 0 & \cdots & 0 \end{pmatrix}, R = \begin{pmatrix} 0_d \\ 1 \\ \theta_1 \\ \vdots \\ \theta_{r-1} \end{pmatrix},$$

$$\alpha_t = \begin{pmatrix} y_{t-1} \\ \vdots \\ \Delta^{d-1} y_{t-1} \\ y_t^* \\ \phi_2 y_{t-1}^* + \ldots + \phi_r y_{t-r+1}^* + \theta_1 \eta_t + \ldots + \theta_{r-1} \eta_{t-r+2} \\ \vdots \\ \phi_r y_{t-1}^* + \theta_{r-1} \eta_t \end{pmatrix}, Q = \sigma^2,$$

$$a_1 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, P_{*,1} = \begin{pmatrix} 0 & 0 \\ 0 & S_r \end{pmatrix}, P_{\infty,1} = \begin{pmatrix} I_d & 0 \\ 0 & 0 \end{pmatrix}, \eta_t = \xi_{t+1}$$

where $\phi_{p+1} = \ldots = \phi_r = \theta_{q+1} = \ldots = \theta_{r-1} = 0$, 1_{d+1} is a $1 \times (d+1)$ vector of ones, U_d is $d \times d$ upper triangular matrix of ones and S_r is the covariance matrix of stationary elements of α_1. The elements of the initial state vector α_1, which correspond to the differenced values $y_0, \ldots, \Delta^{d-1} y_0$ are treated as diffuse. The covariance matrix S_r can be computed by solving the linear equation $(I - T \otimes T)\mathrm{vec}(S_r) = \mathrm{vec}(RR^\top)$ (Durbin and Koopman 2012, p.138).

Note that the `arima` function from **stats** also uses the same state space approach to univariate ARIMA models for estimating model coefficients.

6.3. Linear and generalized linear models

An ordinary linear regression model

$$y_t = x_t^\top \beta + \epsilon_t, \quad t = 1, \ldots, n,$$

where $\epsilon_t \sim N(0, \sigma^2)$ can be written as a Gaussian state space model by defining $Z_t = x_t^\top$, $H_t = \sigma^2$, $R_t = Q_t = 0$ and $\alpha_t = \beta$. Assuming that the prior distribution of β is defined as

diffuse, the diffuse likelihood of this state space model corresponds to a restricted maximum likelihood (REML). Then the estimate for σ^2 obtained from `fitSSM` would be the familiar unbiased REML estimate of residual variance. It is important to notice that for this simple model numerical optimization is not needed, since we can estimate σ^2 by running the Kalman filter with $H_t = 1$, which gives us

$$\hat{\sigma}^2 = \frac{1}{\sum I(F_{\infty,t} = 0)} \sum_{t=1}^{n} I(F_{\infty,t} = 0) v_t^2 / F_t,$$

which equals to the REML estimate of σ^2. The initial Kalman filter already provides correct estimates of β as a_{n+1}, and running Kalman filter again with $H_t = \sigma^2$ gives also the covariance matrix of $\hat{\beta}$ as P_{n+1}.

The extension from linear model to generalized linear model is straightforward as the basic theory behind the exponential family state space modelling can be formulated from the theory of generalized linear models (GLM), and can be thought of as a extension to GLMs with additional dynamic structure. The iterative process of finding the approximating Gaussian model is equivalent with the famous iterative reweighted least squares (IRLS) algoritm (McCullagh and Nelder 1989, p. 40). If the model is ordinary GLM the final estimates of regression coefficients β and their standard errors coincide with maximum likelihood estimates obtained from ordinary GLM fitting. By adjusting the prior distribution for β we can use **KFAS** also for Bayesian analysis of Poisson and binomial regression (as those distributions do not depend on any additional parameters such as residual variance) with Gaussian prior.

A simple (generalized) linear model can be defined using `SSModel` without any auxiliary functions by defining the regression formula in the main part of the `formula`. For example the following code defines a Poisson GLM which is identical to the one found in help page of `glm`:

```
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
d.AD <- data.frame(treatment, outcome, counts)
glmModel1 <- SSModel(counts ~ outcome + treatment,
                     data = d.AD, distribution = "poisson")
```

The previous model could also be defined using the auxiliary function `SSMregression`:

```
glmModel2 <- SSModel(counts ~
                       SSMregression(~outcome + treatment, data = d.AD),
                     distribution = "poisson")
```

Note also the `data` argument in `SSMregression`. This overrides the `data` argument of `SSModel`, but both can exist at the same time. `SSModel` tries to be clever in finding the correct variables in case of multiple `data` arguments, see example in the help page of `SSModel` for an illustration.

If our observations are multivariate, a distinct regression components are defined for each of the series. For example, if counts `counts` above were a bivariate series, then both series

would have own regression coefficients but same covariate values. By using `SSMregression` explicitly, one could also define `type="common"` which would construct common regression coefficients for all series.

With `SSMregression` one can also define more complex regression models. The first argument of `SSMregression`, `rformula` can used to provide single formula or a list of formulas, where each component of the list contains the appropriate formula to be used for the corresponding series (ith formula in the list is used for the ith series defined by argument `index`). When `rformula` is a list, the `data` argument of `SSMregression` can be a single data frame (or environment), or a list of such data objects. If `data` is a list, ith element of that list is used for ith formula, and if `data` is a single data frame or environment, same data is used for all formulas.

6.4. Generalized linear mixed models

Just like in GLM setting, it is also possible to write generalized linear mixed model (GLMM) as a state space model. The difference between fixed and random effects lies in the initial state distribution; fixed effects are initialized via diffuse prior whereas random effects have proper variance defined by elements of P_1. Both types of states are automatically estimated by the Kalman filter, given the covariance structure of the random effects (and the residual variance or other parameters related to distribution of observation equation).

In practice, the mixed model formulation becomes quite cumbersome especially in hierarchical settings, but with large longitudinal settings it might still be useful to write mixed model as state space model, as it is then straightforward to add for example stochastic cycles or trends to the model. An example of defining the linear mixed model using the sleep study data from **lme4** (Bates, Maechler, Bolker, and Walker 2015, 2014) package proceeds as follows:

```
suppressWarnings(library("lme4",quietly=TRUE))
# Split data by grouping variable
Y <- split(sleepstudy["Reaction"], sleepstudy["Subject"])
p <- length(Y) # Number of series
Y <- matrix(unlist(Y), ncol = p,
            dimnames = list(NULL, paste("Subject", names(Y))))
dataf <- split(sleepstudy, sleepstudy["Subject"])

# Assume that we know the covariance structure
# of random effects and the residual variance

covRandom <- matrix(c(625,36,36,625),2,2)
sigma2 <- 650

# Common fixed effect part and distinct random effect parts for each "series"
# Set P1inf = 0 so diffuse initialization is not used for random effects
lmmModel <- SSModel(Y ~ -1
                    + SSMregression(rep(list(~ Days), p), type = "common",
                                    data = dataf, remove.intercept = FALSE)
                    + SSMregression(rep(list(~ Days), p), P1inf = 0,
```

```
                                    data = dataf, remove.intercept = FALSE),
                     H = diag(sigma2, p))
# Set covariance structure of the random effects which are states 3 to 38
# One could also use more common way lmmModel£P1[-(1:2),-(1:2)] <- ...
lmmModel["P1", 2+1:(2*p)] <-
  as.matrix(.bdiag(replicate(p, covRandom, simplify = FALSE)))
```

7. Illustration

I now illustrate the use of **KFAS** with an example case. Our data consists of alcohol related deaths in Finland for years 1969–2012, in age groups 30–39, 40–49, 50–59 and 60–69. We also have an offset term of yearly population sizes in corresponding age groups. The data is taken from Statistics Finland (2014a,b). As an illustration, we use only observations until 2007, and make predictions for years 2008–2013. Figure 1 shows the number of deaths per 100,000 persons for all age groups.

```
data("alcohol")
colnames(alcohol)

## [1] "death at age 30-39"      "death at age 40-49"
## [3] "death at age 50-59"      "death at age 60-69"
## [5] "population by age 30-39" "population by age 40-49"
## [7] "population by age 50-59" "population by age 60-69"

ts.plot(window(alcohol[,1:4]/alcohol[,5:8], end = 2007), col = 1:4,
        ylab = "Alcohol related deaths in Finland per 100,000 persons",
        xlab = "Year")
legend("topleft",col = 1:4, lty = 1,
       legend = colnames(alcohol)[1:4])
```

Natural distributional assumption for modelling counts is a Poisson distribution. Based on the time series plot, we can think of several candidates for capturing the time series aspects of the series. One could try for example an ARIMA model or a structural time series model such as local level or local linear trend. These are closely related, but I feel that the latter models are more easily interpretable so I will use structural time series approach.

Here I choose a multivariate Poisson model

$$\begin{aligned} p(y_t|\theta_t) &= Poisson(u_t e^{\theta_t}), \quad u_t = \text{population}_t, \\ \theta_t &= \mu_t + \epsilon_t, \quad \epsilon_t \sim N(0, Q_{\text{noise}}), \\ \mu_{t+1} &= \mu_t + \nu_t + \xi_t, \quad \xi_t \sim N(0, Q_{\text{level}}), \\ \nu_{t+1} &= \nu_t, \end{aligned}$$

where μ_t is the random walk with drift component, ν_t is a constant slope and ϵ_t is an additional white noise component which captures the extra variation of the series. I make no restrictions

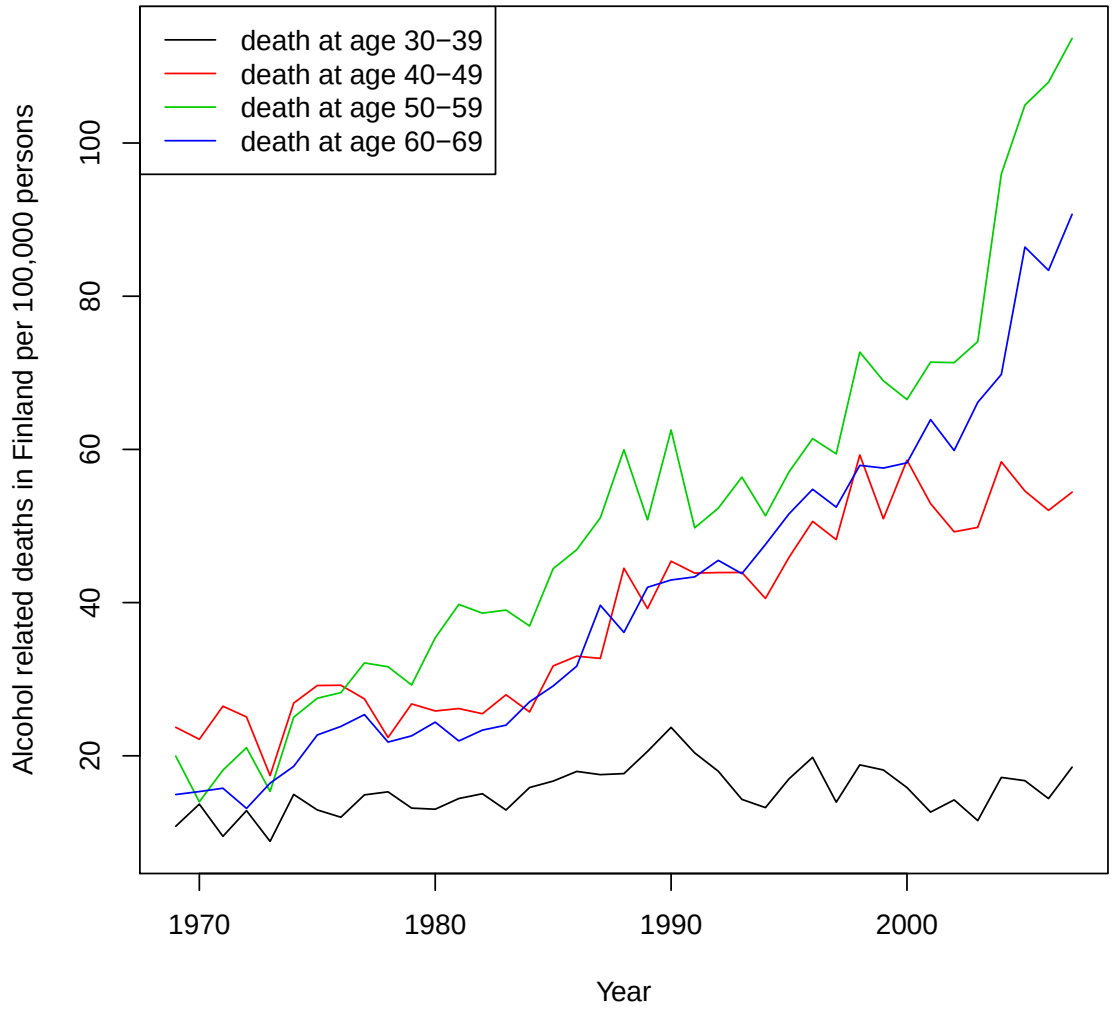

Figure 1: Alcohol related deaths per 100,000 persons in Finland in 1969–2007.

for the covariance structures of the level or the noise component. Note that the random walk with drift is a special case of local linear trend model where the covariance structure of the slope term is set to zero.

We estimate the model parameters first without simulation, and then using those estimates as initial values run the estimation procedure again with importance sampling. In this case, the results obtained from the importance sampling step are practically identical with the ones obtained from the initial step.

```
# remove the last observations
alcoholPred <- window(alcohol, start = 1969, end = 2007)
```

```
model <- SSModel(alcoholPred[,1:4] ~
                   SSMtrend(2, Q = list(matrix(NA,4,4), matrix(0,4,4))) +
                   SSMcustom(Z = diag(1,4), T = diag(0,4),
                             Q = matrix(NA,4,4), P1 = matrix(NA,4,4)),
                 distribution = "poisson", u = alcoholPred[,5:8])

# Model updating function for fitSSM
updatefn <- function(pars, model, ...){
  Q <- diag(exp(pars[1:4]))
  Q[upper.tri(Q)] <- pars[5:10]
  model["Q",etas="level"] <- crossprod(Q)
  Q <- diag(exp(pars[11:14]))
  Q[upper.tri(Q)] <- pars[15:20]
  model["Q",etas=9:12] <- model["P1",states=9:12] <- crossprod(Q)
  model
}
# Initial the covariance structure of the random walks and extra noise
# theta = log(intensity) = log(y/u)
# covariance matrices are parameterized via log-Cholesky in fitSSM
init <- chol(cov(log(alcoholPred[,1:4]/alcoholPred[,5:8]))/10)

fitinit <- fitSSM(model, updatefn = updatefn,
                  inits = rep(c(log(diag(init)), init[upper.tri(init)]),2),
                  method = "BFGS")
# Now with simulation
fit<-fitSSM(model, updatefn = updatefn,
            inits = fitinit$optim.out$par, method = "BFGS", nsim = 250)
varcor <- fit$model["Q", etas = "level"]
varcor[upper.tri(varcor)] <- cov2cor(varcor)[upper.tri(varcor)]
print(varcor,digits=2) #local level component

##        [,1]    [,2]   [,3]  [,4]
## [1,] 0.0074 0.66022 0.8062 0.856
## [2,] 0.0028 0.00239 0.1654 0.711
## [3,] 0.0040 0.00047 0.0034 0.755
## [4,] 0.0033 0.00156 0.0020 0.002

varcor <- fit$model["Q", etas = "custom"]
varcor[upper.tri(varcor)] <- cov2cor(varcor)[upper.tri(varcor)]
print(varcor,digits=2) #local level component #extra noise component

##         [,1]    [,2]    [,3]    [,4]
## [1,] 0.00537 0.73118 0.75627 8.0e-01
## [2,] 0.00315 0.00346 0.99924 9.9e-01
## [3,] 0.00295 0.00313 0.00283 1.0e+00
## [4,] 0.00043 0.00043 0.00039 5.4e-05
```

```
-fitinit$optim.out$val #log-likelihood without simulation

## [1] -704.8052

-fit$optim.out$val     #log-likelihood with simulation

## [1] -704.8034
```

Parameter estimation of state space model is often a difficult task, as the likelihood surface contains multiple maxima, thus making the optimization problem highly dependent on the initial values. Often the unknown parameters are related to the unobserved latent states such as the covariance matrix in this example, without much a priori knowledge. Therefore, it is challenging to guess good initial values especially in more complex settings and multiple initial value configurations possibly with several different type of optimization routines is recommended before one can be reasonably sure that proper optimum is found. Here we use the covariance matrix of the observed series as initial values for the covariance structures.

Another issue in case of non-Gaussian models is the fact that the likelihood computation is based on iterative procedure which is stopped using some stopping criteria (such as relative change of log-likelihood), so the function actually contains some noise. This in turn affects the gradient computations in BFGS and can in theory give unreliable results. Using derivative free method like Nelder-Mead is therefore sometimes recommended. On the other hand BFGS is usually much faster than Nelder-Mead and thus I prefer to try first BFGS at least in preliminary analysis.

Using function `KFS` we can compute the smoothed estimates of states:

```
out <- KFS(fit$model, nsim = 1000)
out

##
##  Smoothed values of states and standard errors at time n = 39:
##                            Estimate    Std. Error
## level.death at age 30-39   2.8559160   0.0784371
## slope.death at age 30-39   0.0107142   0.0137135
## level.death at age 40-49   4.0313117   0.0423763
## slope.death at age 40-49   0.0237188   0.0076318
## level.death at age 50-59   4.7578026   0.0398295
## slope.death at age 50-59   0.0503715   0.0095850
## level.death at age 60-69   4.4938371   0.0332897
## slope.death at age 60-69   0.0482386   0.0072090
## custom1                   -0.0004021   0.0603946
## custom2                   -0.0195488   0.0408846
## custom3                   -0.0169493   0.0370236
## custom4                   -0.0021345   0.0051427
```

From the output of `KFS` we see that the slope term is not significant in the first age group. For

time-varying states we can easily plot the estimated level and noise components, which shows clear trends in three age groups and highly correlated additional variation in all groups:

```
plot(coef(out,states=c("level","custom")),
     main = "Smoothed states", yax.flip=TRUE)
```

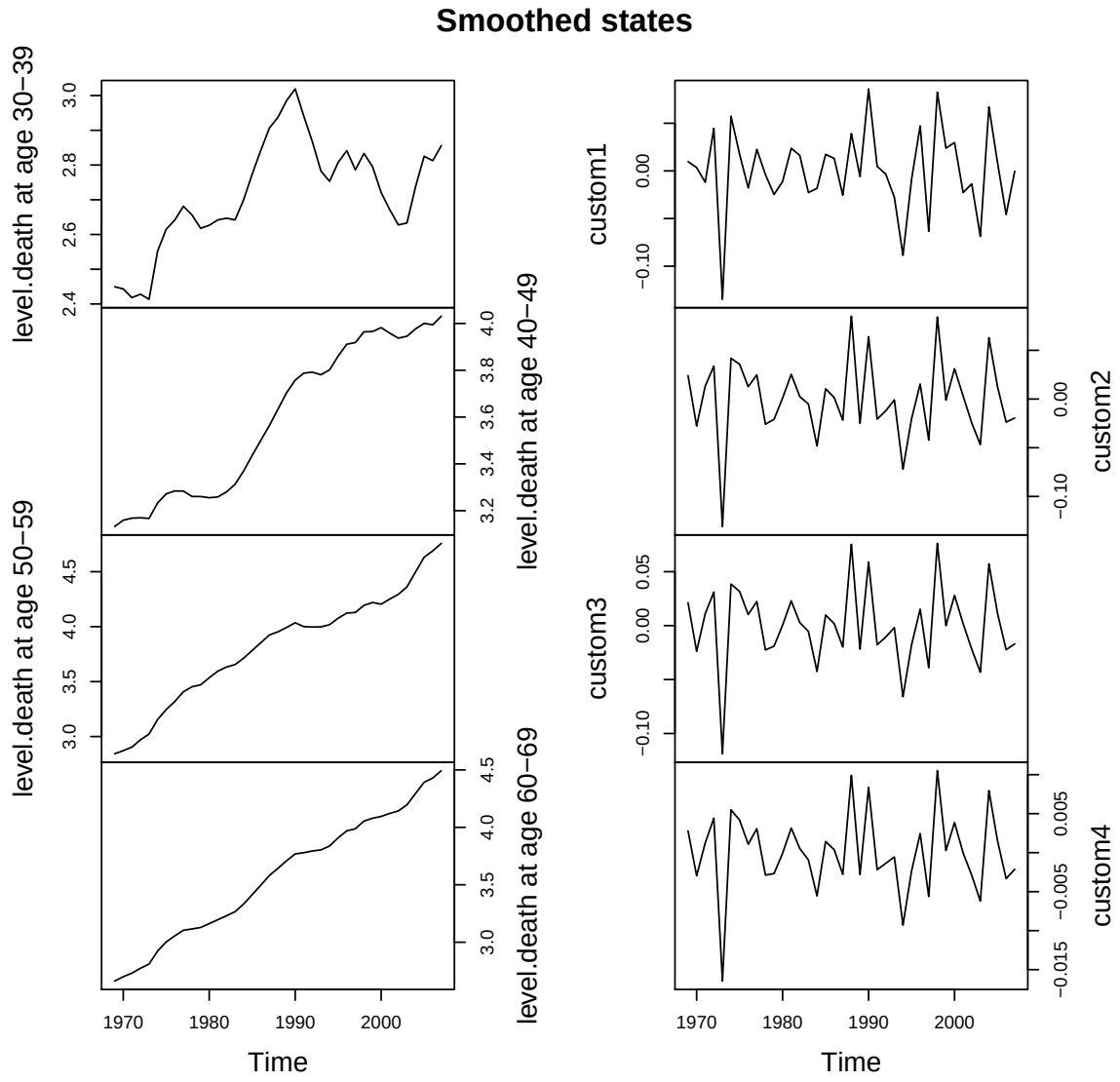

Figure 2: Smoothed level and white noise components.

Note the large drop in noise component which relates to possible outlier in 1973 of the mortality series. As an illustration of model diagnostics, we compute recursive residuals for our model and check whether there is autocorrelation left in the residuals (Figure 3).

```
res <- rstandard(KFS(fit$model, filtering = "mean",
                     smoothing = "none", nsim = 1000))
acf(res, na.action = na.pass)
```

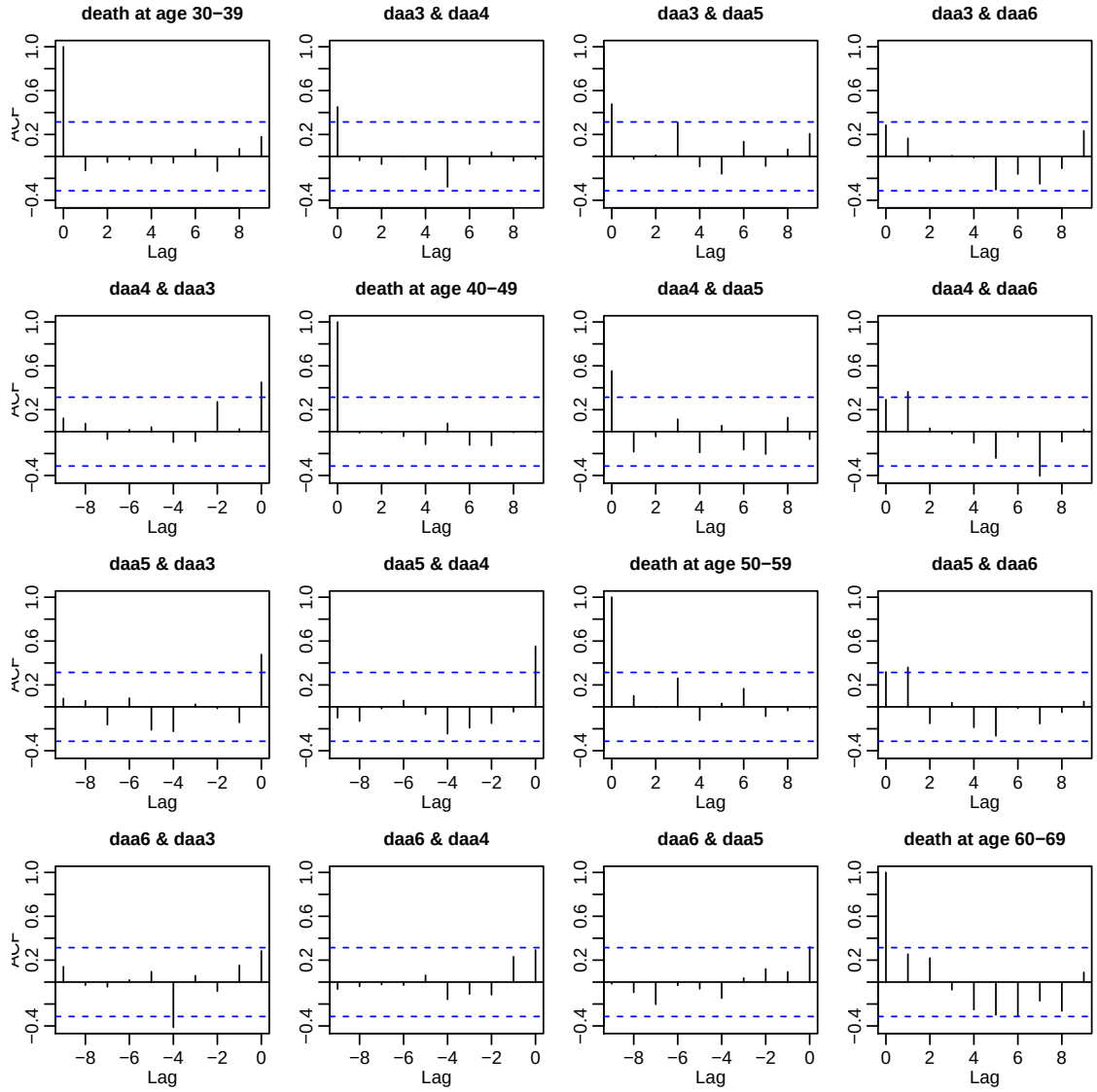

Figure 3: Autocorrelations and cross-correlations of recursive residuals.

We see occasional lagged cross-correlation between the residuals, but overall we can be relatively satisfied with our model.

We can now predict the intensity e^{θ_t} of alcohol related deaths per 100,000 persons for each age group for years 2008–2015 using our estimated model. As our model is time varying (`u` varies), we need to provide the model for the future observations via `newdata` argument. In this case we can use `SSMcustom` function and provide all the necessary system matrices as

once, together with constant `u=1` (our signal θ is already scaled properly as the original u_t was the population per 100,000 persons).

```
pred<-predict(fit$model, newdata =
                SSModel(ts(matrix(NA,6,4), start = 2008) ~ -1
                        + SSMcustom(Z = fit$model$Z, T = fit$model$T,
                                    R = fit$model$R, Q = fit$model$Q), u = 1,
                        distribution= "poisson"),
              interval = "confidence", nsim = 10000)
```

```
trend <- exp(signal(out, "trend")$signal)
par(mfrow = c(2,2), mar = c(2,2,2,2) + 0.1, oma = c(2,2,0,0))
for(i in 1:4)
  ts.plot(alcohol[,i]/alcohol[,4+i],
          trend[,i],
          pred[[i]],
          col = c(1,2,rep(3,3)), xlab = NULL, ylab = NULL,
          main = colnames(alcohol)[i])
mtext("Number of alcohol related deaths per 100,000 persons in Finland",
      side = 2, outer = TRUE)
mtext("Year",side=1,outer=TRUE)
```

Figure 4 shows the observed deaths, smoothed trends for 1969–2007, and intensity predictions for 2008–2012 together with 95% prediction intervals for intensity. When we compare our predictions to true observations, we see that in reality the number of deaths slightly increased in the oldest age group (ages 60–69), whereas in other age they decreased substantially during the forecasting period. This is partly explained by the fact that during this period the total alcohol consumption decreased almost monotonically, which in turn might have been caused by the increase in taxation of alcohol in 2008, 2009 and 2012.

8. Discussion

State space models offers tools for solving a large class of statistical problems. Here I introduced an R package **KFAS** for linear state space modelling where the observations are from an exponential family. With such a general framework, different aspects of the modelling need to be taken into account. Therefore the focus of the package has been to provide reliable and relatively fast tools for multiple inference problems, such as maximum likelihood estimation, filtering, smoothing and simulation. Compared to the early versions of **KFAS**, constructing a state space model with simple components is now possible without explicit definition of the system matrices by using the auxiliary functions and symbolic descriptions with the help of formula objects, which should greatly ease the use of the package.

Currently all the time consuming parts of **KFAS** are written in Fortran, which makes it relatively fast, given the general nature of problems **KFAS** can handle. Still, converting the package to C++ and `S4` classes with help of **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel 2013) could result potential improvements in terms of memory management, scalability and maintenance.

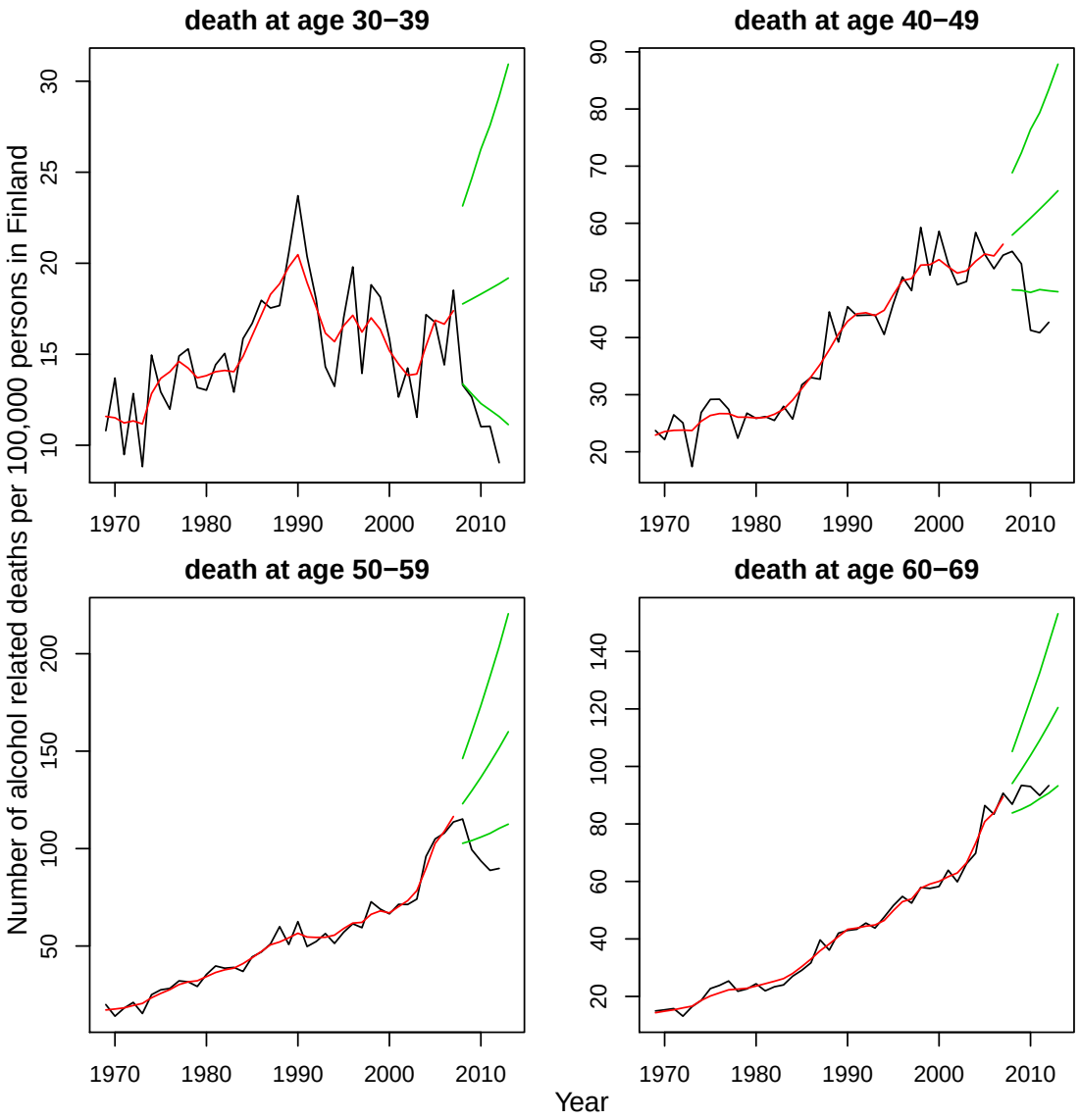

Figure 4: Observed number of alcohol related deaths per 100,000 persons in Finland (black), fitted values (red) and intensity predictions for years 2008–2012 together with 95% prediction intervals (green).

Acknowledgments

The author wishes to thank Jukka Nyblom and Patricia Menendez for the valuable comments and suggestions regarding the paper and the package. Author is also grateful for the financial support from Emil Aaltonen's Foundation.

A. Appendix: Filtering and smoothing recursions

The following formulas summarize the Kalman filtering and smoothing formulas for diffuse and sequential case and are based on Durbin and Koopman (2001) and related articles. The original formulas are somewhat scattered between the references with slightly different notations. Therefore I have collected the equations used in **KFAS** to this Appendix.

A.1. Filtering

Denote

$$a_{t+1} = \mathsf{E}(\alpha_{t+1}|y_t, \ldots, y_1) \quad \text{and}$$
$$P_{t+1} = \mathsf{VAR}(\alpha_{t+1}|y_t, \ldots, y_1).$$

The Kalman filter recursions for the general Gaussian model of form (1) are

$$\begin{aligned}
v_t &= y_t - Z_t a_t \\
F_t &= Z_t P_t Z_t^\top + H_t \\
K_t &= P_t Z_t^\top \\
a_{t+1} &= T_t(a_t + K_t F_t^{-1} v_t) \\
P_{t+1} &= T_t(P_t - K_t F_t^{-1} K_t^\top) T_t^\top + R_t Q_t R_t,
\end{aligned}$$

For the univariate approach, the filtering equations are

$$\begin{aligned}
v_{t,i} &= y_{t,i} - Z_{t,i} a_{t,i} \\
F_{t,i} &= Z_{t,i} P_{t,i} Z_{t,i}^\top + \sigma^2_{t,i} \\
K_{t,i} &= P_{t,i} Z_{t,i}^\top \\
a_{t,i+1} &= a_{t,i} + K_{t,i} F_{t,i}^{-1} v_{t,i} \\
P_{t,i+1} &= P_{t,i} - K_{t,i} K_{t,i}^\top F_{t,i}^{-1} \\
a_{t+1,1} &= T_t a_{t,p_t+1} \\
P_{t+1,1} &= T_t P_{t,p_t+1} T_t^\top + R_t Q_t R_t,
\end{aligned}$$

for $t = 1, \ldots, n$ and $i = 1, \ldots, p_t$, where $v_{t,i}$ and $F_{t,i}$ are scalars, $K_{t,i}$ is a column vector and $\sigma^2_{t,i}$ is the ith diagonal element of H_t. It is possible that $F_{t,i} = 0$, which case $a_{t,i+1} = a_{t,i}$, $P_{t,i+1} = P_{t,i}$, and $v_{t,i}$ is computed as usual.

The diffuse filtering equations for univariate approach are

$$\begin{aligned}
v_{t,i} &= y_{t,i} - Z_{t,i} a_{t,i} \\
F_{*,t,i} &= Z_{t,i} P_{*,t,i} Z_{t,i}^\top + \sigma^2_{t,i} \\
F_{\infty,t,i} &= Z_{t,i} P_{\infty,t,i} Z_{t,i}^\top \\
K_{*,t,i} &= P_{*,t,i} Z_{t,i}^\top \\
K_{\infty,t,i} &= P_{\infty,t,i} Z_{t,i}^\top,
\end{aligned}$$

and

$$
\begin{aligned}
a_{t,i+1} &= a_{t,i} + K_{\infty,t,i} v_{t,i} F_{\infty,t,i}^{-1} \\
P_{*,t,i+1} &= P_{*,t,i} + K_{\infty,t,i} K_{\infty,t,i}^\top F_{*,t,i} F_{\infty,t,i}^{-2} - (K_{*,t,i} K_{\infty,t,i}^\top + K_{*,t,i} K_{\infty,t,i}^\top) F_{\infty,t,i}^{-1} \\
P_{\infty,t,i+1} &= P_{\infty,t,i} - K_{\infty,t,i} K_{\infty,t,i}^\top F_{\infty,t,i}^{-1}
\end{aligned}
$$

if $F_{\infty,t,i} > 0$, and

$$
\begin{aligned}
a_{t,i+1} &= a_{t,i} + K_{*,t,i} v_{t,i} F_{*,t,i}^{-1} \\
P_{*,t,i+1} &= P_{*,t,i} - K_{*,t,i} K_{*,t,i}^\top F_{*,t,i}^{-1} \\
P_{\infty,t,i+1} &= P_{\infty,t,i},
\end{aligned}
$$

if $F_{\infty,t,i} = 0$. The transition equations from t to $t+1$ are

$$
\begin{aligned}
a_{t+1,1} &= T_t a_{t,p_t+1} \\
P_{*,t+1,1} &= T_t P_{*,t,p_t+1} T_t^\top + R_t Q_t R_t \\
P_{\infty,t+1,1} &= T_t P_{\infty,t,p_t+1} T_t^\top.
\end{aligned}
$$

A.2. Smoothing

Denote

$$
\begin{aligned}
\hat{\alpha}_t &= \mathsf{E}(\alpha_t | y_n, \ldots, y_1) \quad \text{and} \\
V_t &= \mathsf{VAR}(\alpha_t | y_n, \ldots, y_1).
\end{aligned}
$$

The smoothing algorithms of **KFAS** are based on the following recursions:

$$
\begin{aligned}
r_{t,i-1} &= Z_{t,i}^\top v_{t,i} F_{t,i}^{-1} + L_{t,i}^\top r_{t,i}, \\
r_{t-1,p_t} &= T_{t-1}^\top r_{t,0}, \\
N_{t,i-1} &= Z_{t,i}^\top Z_{t,i} F_{t,i}^{-1} + L_{t,i}^\top N_{t,i} L_{t,i}, \\
N_{t-1,p_t} &= T_{t-1}^\top N_{t,0} T_{t-1}, \\
L_{t,i} &= I - K_{t,i} Z_{t,i}^\top F_{t,i}^{-1},
\end{aligned}
$$

for $t = n, \ldots, 1$ and $i = p_t, \ldots, 1$, with $r_{n,p_n} = 0$ and $N_{n,p_n} = 0$. From these recursions, we get state smoothing recursions

$$
\begin{aligned}
\hat{\alpha}_t &= a_{t,1} + P_{t,1} r_{t,0} \\
V_t &= P_{t,1} - P_{t,1} N_{t,0} P_{t,1},
\end{aligned}
$$

and disturbance smoothing recursions

$$
\begin{aligned}
\hat{\epsilon}_{t,i} &= \sigma_{t,i}^2 F_{t,i}^{-1} (v_{t,i} - K_{t,i}^\top r_{t,i}), \\
\mathsf{VAR}(\hat{\epsilon}_{t,i}) &= \sigma_{t,i}^2 - \sigma_{t,i}^4 (F_{t,i}^{-1} - K_{t,i}^\top N_{t,i} K_{t,i} F_{t,i}^{-2}), \\
\hat{\eta}_t &= Q_t R_t^\top r_{t,0}, \\
\mathsf{VAR}(\hat{\eta}_{t,i}) &= Q_t R_t^\top N_{t,0} R_t Q_t.
\end{aligned}
$$

The recursions for diffuse phase are as follows.

$$
\begin{aligned}
L_{\infty,t,i} &= I - K_{\infty,t,i} Z_{t,i} F_{\infty,t,i}^{-1},\\
L_{t,i} &= (K_{\infty,t,i} F_{t,i} F_{\infty,t,i}^{-1} - K_{t,i}) Z_{t,i} F_{\infty,t,i}^{-1},\\
r_{0,t,i-1} &= L_{\infty,t,i}^\top r_{0,t,i},\\
r_{1,t,i-1} &= Z_{t,i}^\top v_{t,i} F_{\infty,t,i}^{-1} + L_{\infty,t,i}^\top r_{1,t,i} + L_{t,i}^\top r_{0,t,i},\\
N_{0,t,i-1} &= L_{\infty,t,i}^\top N_{0,t,i} L_{\infty,t,i}\\
N_{1,t,i-1} &= L_{t,i}^\top N_{0,t,i} L_{\infty,t,i} + L_{\infty,t,i}^\top N_{1,t,i} L_{\infty,t,i} + Z_{t,i}^\top Z_{t,i} F_{\infty,t,i}^{-1},\\
N_{2,t,i-1} &= L_{t,i}^\top N_{0,t,i} L_{t,i} + L_{\infty,t,i}^\top N_{1,t,i} L_{t,i} + (L_{\infty,t,i}^\top N_{1,t,i} L_{t,i})^\top + L_{\infty,t,i} N_{2,t,i}^\top L_{\infty,t,i} - Z_{t,i}^\top Z_{t,i} F_{t,i} F_{\infty,t,i}^{-2},\\
N_{t-1,p_t} &= T_{t-1}^\top N_{t,0} T_{t-1},
\end{aligned}
$$

if $F_{\infty,t,i} > 0$, and

$$
\begin{aligned}
L_{t,i} &= I - K_{t,i} Z_{t,i} F_{t,i}^{-1},\\
r_{0,t,i-1} &= Z_{t,i}^\top v_{t,i} F_{t,i}^{-1} + L_{t,i}^\top r_{0,t,i},\\
r_{1,t,i-1} &= L_{t,i}^\top r_{1,t,i},\\
N_{0,t,i-1} &= L_{t,i}^\top N_{0,t,i} L_{t,i} + Z_{t,i}^\top Z_{t,i} F_{t,i}^{-1}\\
N_{1,t,i-1} &= N_{1,t,i} L_{t,i}\\
N_{2,t,i-1} &= N_{2,t,i} L_{t,i},
\end{aligned}
$$

otherwise. The transition from time t to $t-1$ is by $N_{j,t-1,p_t} = T_{t-1}^\top N_{j,t,0} T_{t-1}$ for $j = 0, 1, 2$, and $r_{j,t-1,p_t} = T_{t-1}^\top r_{j,t,0}$ for $j = 0, 1$, with $r_{0,d,j} = r_{d,j}$, $r_{1,d,j} = 0$, $N_{0,d,j} = N_{d,j}$, and $N_{1,d,j} = N_{2,d,j} = 0$, where (d, j) is the last point of diffuse phase. From these basic recursions, we get state smoothing recursions for diffuse phase as

$$
\begin{aligned}
\hat{\alpha}_t &= a_{t,1} + P_{t,1} r_{0,t,0} + P_{\infty,t,1} r_{1,t,0},\\
V_t &= P_{t,1} - P_{t,1} N_{0,t,0} P_{t,1} - (P_{\infty,t,1} N_{1,t,0} P_{t,1})^\top - P_{\infty,t,1} N_{1,t,0} P_{t,1} - P_{\infty,t,1} N_{2,t,0} P_{\infty,t,1},
\end{aligned}
$$

and disturbance smoothing recursions

$$
\begin{aligned}
\hat{\epsilon}_{t,i} &= -\sigma_{t,i}^2 K_{\infty,t,i}^\top r_{0,t,i},\\
\mathsf{VAR}(\hat{\epsilon}_{t,i}) &= \sigma_{t,i}^2 - \sigma_{t,i}^4 K_{\infty,t,i}^\top N_{0,t,i} K_{\infty,t,i} F_{\infty,t,i}^{-2},
\end{aligned}
$$

if $F_{\infty,t,i} > 0$, and

$$
\begin{aligned}
\hat{\epsilon}_{t,i} &= -\sigma_{t,i}^2 (v_{t,i} F_{\infty,t,i}^{-1} - K_{t,i}^\top r_{0,t,i}),\\
\mathsf{VAR}(\hat{\epsilon}_{t,i}) &= \sigma_{t,i}^2 - \sigma_{t,i}^4 (F_{t,i}^{-1} - K_{t,i}^\top N_{0,t,i} K_{t,i} F_{t,i}^{-2}),
\end{aligned}
$$

if $F_{\infty,t,i} = 0$. For $\hat{\eta}$, recursions are

$$
\begin{aligned}
\hat{\eta}_t &= Q_t R_t^\top r_{0,t,0},\\
\mathsf{VAR}(\hat{\eta}_{t,i}) &= Q_t R_t^\top N_{0,t,0} R_t Q_t.
\end{aligned}
$$

References

Anderson B, Moore J (1979). *Optimal Filtering*. Prentice-Hall, Englewood Cliffs.

Bates D, Maechler M, Bolker BM, Walker S (2014). ***lme4****: Linear Mixed-Effects Models Using Eigen and S4*. R package version version 1.1-8, URL http://CRAN.R-project.org/package=lme4.

Bates D, Maechler M, Bolker BM, Walker S (2015). "**lme4**: Linear Mixed-Effects Models Using Eigen and S4." *Submitted*. URL http://arxiv.org/abs/1406.5823.

Chowdhury KR (2014). ***rucm****: Implementation of Unobserved Components Model (UCM) in R*. R package version 0.4, URL http://CRAN.R-project.org/package=rucm.

Dethlefsen C, Lundbye-Christensen S, Christensen L (2012). ***sspir****: State Space Models in R*. Orphaned on 2013-12-20, URL http://cran.r-project.org/src/contrib/Archive/sspir/.

Durbin J, Koopman SJ (2000). "Time Series Analysis of Non-Gaussian Observations Based on State Space Models from Both Classical and Bayesian Perspectives." *Journal of Royal Statistical Society B*, **62**, 3–56.

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

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

Durbin J, Koopman SJ (2012). *Time Series Analysis by State Space Methods (Second Edition)*. Oxford University Press, New York.

Eddelbuettel D (2013). *Seamless R and C++ Integration with* ***Rcpp***. Springer, New York.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. URL http://www.jstatsoft.org/v40/i08/.

Francke MK, Koopman SJ, De Vos AF (2010). "Likelihood Functions for State Space Models with Diffuse Initial Conditions." *Journal of Time Series Analysis*, **31**(6), 407–414. ISSN 1467-9892. doi:10.1111/j.1467-9892.2010.00673.x. URL http://dx.doi.org/10.1111/j.1467-9892.2010.00673.x.

Harvey AC, Koopman SJ (1992). "Diagnostic Checking of Unobserved-Components Time Series Models." *Journal of Business & Economic Statistics*, **10**(4), 377–89.

Holmes E, Ward E, Wills K (2013). ***MARSS****: Multivariate Autoregressive State-Space Modeling*. R package version 3.9, URL http://cran.r-project.org/web/packages/MARSS/.

Holmes EE, Ward EJ, Wills K (2012). "**MARSS**: Multivariate Autoregressive State-space models for Analyzing Time-series Data." *The R Journal*, **4**(1), 30.

King AA, Ionides EL, Bretó CM, Ellner SP, Ferrari MJ, Kendall BE, Lavine M, Nguyen D, Reuman DC, Wearing H, Wood SN (2014). ***pomp****: Statistical Inference for Partially Observed Markov Processes (R Package)*. URL http://pomp.r-forge.r-rproject.org.

Koopman SJ, Durbin J (2000). "Fast Filtering and Smoothing for Multivariate State Space Models." *Journal of Time Series Analysis*, **21**, 281–296. doi:10.1111/1467-9892.00186.

Koopman SJ, Durbin J (2003). "Filtering and Smoothing of State Vector for Diffuse State-space Models." *Journal of Time Series Analysis*, **24**, 85–98. doi:10.1111/1467-9892.0029.

McCullagh P, Nelder JA (1989). *Generalized Linear Models (Second Edition)*. London: Chapman & Hall.

Petris G, Petrone S (2011). "State Space Models in R." *Journal of Statistical Software*, **41**(4), 1–25. ISSN 1548-7660. URL http://www.jstatsoft.org/v41/i04.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rue H, Martino S, Lindgren F, Simpson D, Riebler A, Krainski ET (2015). ***INLA**: Functions Which Allow to Perform Full Bayesian Analysis of Latent Gaussian Models Using Integrated Nested Laplace Approximaxion*. R package version 0.0-1420281647.

Statistics Finland (2014a). "Deaths by Gender, Age and Underlying Cause of Death 1969-2013." URL http://pxweb2.stat.fi/database/StatFin/ter/ksyyt/ksyyt_en.asp.

Statistics Finland (2014b). "Population According to Age (5-year) and Sex in the Whole Country 1865 - 2014." URL pxweb2.stat.fi/database/StatFin/vrm/vaerak/vaerak_en.asp.

Szymanski C (2014). ***dlmodeler**: Generalized Dynamic Linear Modeler*. R package version 1.4-2, URL http://CRAN.R-project.org/package=dlmodeler.

Tusell F (2011). "Kalman Filtering in R." *Journal of Statistical Software*, **39**(2), 1–27. ISSN 1548-7660. URL http://www.jstatsoft.org/v39/i02.

Affiliation:

Jouni Helske
University of Jyväskylä
Department of Mathematics and Statistics
40014 Jyväskylä, Finland
E-mail: Jouni.Helske@jyu.fi