Dynamichazard

Dynamic Hazard Models Using State Space Models

Christoffersen, Benjamin

Document Version
Final published version

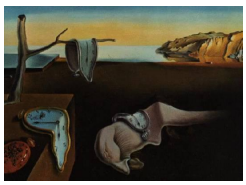
Published in:
Journal of Statistical Software

DOI:
10.18637/jss.v099.i07

Publication date:
2021

License
CC BY

Citation for published version (APA):
Christoffersen, B. (2021). Dynamichazard: Dynamic Hazard Models Using State Space Models. *Journal of Statistical Software*, *99*(7), 1-38. https://doi.org/10.18637/jss.v099.i07



Journal of Statistical Software

September 2021, Volume 99, Issue 7. *doi: 10.18637/jss.v099.i07*

dynamichazard: Dynamic Hazard Models Using State Space Models

Benjamin Christoffersen
Copenhagen Business School

Abstract

The **dynamichazard** package implements state space models that can provide a computationally efficient way to model time-varying parameters in survival analysis. I cover the models and some of the estimation methods implemented in **dynamichazard**, apply them to a large data set, and perform a simulation study to illustrate the methods' computation time and performance. One of the methods is compared with other models implemented in R which allow for left-truncation, right-censoring, time-varying covariates, and time-varying parameters.

Keywords: survival analysis, time-varying parameters, extended Kalman filter, EM algorithm, unscented Kalman filter, parallel computing, R, **Rcpp**, **RcppArmadillo**.

1. Introduction

The **dynamichazard** package is for survival analysis with time-varying parameters using state space models. The contribution of this paper is to give an overview of computationally fast nonlinear filtering methods for state space models in survival analysis that scale well in the dimension of the observational equation and illustrate the interface in **dynamichazard** for the methods.

I will start by motivating why one would consider time-varying parameters with the Cox proportional hazards model (Cox 1972) and give a short overview of available software to estimate time-varying parameters in survival analysis. All mentioned packages or functions are in R (R Core Team 2021) unless stated otherwise. For simplicity, we start with n individuals where each individual $i = 1, \dots, n$ have a single fixed (not time-varying) covariate x_i and a stop time T_i. Later we look at time-varying multivariate covariate vectors, delayed-entry (also known as left-truncation), and random right-censoring. All three can be handled by the methods in the **dynamichazard** package. Denote the instantaneous hazard rate of an event

for individual i at time t by

$$\lambda\left(t \mid x_i\right) = \lim_{h \rightarrow 0^+} \frac{\mathsf{P}\left(t \leq T \leq t + h \mid T \geq t, x_i\right)}{h}$$

which can be interpreted as the rate of an event over an infinitesimal unit of time. The Cox proportional hazard model is a commonly used model where the instantaneous hazard rate is

$$\lambda\left(t \mid x_i\right) = \lambda_0(t) \exp\left(\beta x_i\right)$$

where $\lambda_0(t)$ is a nonparametric baseline hazard and β is the single parameter in the model. One advantage of the Cox proportional hazard model is the ease of interpreting the parameter: $\exp(\beta) = \lambda\left(t \mid x_i = x' + 1\right) / \lambda\left(t \mid x_i = x'\right)$ is the proportional change of the hazard of a unit increase of the covariate regardless of time, t. However, the effect of a covariate may change across time. For instance, suppose we look at the effect of a drug on the risk of a specific disease and we use age as the time variable. Then different dose levels of a drug may not have the same proportional effect for an adult as for a child.

One way to relax the proportional hazard assumption is to use an interaction between the covariate and a deterministic function of time such that the instantaneous hazard rate is

$$\lambda\left(t \mid x_i\right) = \lambda_0(t) \exp\left(\boldsymbol{\beta}^\top \boldsymbol{g}(t) x_i\right) \tag{1}$$

I will refer to the elements of $\boldsymbol{\beta}$ as coefficients and the dot product $\boldsymbol{\beta}^\top \boldsymbol{g}(t)$ as a time-varying parameter to avoid confusion. Thomas and Reyes (2014) show how to estimate the model in Equation 1 in R using the `coxph` function in the **survival** package (Therneau 2021; Therneau and Grambsch 2000), and provide macros to do it in SAS (SAS Institute Inc 2017). It has become even easier with the `coxph` function after Thomas and Reyes' article was published because of the new `tt` argument of `coxph`. Similar functionality is available in Stata (StataCorp 2017) using the `stcox` command with the `tvc` and `texp` arguments. The model can also be estimated in R with the `cph` function in the **rms** package (Harrell Jr. 2021), and the `coxreg` function in the **eha** package (Broström 2012).

A downside is that the researcher has to specify the function $\boldsymbol{g}(t)$. A flexible choice is to use a spline such that $\boldsymbol{g}(t) = \left(g_1(t), g_2(t), \ldots, g_k(t)\right)^\top$ where g_is are basis functions. This is done in the **dynsurv** package (Wang, Chen, Wang, and Yan 2020) with the `splineCox` function which ultimately uses the `coxph` function in the **survival** package. However, models with several covariates with time-varying effects have a lot of coefficients, and the researcher has to choose the number of knots, and placement of the knots. An alternative for the Cox model is the nonparametric Cox model in the `timecox` function in the **timereg** package (Martinussen and Scheike 2006). The downside of all the methods is that the researcher has to choose hyperparameters where only some of the implementations provide an automated procedure to select the hyperparameters.

Another option is to use the Aalen's additive regression model (Aalen 1989) where the instantaneous hazard rate is

$$\lambda\left(t \mid x_i\right) = \lambda_0(t) + \beta(t) x_i$$

where $\beta(t)$ is estimated nonparametrically. The Aalen model can be estimated in R with the `aareg` function in the **survival** package, and the `aalen` function in the **timereg** package. The `stlh` command can be used Stata and the **lifelines** package (Davidson-Pilon *et al.* 2021) can

be used in Python (Rossum 2017). An issue with the Aalen model is that the estimate of the instantaneous hazard rate can become negative.

A drawback of the nonparametric and semiparametric methods is that they cannot be used to make prediction outside the time range used in the estimation due to the nonparametric parts of the hazard. This is an issue for instance when the objective is to make predictions about the future and we use calendar time as the time scale. One solution is to use a fully parametric function for the cumulative hazard denoted by

$$\Lambda\left(t \mid x_i\right) = \int_0^t \lambda\left(z \mid x_i\right) dz$$

In particular, we can model the log cumulative hazard function with a restricted cubic spline for the intercept such that

$$\Lambda\left(t \mid x_i\right) = \exp\left(\boldsymbol{\gamma}^\top \boldsymbol{k}\left(\log(t)\right) + \boldsymbol{\beta}^\top \boldsymbol{g}\left(\log(t)\right) x_i\right) \tag{2}$$

where $\boldsymbol{\gamma}$ is a coefficient vector, $\boldsymbol{k}(z)$ is a vector of basis functions, and $\boldsymbol{g}(z)$ is a vector of basis functions to get a time-varying parameter like in Equation 1. This is implemented in the **rstpm2** package (Clements, Liu, and Christoffersen 2021; Liu, Pawitan, and Clements 2016, 2017), the **flexsurv** package (Jackson 2016), and the `stpm2` command (Lambert and Royston 2009) in Stata. All of the packages can fit other models than generalization like in Equation 2 of the proportional hazard model (the special case of Equation 2 when $\boldsymbol{g}\left(\log(t)\right)$ is constant). Further, the **rstpm2** package includes penalized methods. This is useful as it allows for flexible splines with a large number of basis functions that do not overfit. The `hare` function in the **polspline** package (Kooperberg 2020) is another alternative which uses linear splines instead of restricted cubic splines, and models the hazard function such that

$$\lambda\left(t \mid x_i\right) = \exp\left(\boldsymbol{\gamma}^\top \boldsymbol{k}(t) + \boldsymbol{\beta}^\top \boldsymbol{g}(t) x_i\right)$$

Another alternative is to consider discrete time hazard models. Let T be the event time and

$$Y_t = \begin{cases} 1 & \text{if } T \in (t-1, t] \\ 0 & \text{otherwise} \end{cases}, \qquad t = 1, 2, \ldots$$

be an indicator for whether there is an event between time $t-1$ and t. Then we model the conditional probability of event given survival up to time $t-1$ by

$$l\left(\mathsf{P}\left(Y_t = 1 \mid \boldsymbol{\alpha}_t, T > t-1\right)\right) = \boldsymbol{\gamma}^\top \boldsymbol{k}\left(t\right) + \boldsymbol{\beta}^\top \boldsymbol{g}\left(t\right) x_i \tag{3}$$

where l is a link function and $\boldsymbol{k}(z)$ and $\boldsymbol{g}(z)$ are vectors of basis functions to get a time-varying parameter as before (see Tutz and Schmid 2016, Chapter 5 for examples). Equation 3 is the discrete hazard rate on the link scale. Software to penalize the coefficients with, potentially, large dimensional $\boldsymbol{k}$ and $\boldsymbol{g}$ are available and well established. As mentioned with the **rstpm2** package, this is important to allow for high dimensional splines that do not overfit. A few examples of packages are **glmnet** (Simon, Friedman, Hastie, and Tibshirani 2011), **glmpath** (Park and Hastie 2018), **mgcv** (Wood 2006), and **penalized** (Goeman 2010). A discrete hazard model is an obvious choice if the outcomes are only observed in discrete time intervals and may yield similar results to a continuous time model if the discrete time periods are sufficiently narrow and the time of events is observed. A nonparametric alternative is the temporal

process regression in the **tpr** package (Fine, Yan, and Kosorok 2004; Yan and Fine 2004) which uses nonparametric time-varying effects in generalized linear models.

While all of the parametric models allow for extrapolation beyond the observed time period, the values of the prediction depend on the chosen type of splines. Some other survival analysis options in R are the **pch** package (Frumento 2021) and **eha** package. The **pch** package and the `pchreg` function in the **eha** package fit time-varying parameters by dividing time into d periods each respectively covering the period $(s_0, s_1], (s_1, s_2], \ldots, (s_{d-1}, s_d]$ and using separate coefficients in each interval for time-varying parameters. Thus, instantaneous hazards are piecewise constant. If all parameters are time-varying then the instantaneous hazard rate is

$$\lambda\left(t \mid x_i\right) = \exp\left(\gamma_k + \beta_k x_i\right), \qquad k : s_{k-1} < t \leq s_k \tag{4}$$

Similar models are easily estimated with `streg` and `stsplit` commands in Stata or a bit of pre-processing followed by the `transreg` procedure in SAS. The models have a lot of coefficients even with a moderate amount of time periods, and can yield unstable coefficients with large jumps. Forecasting of future outcomes conditional on the covariate are predictions from an exponential distribution for the most recent period, $(s_{d-1}, s_d]$, which may be based on a sparse amount of data. Moreover, the number of points where the coefficients jump, d, and the location of the jumps, $s_1, s_2, \ldots, s_d$, have to be chosen. The `bayesCox` function in the **dynsurv** package alleviates these issues in a Bayesian analysis where the number of jumps and location of the jumps is a random variable over a fixed size grid of jump locations, the baseline hazard in each interval is gamma distributed, and the coefficients for the covariates follow a first order random walk. Though, the implemented Markov chain Monte Carlo (MCMC) methods is very computationally expensive.

Package **dynamichazard** (Christoffersen 2021) is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=dynamichazard`. **dynamichazard** adds to the existing literature by providing a simple and efficient implementation of the models covered by Fahrmeir (1992, 1994). In the two papers, Fahrmeir shows how to model time-varying parameters by using discrete time state space models where the parameters are assumed to be piecewise constant. One possible model in this framework is a model with instantaneous hazard rate like in the piecewise constant hazard model shown in Equation 4 given by

$$\begin{aligned} \lambda\left(t \mid x_i\right) &= \exp\left(\gamma_k + \beta_k x_i\right), \qquad k = \lceil t \rceil \\ \left(\gamma_k, \beta_k\right) &\sim f\left(\gamma_{k-1}, \beta_{k-1}\right) \end{aligned} \tag{5}$$

where f is a multivariate normal distribution with a mean depending on γ_{k-1} and β_{k-1}, $\lceil t \rceil$ is the ceiling of t, and we use time periods with length 1. This is implemented in the **dynamichazard** package. An advantage of the state space approach is that the issues with the number of coefficients in the piecewise constant hazard model shown in Equation 4 are eased by the dependence induced through f. Further, the state space model provides a parametric model for the parameters that allows one to project future parameter values. Thus, it is easy to make future forecasts, and all available data is used in the estimation. Moreover, the models can be estimated approximately with fast methods with a linear computational cost relative to the number of observed individuals, cubic computational cost relative to the number of parameters, and are easily computed in parallel.

There is a lot of software options for fitting general state space models. Two reviews of packages in R for linear Gaussian models from 2011 are Petris and Petrone (2011) and Tusell (2011). They only briefly mention nonlinear methods. The **KFAS** package (Helske 2017) is the most closely related package to **dynamichazard**. **KFAS** can be used for survival analysis although this is not the primary focus of the package. The researcher can also estimate the models in **dynamichazard** with software like the **pomp** package (King, Nguyen, and Ionides 2016; King, Ionides, and Bretó 2021) in R, Stan (Carpenter *et al.* 2017), the **SSM** toolbox (Peng and Aston 2011) in MATLAB, the **Control System Toolbox** in MATLAB, the **SsfPack** library (Koopman, Shephard, and Doornik 2008) in C, the **pyParticleEst** library (Nordh 2017) in Python, the **vSMC** library (Zhou 2015) in C++, and the **SMCTC** library (Johansen 2009) in C++ just to name a few. Because all of these are quite general, using them to set up models like those in **dynamichazard** is cumbersome or computationally expensive. Some are computationally expensive as they are intended for a few outcomes at each time point which is common in the state space model literature. This is not the case for the model given in Equation 5 as the number of observations at risk at each point in time may be large.

This package is motivated by Fahrmeir (1992, 1994). The current implementation uses the expectation maximization (EM) algorithm from these papers. The reader may want further information on the filters covered later, as this paper only introduces them briefly. Durbin and Koopman (2012, Chapter 4) cover the Kalman filter, which provides a basis for understanding all the filters in the package. Fahrmeir (1992, 1994) covers the extended Kalman filter this package uses, whereas Durbin and Koopman (2012, Section 10.2) cover the more common form of the extended Kalman filter. Durbin and Koopman (2012, Section 10.3) and Wan and Merwe (2000) provide an introduction to the unscented Kalman filter. Another resource is Hartikainen, Solin, and Särkkä (2011) who introduce the Kalman filter, extended Kalman filter, and unscented Kalman filter.

The hard disk data set example is mainly chosen because it is publicly available and moderately large. It contains data on the hard disk failure times from the time of installation. The hard disks are from Backblaze which a data storage provider. The hard disk survival time seem to differ substantially between both manufacturers and hard disk versions motivating a different process for each hard disk version.

The typical application of the implemented models are cases where we expect time-varying effects, assume that a model like in Equation 5 is a good approximation of the hazard rates, and we are interested in future forecasts. Examples are churn analysis where the rate at which customers leave a company may have non-constant associations with observable variables due to e.g., a competitor who launches a marketing campaign targeted towards a group of customers, or firm default prediction where changes in banks lending behavior may effect the rate at which firms with high debt default. The examples can be modeled with calendar time as the time scale and using delayed entry for customers who join a company's service at different points in time or firms who incorporate at different points in time. Another example is mortality rates in life insurance where the rates may be varying in calendar time.

In all cases, we may be interested in predicting future hazard rates and not present ones. Thus, having a model for the relation between present parameter values and future parameter values is useful. The hard disk data set presented later is similar in the sense that hard disks of the same type are typically installed within a short time period. Thus, we do not have data for a given type of hard disk in the range we are interested as they are all installed at roughly the same time.

All methods are implemented in C++ with use of **BLAS** and **LAPACK** (Anderson *et al.* 1999) either by direct calls to the methods or through the C++ library **Armadillo** (Sanderson and Curtin 2016). The implemented estimations methods are fast because the algorithms are fast, the methods are implemented in C++, and most of them support computations in parallel. The reported computational complexities in the rest of the paper are based on a single iteration of the EM algorithm.

The rest of this paper is organized as follows. Section 3 covers the discrete hazard model implemented in the package. Section 4 shows the EM algorithm on which all the methods are based, followed by four different filters used in the E-step. A data set with hard disk failures will be used throughout this section to illustrates how to use the methods. Section 5 covers the two implemented models. Section 6 illustrates the methods' performance and the computation time of the methods on simulated data. One of the methods is compared with some of the above mentioned methods from other packages in Section 7. I conclude and discuss extensions in Section 8.

2. Notation

It is assumed that the reader has some familiarity with survival analysis. However, since the combination of survival analysis and state space models is less common, I will cover the main state space notation in this section. We will separate time zero to the ceiling of the maximum observed time into d equidistant periods of length one during estimation of the model. Let $\boldsymbol{Y}_t$ denote the vector of the individuals' observed truncated event time in interval t (made precise later) or binary event indicators within time period t for the individuals who are at-risk in the tth interval. Thus, the number of elements of $\boldsymbol{Y}_t$ varies depending on the risk set in the tth interval.

R_t and $\mathcal{R}_t$ respectively denotes the discrete and continuous risk set in interval t. Further, i_{kt} denotes the index value of the kth element of R_t or $\mathcal{R}_t$ and n_t denotes the number elements in R_t or $\mathcal{R}_t$. Which of the two is given by the context in which i_{kt} and n_t is used.

There will be $d + 1$ so-called *state vectors* denoted by $\boldsymbol{\alpha}_0, \dots, \boldsymbol{\alpha}_d$ with $\boldsymbol{\alpha}_t \in \mathbb{R}^q$ which each contains the assumed time-varying parameters' values in a given interval. $\boldsymbol{z}_t(\boldsymbol{\alpha}_t)$ denotes the conditional mean of $\boldsymbol{Y}_t$ given the state vector $\boldsymbol{\alpha}_t$ and $\mathbf{H}_t(\boldsymbol{\alpha}_t)$ denotes the corresponding conditional covariance matrix which is assumed to be a diagonal matrix. Both $\boldsymbol{z}_t$ and $\mathbf{H}_t$ implicitly depends on an assumed link function g, and potentially time-varying covariates $\boldsymbol{x}_{ij} \in \mathbb{R}^m$. h will denote the inverse of the link function g. p denotes a (conditional) density functions or probability mass functions which specification is implicitly given by the context and its arguments.

The following notation is used for the conditional mean and covariance matrix for the state vector

$$\boldsymbol{a}_{t|s} = \mathsf{E}\left(\boldsymbol{\alpha}_t \mid \boldsymbol{y}_1, \dots, \boldsymbol{y}_s\right), \qquad \mathbf{V}_{t|s} = \mathsf{VAR}\left(\boldsymbol{\alpha}_t \mid \boldsymbol{y}_1, \dots, \boldsymbol{y}_s\right)$$

Notice that the letter 'a' is used for the mean estimates, whereas 'α' is used for the unknown states. The notation both covers filter estimates in the case where $s \leq t$ and smoothed estimates when $s > t$. $\mathbf{0}_k$ denotes an $k \times k$ matrix of zeros and $\mathbf{I}_k$ denotes the $k \times k$ identity matrix.

3. Discrete hazard model

I will start by introducing the discrete time model for survival analysis. Outcomes in the model are binary as the model in Equation 3. Either an individual has an event or not within each interval. I generalize in Section 5 to a continuous time model. We are observing individual $1, 2, \ldots, n$ who each has an *event* at time $T_1, T_2, \ldots, T_n$. We define the *left-truncation and right-censoring indicators* $D_{i1}, D_{i2}, \ldots, D_{id}$ with $D_{it} \in \{0, 1\}$. The indicator is one if the individual is either left-truncated or right-censored. By definition I set $D_{ik} = 1$ for $k > t$ if we observe an event for individual i at time t. I define the following series of outcome indicators for each individual

$$Y_{it} = 1_{\{T_i \in (t-1,t]\}} = \begin{cases} 1 & \text{if } T_i \in (t-1, t] \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

y_{it} denotes whether individual i experiences an event in interval $(t-1, t]$. We observe covariate vector $\boldsymbol{x}_{ij}$ for each individual i if $D_{ij} = 0$ where the latter subscripts correspond to the interval number. Next, the *risk set* in time interval t is given by

$$R_t = \{i \in \{1, \ldots, n\} : D_{it} = 0\}$$

I will refer to this as the *discrete time risk set*, as I will introduce a continuous time version later. The probability of an event for a given individual i in interval t conditional on $\boldsymbol{\alpha}_t$ is given by

$$\mathsf{P}\left(Y_{it} = 1 \mid \boldsymbol{\alpha}_t, T_i > t - 1\right) = h(\boldsymbol{\alpha}_t^\top \boldsymbol{x}_{it})$$

The inverse logit function, $h(\eta) = \exp(\eta)/(1 + \exp(\eta))$, is used by default. The model written in the state space form is

$$\begin{aligned} \mathsf{E}\left(\boldsymbol{Y}_t \mid \boldsymbol{\alpha}_t\right) &= \boldsymbol{z}_t(\boldsymbol{\alpha}_t) \\ \boldsymbol{\alpha}_{t+1} &= \mathbf{F}\boldsymbol{\alpha}_t + \mathbf{R}\boldsymbol{\eta}_t \qquad \boldsymbol{\eta}_t \sim N(\mathbf{0}, \mathbf{Q}), \qquad t = 1, \ldots, d \\ & \qquad\qquad \boldsymbol{\alpha}_0 \sim N(\boldsymbol{\mu}_0, \mathbf{Q}_0) \end{aligned} \tag{7}$$

where $\boldsymbol{Y}_t = (Y_{it})_{i \in R_t}$. Notice that the bold '$\mathbf{R}$' is a system matrix, whereas the italic 'R_t' is a risk set. The equation for $\boldsymbol{Y}_t$ is referred to as the *observational equation*. The equation for the state vector, $\boldsymbol{\alpha}_t$, is referred to as the *state equation*. The conditional mean outcomes, $\boldsymbol{z}_t(\boldsymbol{\alpha}_t)$, and the covariance matrix, $\mathbf{H}(\boldsymbol{\alpha}_t)$, are

$$\begin{aligned} z_{kt}(\boldsymbol{\alpha}_t) &= \mathsf{E}\left(Y_{i_{kt}t} \mid \boldsymbol{\alpha}_t\right) = h(\boldsymbol{\alpha}_t^\top \boldsymbol{x}_{i_{kt}t}) \\ H_{kk't}(\boldsymbol{\alpha}_t) &= \begin{cases} z_{i_{kt}t}(\boldsymbol{\alpha}_t)(1 - z_{i_{kt}t}(\boldsymbol{\alpha}_t)) & k = k' \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{8}$$

where $R_t = \{i_{1t}, \ldots, i_{n_t t}\}$. The state equation is implemented with a first and second order random walk. The first order random walk model has $\mathbf{F} = \mathbf{R} = \mathbf{I}_m$. Thus, $q = m$ for the first order random walk model. For the second order random walk model, we have

$$\mathbf{F} = \begin{pmatrix} 2\mathbf{I}_m & -\mathbf{I}_m \\ \mathbf{I}_m & \mathbf{0}_m \end{pmatrix}, \qquad \mathbf{R} = \begin{pmatrix} \mathbf{I}_m \\ \mathbf{0}_m \end{pmatrix} \tag{9}$$

That is, we have taken the difference twice. To see this, let $\boldsymbol{\alpha}_t = (\boldsymbol{\xi}_t^\top, \boldsymbol{\xi}_{t-1}^\top)^\top$. Then Equation 9 implies that $\boldsymbol{\xi}_t - 2\boldsymbol{\xi}_{t-1} + \boldsymbol{\xi}_{t-2} = \boldsymbol{\epsilon}_t$ which states that second-order difference are independent normally distributed. Further, we replace the linear predictor, $\boldsymbol{\alpha}_t^\top \boldsymbol{x}_{i_{kt}t}$, in Equation 8 with

$$z_{kt}(\boldsymbol{\alpha}_t) = h(\boldsymbol{\xi}_t^\top \boldsymbol{x}_{i_{kt}t})$$

Notice that the dimension of the state vector is $q = 2m$, which affects the computational complexity. The complete data likelihood of the model can be written as follows by an application of the Markov property of the model

$$L(\mathbf{Q}, \mathbf{Q}_0, \boldsymbol{\mu}_0) = p\left(\boldsymbol{\alpha}_0\right) \prod_{t=1}^{d} p\left(\boldsymbol{\alpha}_t \mid \boldsymbol{\alpha}_{t-1}\right) \prod_{i \in R_t} p\left(y_{it} \mid \boldsymbol{\alpha}_t\right)$$

Thus, the log-likelihood (... depends on an omitted normalization constant) is

$$\begin{aligned} \log L\left(\mathbf{Q}, \mathbf{Q}_0, \boldsymbol{\mu}_0\right) = &-\frac{1}{2}\left(\boldsymbol{\alpha}_0 - \boldsymbol{\mu}_0\right)^\top \mathbf{Q}_0^{-1}\left(\boldsymbol{\alpha}_0 - \boldsymbol{\mu}_0\right) \\ &-\frac{1}{2}\sum_{t=1}^{d}\left(\boldsymbol{\alpha}_t - \mathbf{F}\boldsymbol{\alpha}_{t-1}\right)^\top \mathbf{R}^\top \mathbf{Q}^{-1} \mathbf{R}\left(\boldsymbol{\alpha}_t - \mathbf{F}\boldsymbol{\alpha}_{t-1}\right) \\ &-\frac{1}{2}\log|\mathbf{Q}_0| - \frac{1}{2d}\log|\mathbf{Q}| \\ &+\sum_{t=1}^{d}\sum_{i \in R_t} l_{it}(\boldsymbol{\alpha}_t) + \dots \end{aligned} \tag{10}$$

$$l_{it}(\boldsymbol{\alpha}_t) = y_{it} \log h(\boldsymbol{x}_{it}^\top \boldsymbol{\alpha}_t) + (1 - y_{it}) \log\left(1 - h(\boldsymbol{x}_{it}^\top \boldsymbol{\alpha}_t)\right) \tag{11}$$

This completes the introduction of the discrete time model. I continue with the methods used to fit the model. In the rest of the paper, all comments about computational costs are assuming that the number of observations, n, is much greater than the number of coefficients, q.

4. Methods

I will focus on the `ddhazard` function throughout this article. All the methods that are available with this function use the M-step and parts of the E-step of the EM algorithm described in Fahrmeir (1992, 1994). Moreover, the method is exactly as in the former mentioned papers when the extended Kalman filter with one iteration (which I introduce later) is used. The EM algorithm is similar to the method in Shumway and Stoffer (1982) but with a nonlinear observational equation. The unknown hyperparameters in Equation 7 are the covariance matrices $\mathbf{Q}$ and $\mathbf{Q}_0$ and the initial state mean $\boldsymbol{\mu}_0$. $\mathbf{Q}$ and $\boldsymbol{\mu}_0$ will be estimated in the M-step of the EM algorithm. It is common practice with Kalman filters to set the diagonal elements of $\mathbf{Q}_0$ to large fixed values such that one term is removed from Equation 10.

The EM algorithm is shown in Algorithm 1. The matrices $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_d$ are the design matrices given by the risk sets $R_1, R_2, \dots, R_d$ and the covariate vectors. The only unspecified part is the filter in Line 4 of Algorithm 1. Notice that the other lines involve only products of matrices and vectors of dimension equal to the state space vector's dimension, q. Moreover, the computational cost is independent of the size of the risk sets for the specified parts of Algorithm 1. Thus, the computational complexity so far is $\mathcal{O}\left(q^3 d\right)$. The threshold for convergence is determined by the `eps` argument of the `ddhazard_control` functions which is passed as the `control` argument to `ddhazard` (e.g., `ddhazard_control(eps = 0.001, ...)`) similar to the `glm` function. The EM algorithm tends to converge slowly toward the end. The filters implemented for Line 4 of Algorithm 1 are an extended Kalman filter (EKF), an unscented Kalman filter (UKF), a sequential mode approximation (SMA), and a

Algorithm 1 EM algorithm with unspecified filter. $\|\cdot\|_2$ is the L2 norm.

Input:

$\mathbf{Q}, \mathbf{Q}_0, \boldsymbol{\mu}_0, \mathbf{X}_1, \dots, \mathbf{X}_d, \boldsymbol{y}_1, \dots, \boldsymbol{y}_d, R_1, \dots, R_d$

Convergence threshold ϵ

1: Set $\boldsymbol{a}_{0|0}^{(0)} = \boldsymbol{\mu}_0$ and $\mathbf{Q}^{(0)} = \mathbf{Q}$

2: **for** $k = 1, 2, \dots$ **do**

3: **procedure** E-STEP

4: Apply filter with $\boldsymbol{a}_{0|0}^{(k-1)}$, $\mathbf{Q}^{(k-1)}$ and $\mathbf{Q}_0$ to get
$\boldsymbol{a}_{1|0}$, $\boldsymbol{a}_{1|1}$, $\boldsymbol{a}_{2|1}, \dots, \boldsymbol{a}_{d|d-1}$, $\boldsymbol{a}_{d|d}$ and
$\mathbf{V}_{1|0}$, $\mathbf{V}_{1|1}$, $\mathbf{V}_{2|1}, \dots, \mathbf{V}_{d|d-1}$, $\mathbf{V}_{d|d}$

Apply smoother by computing

5: **for** $t = d, d-1, \dots, 1$ **do**

6: $\mathbf{B}_t^{(k)} = \mathbf{V}_{t-1|t-1}\mathbf{F}\mathbf{V}_{t|t-1}^{-1}$

7: $\boldsymbol{a}_{t-1|d}^{(k)} = \boldsymbol{a}_{t-1|t-1} + \mathbf{B}_t^{(k)}(\boldsymbol{a}_{t|d}^{(k)} - \boldsymbol{a}_{t|t-1})$

8: $\mathbf{V}_{t-1|d}^{(k)} = \mathbf{V}_{t-1|t-1} + \mathbf{B}_t^{(k)}(\mathbf{V}_{t|d}^{(k)} - \mathbf{V}_{t|t-1})\left(\mathbf{B}_t^{(k)}\right)^\top$

9: **procedure** M-STEP

Update the initial state and the covariance matrix by

10: $\boldsymbol{a}_{0|0}^{(k)} = \boldsymbol{a}_{0|d}^{(k)}$

11:
$$\mathbf{Q}^{(k)} = \frac{1}{d}\sum_{t=1}^{d}\mathbf{R}^\top\left(\left(\boldsymbol{a}_{t|d}^{(k)} - \mathbf{F}\boldsymbol{a}_{t-1|d}^{(k)}\right)\left(\boldsymbol{a}_{t|d}^{(k)} - \mathbf{F}\boldsymbol{a}_{t-1|d}^{(k)}\right)^\top + \mathbf{V}_{t|d}^{(k)} - \mathbf{F}\mathbf{B}_t^{(k)}\mathbf{V}_{t|d}^{(k)} - \left(\mathbf{F}\mathbf{B}_t^{(k)}\mathbf{V}_{t|d}^{(k)}\right)^\top + \mathbf{F}\mathbf{V}_{t-1|d}^{(k)}\mathbf{F}^\top\right)\mathbf{R}$$

Stop the if sum of relative norm of changes is below the threshold

12: $\sum_{t=0}^{d}\frac{\left\|\boldsymbol{a}_{t|d}^{(k)} - \boldsymbol{a}_{t|d}^{(k-1)}\right\|_2}{\left\|\boldsymbol{a}_{t|d}^{(k-1)}\right\|_2} < \epsilon$

global mode approximation (GMA). I will cover these in their respective order. First, I will briefly cover the Kalman filter and use the Kalman filter to illustrate why we need to use approximations in the filters. Then, I will give a brief overview of all the methods before covering each method in more detail.

The Kalman filter can be applied in Line 4 of Algorithm 1 when the observed outcomes, $\boldsymbol{y}_t$, in Equation 7 are normally distributed conditional on the state vector and depend linearly on the state vector. The Kalman filter is a two-step recursive algorithm. The first step in the Kalman Filter is the *prediction step* where we estimate $\boldsymbol{a}_{t|t-1}$ and $\mathbf{V}_{t|t-1}$ based on $\boldsymbol{a}_{t-1|t-1}$ and $\mathbf{V}_{t-1|t-1}$. Secondly, we carry out the *correction step* where we estimate $\boldsymbol{a}_{t|t}$ and $\mathbf{V}_{t|t}$ based on $\boldsymbol{a}_{t|t-1}$ and $\mathbf{V}_{t|t-1}$ and the observations. We repeat the process until $t = d$. One advantage with Kalman filter is that both steps can be solved analytically. However, there is no analytical solution for the models covered in this paper. While the prediction step can be solved analytically as the state model is linear and Gaussian, the correction step cannot because of the non-Gaussian distribution of the outcomes given the state vector. Thus, an approximation is needed. The `ddhazard` function provides four fast approximate filters which I will illustrate how to use with a hard disk failure data set.

	EKF (single iteration)	UKF	SMA	GMA
Approximation in correction step	Taylor	UT	Mode	Mode
Parallel	Yes	No	No	Yes
Depends on ordering	No	No	Yes	No
Additional hyperparameters	No	Yes	No	No
Sensitive to $\mathbf{Q}_0$	No	Yes	No	Yes

Table 1: Properties for the filter methods for Line 4 of Algorithm 1. UT stands for unscented transform. The parallel row indicates whether the current implementation supports parallel computation. The "Depends on ordering" row indicates whether the method is sensitive to the ordering of the data set. The "additional hyperparameters" indicates whether there are additional important hyperparameters with the method. The final row indicates whether the method often perform poorly if $\mathbf{Q}_0$ has large entries in the diagonal elements.

Table 1 shows the pros and cons of the methods. We make a Taylor expansion in the EKF given the $\boldsymbol{a}_{t|t-1}$ estimate from the prediction step. The UKF uses the so-called unscented transformation instead. This may yield a better approximation than the Taylor expansion. The SMA approximates the mode of $\boldsymbol{\alpha}_t$ given $\boldsymbol{a}_{t|t-1}$ and $\mathbf{V}_{t|t-1}$ adding the information of each observed outcome, $y_{i_1 t}, \dots, y_{i_{n_t} t}$, in terms. The GMA does the same but uses all the observed outcomes, $\boldsymbol{y}_t$, at the same time. The UKF is currently not supporting computation in parallel but could potentially. The simulation examples in Section 6 suggest that the EKF with multiple iterations and the GMA may be preferable. The methods will be covered in more detail in the following sections including examples of how to use with the **dynamichazard** package.

Example data set

I will use time until failure for hard disks as an example throughout this paper. Predicting when a hard disk will fail is important for any firm that manages large amounts of data stored locally to replace the hard disks before they fail. Self-monitoring, analysis, and reporting technology (SMART) is one tool used to predict future hard disk failures. The data set I will use is publicly available from BackBlaze (2017), which is a data storage provider that currently manages more than 65000 hard disks. Backblaze has a daily snapshot of the SMART attributes for all its hard disks going back to April 2013. The final data set is included with the package and has the name `"hds"`. Some minor changes[1] are made in this paper to the `"hds"` data set. The final data set I use has 79668 unique hard disks. It has 522041 rows in start-stop format for survival analysis.

A hard disk is marked as a failure if "... the drive will not spin up or connect to the OS, the drive will not sync, or stay synced, in a RAID Array ... [or] the Smart Stats we [Backblaze] use show values above our [Backblaze's] thresholds" (Klein 2016). A hard drive with a failure is removed. I will not use the SMART attributes that Backblaze uses as covariates because of the third condition. These are SMART attributes 5, 187, 188, 197, and 198 (BackBlaze 2014).

I will use the power-on hours (SMART attribute number 9) as the time variable in the

[1] I use last observation carried forward for the covariate, change the time scale to months, and I set time zero to 4 days of running.

	$t \in (0, 20]$		$t \in (20, 40]$		$t \in (40, 60]$	
Hard disk version	#D	#F	#D	#F	#D	#F
ST4000DM000	36131	1036	12081	472	31	1
HMS5C4040BLE640	8511	34	3091	2	0	
HMS5C4040ALE640	7155	78	7077	11	44	
ST8000DM002	3927	13	0		0	
HDS5C3030ALA630	3864	19	4625	47	4532	52
HDS5C4040ALE630	2717	40	2665	33	2364	3
ST6000DX000	1915	35	45	1	0	
WD30EFRX	1284	126	876	22	129	1
ST500LM012 HN	800	24	147	2	0	
HDS723030ALA640	792	6	1040	30	997	23
WD60EFRX	495	36	253	12	0	
WD30EZRX	483	6	370	10	0	
ST31500541AS	150	14	712	49	1986	235
HDS722020ALA330	134	7	4765	46	4658	138
ST31500341AS	114	16	345	23	669	114
WD10EADS	38	2	124	8	463	16

Table 2: Summary information for each of the hard disk versions. The hard disk version is indicated by the first column. The number of disks is abbreviated as "#D" and total failures is abbreviated as "#F". The $t \in (x, y]$ indicates which time interval the figures apply to. Blank cells indicate zeros.

model I estimate. The hard disks run 24 hours a day unless they are shut down (e.g., for maintenance). Thus, the power on hours reflects both the usage and age of the hard disk. The SMART attribute I will use as a predictor is the power cycle count (SMART attribute number 12). This counts the number of times a hard disk has undergone a full hard disk power on/off cycle. The power cycle count may be a proxy of batch effects as hard disks are stored in storage pods with 45 or 60 hard disks. An entire storage pod has to be shut down if a hard disk has to be replaced. Thus, the power cycle count may be a proxy for batch effects if hard disks from the same batch are stored in the same storage pods.

I will include a factor level for the hard disk version,[2] as the differences in failure rates between hard disk versions are large. In particular, one 3 terabyte (TB) Seagate hard disk version (ST3000DM001) has a high failure rate (Klein 2015). I remove the 3 TB Seagate hard disks as only half of the hard disk that fails have a failure indicator set to one. I remove versions with fewer than 400 unique hard disks. These have either few cases or few observations. I winsorize at the 0.995 quantile for the power cycle count (i.e., I set values above the 0.995 quantile to the 0.995 quantile).

Table 2 provides information about each of the hard disk versions. The table shows that data is available only for some versions during parts of the 60-month period. Thus, some of the curves shown later will partly be extrapolation.

[2]I write hard disk version instead of model to avoid confusion between a fitted statistical model and a hard disk model.

4.1. Extended Kalman filter

The EKF approximates nonlinear state space models by making a given order Taylor expansion about the state vector, most commonly using the first order Taylor expansion. One of the EKF's advantages is that it results in formulas similar to the Kalman filter. The implemented algorithm is due to Fahrmeir (1992, 1994). The largest computational cost is the Taylor approximation which is $\mathcal{O}\left(q^2 n_t + q^3\right)$. However, the computation is "embarrassingly parallel" because the computation can easily be separated into independent tasks that can be executed in parallel. This is exploited in the current version of `ddhazard` using the **thread** library in C++. The C++ **thread** library is a portable and standardized multithreading library.

Fahrmeir (1992) notes that the EKF is similar to a Newton-Raphson step, which can motivate us to take further steps. This is supported with the `ddhazard` function. Moreover, the EKF may have divergence problems. A learning rate (or step length) can be used in cases where divergence is a problem. Further, `ddhazard` increases the variance in the denominator of the terms used in the algorithm (see the "*ddhazard*" vignette in the package). This reduces the effect of values predicted near the boundaries of the outcome space. This is similar to the approach taken in **glmnet** to deal with large absolute values of the linear predictor.

One option is to take extra correction steps (extra Newton-Raphson steps) which is not done by default. They will be taken if the `NR_eps` argument to `ddhazard_control` is set to the value of convergence threshold in the Newton-Raphson method. The user sets the learning rate (or step length) with the `LR` argument to `ddhazard_control`. By default, the current implementation tries a decreasing series of learning rates, starting with `LR` value until the algorithm does not diverge. The extra term in the denominators, as in the **glmnet** package, are set with the `denom_term` argument to `ddhazard_control`. Values in the range $[10^{-6}, 10^{-4}]$ tend to be sufficient in most cases. My experience is that the user should focus on the learning rate. See the "*ddhazard*" vignette in the package for the algorithm and further details.

I fit the model using the EKF with a single iteration in the correction step. I use a natural cubic spline for the number of power cycles to capture potential nonlinear effects. The code is shown below. First, I assign the `formula` using the `ns` function for a natural cubic spline:

```
R> library("splines")
R> library("dynamichazard")
R> frm <- Surv(tstart, tstop, fails) ~ -1 + model + ns(smart_12,
+    knots = seq(3, 53, 10), Boundary.knots = c(0, 115))
```

I remove the intercept to not get a reference level for the disk version with the `-1`. Then I fit the model:

```
R> system.time(ddfit <- ddhazard(formula = frm, data = hd_dat, by = 1,
+    max_T = 60, id = hd_dat$serial_number, Q_0 = diag(1, 23), Q = diag(
+    0.1, 23), control = ddhazard_control(method = "EKF", eps = 0.001)))

   user  system elapsed
 43.136   0.302  13.102
```

`system.time` is used to show the computation time in seconds. `Q` is the initial value of the covariance matrix $\mathbf{Q}$ and `Q_0` is the covariance matrix $\mathbf{Q}_0$. The `by` argument in the call is used

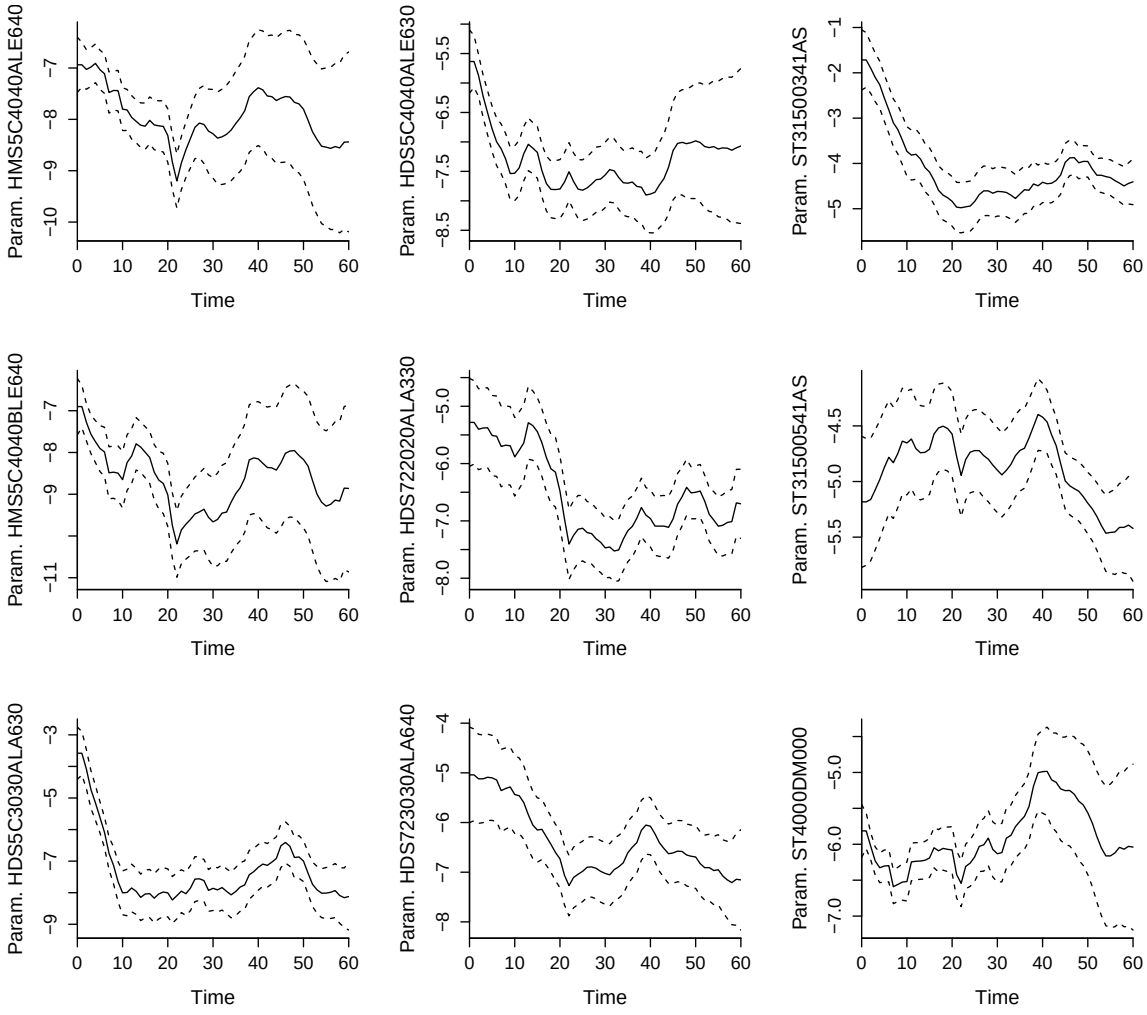


Figure 1: Predicted parameters for factor levels for the hard disk version with EKF with a single iteration in the correction step.

to specify the length of each interval. Thus, `by = 0.5` would give twice as many intervals, each with half the length. The last period we observe when estimating ends at `max_T`. `method = "EKF"` specifies that we want the EKF and `eps` is the convergence threshold used in the EM algorithm.

I will focus on the first nine predicted parameters for the hard disk versions in this paper. Figure 1 shows the predicted parameters of the versions' factor levels. The plot shows the conditional log odds of failing in month t given survival up to time $t-1$ for a hard disk with zero power cycles. It is interesting that some versions seem to have a decreasing parameter for the factor in Figure 1, whereas the parameter increases for others. This can partly be explained by the "bathtub curve" used in reliability engineering. The bathtub curve is a hypothetical hazard curve which has a decreasing hazard rate to start with which is due to defective disks while later having an increasing hazard curve due to wear out. The early failures due to defective disks or later wear out may not be a factor for a particular version

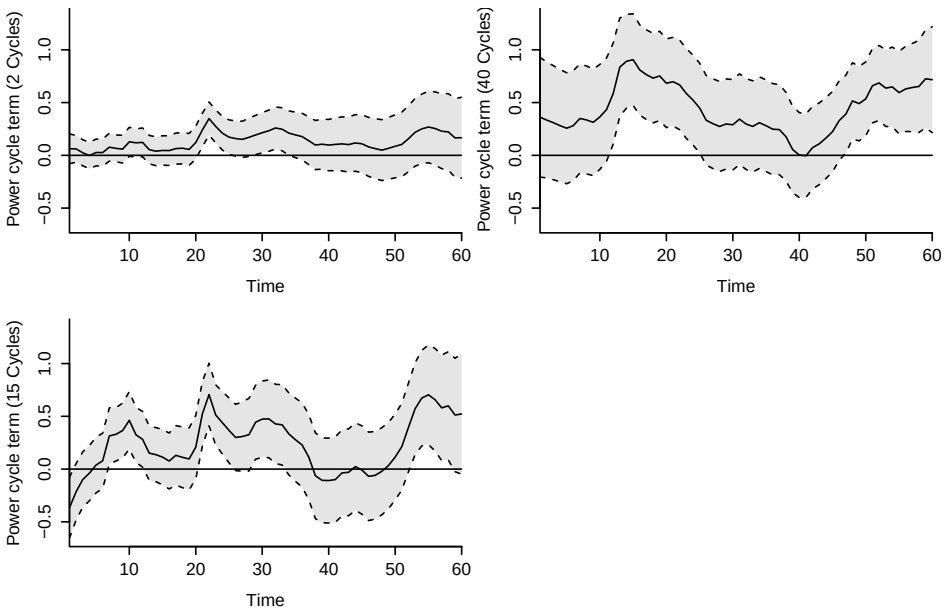


Figure 2: Plots of predicted terms on the linear predictor scale for different values of power cycle counts.

which can explain the curves we see. Notice that some of the prediction intervals get wider or shorter in the start or at the end because of extrapolation. I only have data for at most three years for each hard disk. Furthermore, I only have data for some versions in parts of the 60-month period because of Backblaze's purchasing patterns. Thus, we see increasing or decreasing width of the prediction intervals for some parameters of factor levels in certain periods.

Figure 2 shows how the effect of the number of power cycles evolves over time for three specific choices of the power cycle count. It may seem odd that we do not have a monotone effect as we may expect the batch effect mentioned previously in which case we should see a monotonically increasing effect. However, some of the curves are not based on much data in some parts of the plot. E.g., it is not likely that a hard disk has had many power cycles in the start of the first few months.

Examples with more iterations with the EKF

Next, I fit a model with more iterations in the correction step:

```
R> system.time(ddfit_xtr <- ddhazard(formula = frm, data = hd_dat, by = 1,
+    max_T = 60, id = hd_dat$serial_number, Q_0 = diag(1, 23), Q = diag(0.1,
+    23), control = ddhazard_control(method = "EKF", eps = 0.001,
+    NR_eps = 0.00001)))

   user  system elapsed
137.105   0.966  31.459
```

`NR_eps` is the tolerance for the extra iterations in correction step. The default is `NULL` which

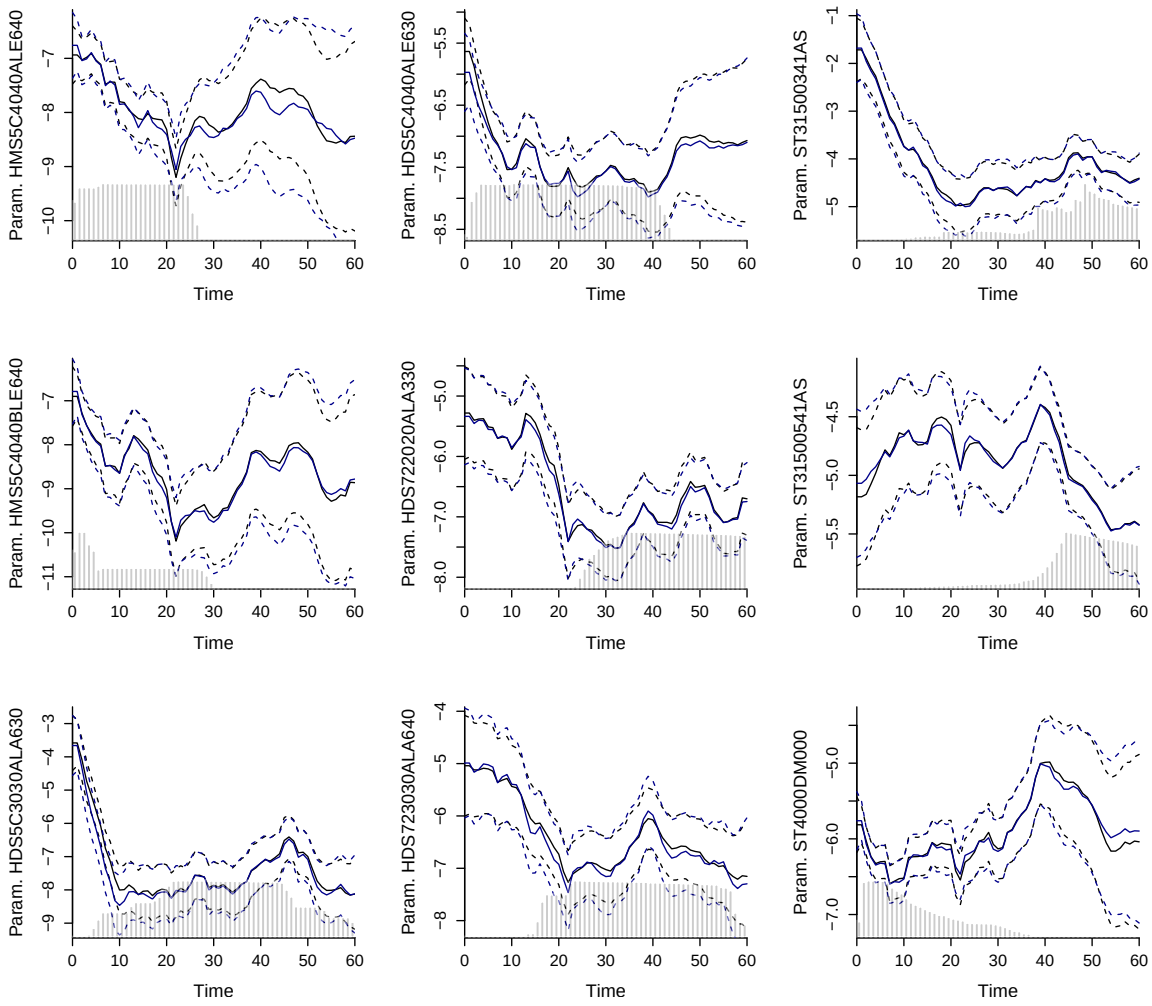


Figure 3: Predicted parameters with and without extra iterations in the correction step with the EKF. The blue curves are the estimate with extra iterations. Grey transparent bars indicate the number of individuals at risk for the specific hard disk version. Heights are only comparable within each frame.

yields only a single iteration. It takes longer due the additional correction steps. I plot the first nine factor levels with the following call:

```
R> for(i in 1:9){
+    plot(ddfit, cov_index = i)
+    plot(ddfit_xtr, cov_index = i, add = TRUE, col = "darkblue")
+    add_hist(i)
+  }
```

Figure 3 shows the plots. The `add_hist` is a function for the specific data set that adds the bars which heights reflect the relative number of individuals at risk in each interval for a given hard disk version. For some of the parameters, the two plots differ noticeably, particularly

the factors levels with a sparse amount of data (observations and/or failures) in some periods (cf., Table 2). A disadvantage of the EKF is that it may provide a poor approximation of the nonlinearities; This is what motivates the unscented Kalman filter in the next section.

4.2. Unscented Kalman filter

The UKF is introduced by Julier and Uhlmann (1997). The idea is select a fixed set of vectors and weights such that the mean and covariance matrix match those of the filtered state vector distribution at the prediction step. The points are then transformed in the correction step and an approximation of the mean and covariance matrix of the the filtered state vector distribution of the next time step is then easily computed. The vectors and weights with the UKF are respectively known as *sigma points* and *sigma weights*. The UKF potentially provides a better approximation of the nonlinear dynamics than does the linear approximation used in the EKF. Further, the UKF does not require computation of the Jacobian. The latter advantage is not as important since deriving and computing the Jacobian is not complicated for the models in this paper.

The unscented transform perform the correction step by evaluating the conditional mean and covariance matrix of $\boldsymbol{y}_t$ at $2q + 1$ weighted points, the so-called *sigma points*, given by:

$$\begin{aligned} \widehat{\boldsymbol{a}}_0 &= \boldsymbol{a}_{t|t-1} \\ \widehat{\boldsymbol{a}}_j &= \boldsymbol{a}_{t|t-1} + \sqrt{q+\lambda}\left(\mathbf{V}_{t|t-1}^{1/2}\right)_j, \qquad j = 1, 2, \dots, q \\ \widehat{\boldsymbol{a}}_{j+q} &= \boldsymbol{a}_{t|t-1} - \sqrt{q+\lambda}\left(\mathbf{V}_{t|t-1}^{1/2}\right)_j \end{aligned} \tag{12}$$

with associated *sigma weights*:

$$\begin{aligned} W_0^{[m]} &= \frac{\lambda}{q+\lambda} \\ W_0^{[c]} &= \frac{\lambda}{q+\lambda} + 1 - \alpha^2 \\ W_j^{[m]} &= W_j^{[c]} = \frac{1}{2(q+\lambda)}, \qquad j = 1, \dots, 2q \end{aligned}$$

where $\lambda = \alpha^2(q + \kappa) - q$, κ and α are hyperparameters, $W_j^{[m]}$ are weights used to compute the conditional mean of $\boldsymbol{y}_t$, and $W_0^{[c]}$ are weights used to compute the conditional covariance matrix of $\boldsymbol{y}_t$. $\mathbf{V}_{t|t-1}^{1/2}$ denotes the "square root" matrix of $\mathbf{V}_{t|t-1}$ and $\left(\mathbf{V}_{t|t-1}^{1/2}\right)_j$ denotes the jth column of the "square root" matrix. We can then evaluate an approximate conditional mean and covariance matrix of $\boldsymbol{y}_t$ by

$$\mathsf{E}\left(\boldsymbol{y}_t \mid \boldsymbol{y}_1, \dots, \boldsymbol{y}_{t-1}\right) \approx \overline{\boldsymbol{y}} = \sum_{j=0}^{2q} W_j^{[m]} \boldsymbol{z}_t\left(\widehat{\boldsymbol{a}}_j\right) \tag{13}$$

$$\mathsf{VAR}\left(\boldsymbol{y}_t \mid \boldsymbol{y}_1, \dots, \boldsymbol{y}_{t-1}\right) \approx \sum_{j=0}^{2q} W_j^{[c]} (\widehat{\boldsymbol{y}}_j - \overline{\boldsymbol{y}})(\widehat{\boldsymbol{y}}_j - \overline{\boldsymbol{y}})^\top + \sum_{j=0}^{2q} W_j^{[c]} \mathbf{H}_t(\widehat{\boldsymbol{a}}_j) \tag{14}$$

`ddhazard` uses the Cholesky decomposition for the $\mathbf{V}_{t|t-1}^{1/2}$ decomposition. The hyperparameters on which the sigma points and sigma weights depend on can have values $0 < \alpha \leq 1$,

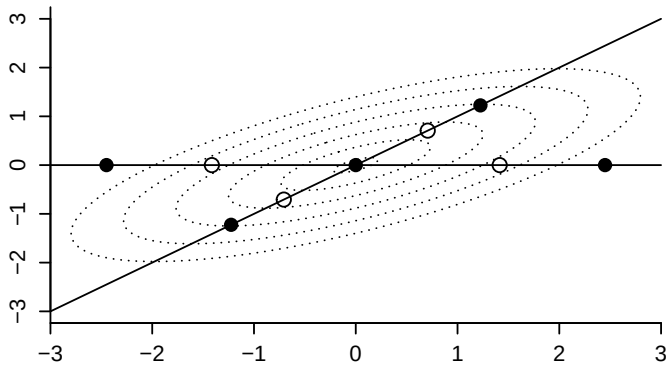


Figure 4: Illustration of sigma points in the example from Equation 15. The dashed lines are the contours of the density given by $\boldsymbol{a}_{t|t-1}$ and $\mathbf{V}_{t|t-1}$. The full lines are the direction given by the columns of the Cholesky decomposition. The filled circles are sigma points with $(\alpha, \kappa) = (1, 1)$ and the open circles are the sigma points with $(\alpha, \kappa) = (1/\sqrt{3}, 1)$. The point at $(0, 0)$ is a sigma point for both sets for hyperparameters.

$\kappa \in \mathbb{R}$ under the restriction that $q + \lambda = \alpha^2(q + \kappa) > 0$. The following example will provide an idea of the effect of the hyperparameters. Suppose that at time t in the correction step of the filter with a two dimensional state equation, $q = 2$, we have

$$\boldsymbol{a}_{t|t-1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \mathbf{V}_{t|t-1} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}, \qquad \mathbf{V}_{t|t-1}^{1/2} = \begin{pmatrix} 1.41 & 0.707 \\ 0 & 0.707 \end{pmatrix} \tag{15}$$

Then the following hyperparameters yield to the following weights

$$\begin{aligned} (\alpha, \kappa) = (1, 1) \quad &\Rightarrow \quad \left(W_0^{[m]}, W_1^{[m]}, \ldots, W_{2q}^{[m]}\right) = (1/3, 1/6, \ldots, 1/6) \\ (\alpha, \kappa) = (1/\sqrt{3}, 1) \quad &\Rightarrow \quad \left(W_0^{[m]}, W_1^{[m]}, \ldots, W_{2q}^{[m]}\right) = (-1, 1/2, \ldots, 1/2) \end{aligned}$$

Decreasing α increases the absolute size of the weights of the last $2q$ sigma points and can lead to a negative weight on the zero sigma point, $\widehat{\boldsymbol{a}}_0$, as it does here. α also controls the spread of the sigma points through a $\alpha\sqrt{q + \kappa}$ factor in Equation 12. Decreasing α decreases the spread of the sigma points, as Figure 4 illustrates. The filled circles are the sigma points with $(\alpha, \kappa) = (1, 1)$, and the open circles are the sigma points with $(\alpha, \kappa) = (1/\sqrt{3}, 1)$.

A negative weight on the zero-th sigma point, $W_0^{[m]} < 0$, can cause computational issues, as Menegaz (2016) points out, since the conditional covariance matrix in Equation 14 can fail to be positive definite. Thus, we may select a specific value of $W_0^{[m]} > 0$ by setting $\kappa = q(1 + \alpha^2(W_0^{[m]} - 1))/(\alpha^2(1 - W_0^{[m]}))$ for a given value of α. The UKF can be tuned more than the EKF to any given data set while it may be hard to make estimation in an automatic fashion with the UKF.

`ddhazard` uses the three hyperparameter UKF given by Wan and Merwe (2000). There is an additional parameter denoted by β which is not included here for the sake of brevity. Many different UKFs have been suggested with different hyperparameters, algorithms, and sigma points (see Menegaz 2016 for a comparison of different forms of UKFs in the literature). Evaluating Equation 14 as in Wan and Merwe (2000) yields an $\mathcal{O}\left(n_t^3\right)$ computational complexity

algorithm. It is reduced to $\mathcal{O}(n_t)$ with an application the Woodbury matrix identity. See the "*ddhazard*" vignette for further details.

Computation in parallel is not supported in the current version of `ddhazard` with the UKF. An identity matrix times a scalar is added to Equation 14 to reduce the effect of observation predicted near the boundary of the outcome space as done with the EKF. The scalar can be set with `denom_term` argument to `ddhazard_control`. I will end this section on the UKF with an example.

Examples with the UKF

One problem with the UKF compared with the EKF is its greater sensitivity to the choice of $\mathbf{Q}_0$ because it is used to form the sigma points at time zero. I will illustrate this in the following paragraphs. I fit the model below and plot the predicted parameters. I set $\mathbf{Q}_0$ to a diagonal matrix but with larger entries than before. I specify that I want the UKF by setting the argument `method = "UKF"` in the `ddhazard_control` call. `eps` is increased such that the methods is deemed to have converged within the default amount of maximum EM iterations.

```
R> system.time(ddfit_ukf <- ddhazard(formula = frm, data = hd_dat, by = 1,
+    max_T = 60, id = hd_dat$serial_number, Q_0 = diag(10, 23), Q = diag(
+    0.1, 23), control = ddhazard_control(method = "UKF", eps = 0.01)))

   user  system elapsed
 62.237   0.166  61.819
```

Figure 5 shows the result. Figure 6 shows the same model but with $\mathbf{Q}_0$'s diagonal entries equal to 0.1. The latter figure is comparable to what we have seen previously. A similar comment applies to the starting value of $\mathbf{Q}$. My experience is that we need to select a matrix that has *large* but not *too large* elements in the diagonal. See Xiong, Zhang, and Chan (2006) for the covariance matrix role in a slightly different class of models. In contrast to the UKF, the EKF with one iteration in the correction step can have large entries in the diagonal of $\mathbf{Q}_0$.

4.3. Sequential approximation of the posterior modes

Another idea is to replace the means in the filters with the modes in each correction step. That is, we are still looking for a method to perform the filtering in Algorithm 1. We perform the same prediction step as with the EKF and UKF, and we change the correction step from finding the mean to finding the mode. In making this replacement, we must find the minimum of Equation 16, followed by an update of the covariance matrix.

$$\boldsymbol{a}_{t|t} = \arg\min_{\boldsymbol{\alpha}} \left(-\log p\left(\boldsymbol{\alpha} \mid \boldsymbol{a}_{t|t-1}, \mathbf{V}_{t|t-1}\right) - \sum_{i \in R_t} \log p\left(y_{it} \mid \boldsymbol{\alpha}\right) \right) \tag{16}$$

One way of finding an approximate minimum is to replace Equation 16 with n_t rank-one updates of the form in Equation 17 and an update of the covariance matrix. I use a superscript to indicate the previous result from the rank-one update.

$$\boldsymbol{a}_{t|t}^{(k)} = \arg\min_{\boldsymbol{\alpha}} \left(-\log p\left(\boldsymbol{\alpha} \mid \boldsymbol{a}_{t|t}^{(k-1)}, \mathbf{V}_{t|t}^{(k-1)}\right) - \log p\left(y_{it} \mid \boldsymbol{\alpha}\right) \right) \tag{17}$$

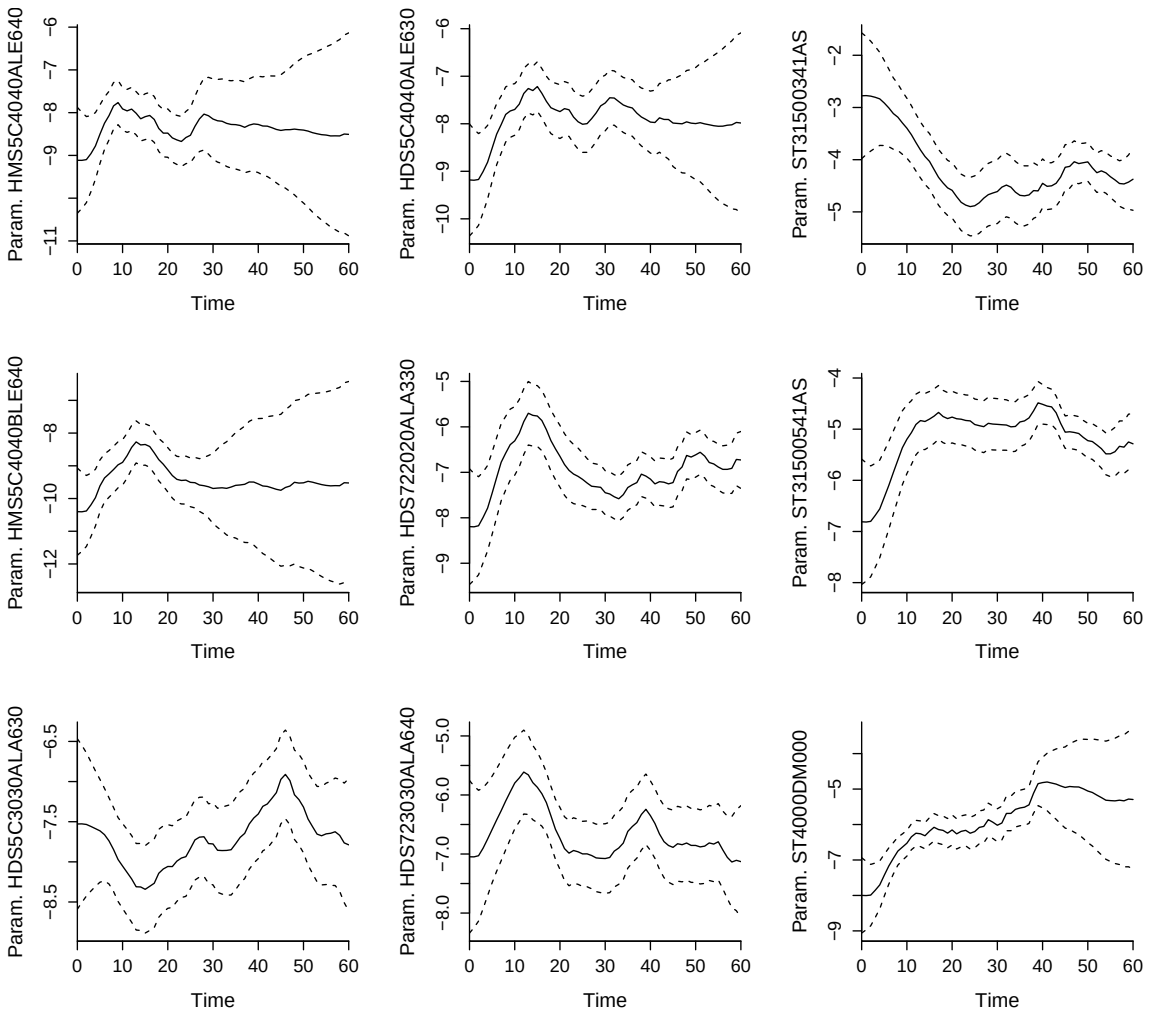


Figure 5: Predicted parameters with the UKF used on the hard disk failure dataset where $\mathbf{Q}_0$ has large entries in the diagonal.

I will refer to this method as the SMA. There are two implemented versions: one which makes the updates of the covariance matrix using the Woodbury matrix identity and one which updates a Cholesky decomposition of the concentration matrix instead. The latter guarantees that the covariance matrix is positive semi-definite but is slower. See the "*ddhazard*" vignette for further details.

The SMA can have large entries in the diagonal of $\mathbf{Q}_0$ like the EKF with one iteration. A disadvantage of SMA is that it is sequential and all matrix and vector products are in dimension $q \times q$ and q. Thus, although one could do the matrix operations in parallel, computation in parallel of the matrix operations is only advantageous if q is large. Moreover, the result depends on the order of the risk set. For this reason, the risk sets are permuted once before running the algorithm. This can be avoided by setting passing `permu = FALSE` in the `ddhazard_control` call.

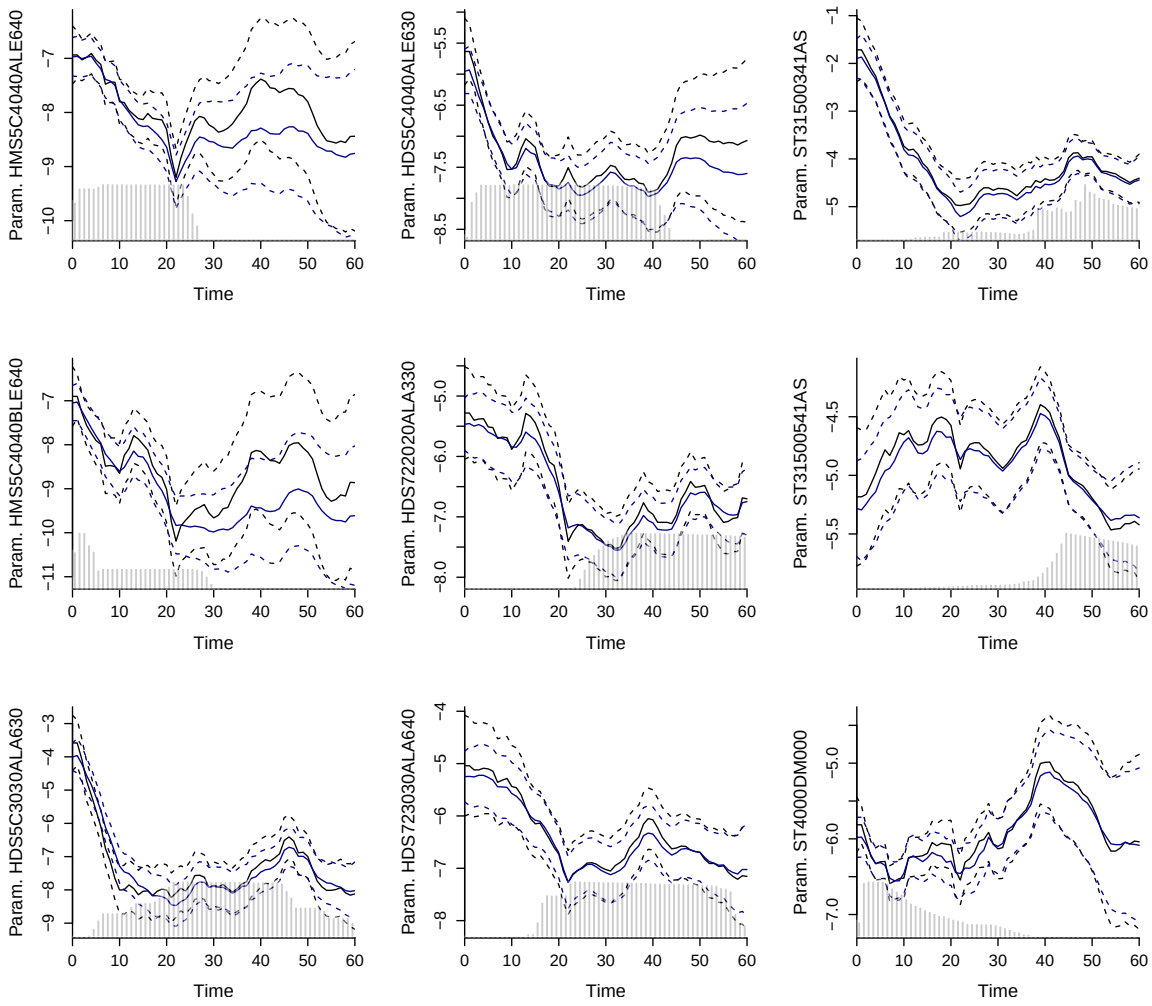


Figure 6: Similar plot to Figure 5 but where the diagonal entries of $\mathbf{Q}_0$ are 0.1. The black curve is the estimates from the EKF with one iteration in the correction step. Grey transparent bars indicate the number of individuals at risk for the specific hard disk version. Heights are only comparable within each frame.

4.4. Global mode approximation

We can also minimize the right-hand side of Equation 16 directly. This will be called the GMA method. It is equivalent to an L2 penalized generalized linear model (GLM) in every iteration. It can be shown that the EKF with more iterations solves an equivalent problem as a Newton-Raphson to minimize the right-hand side of Equation 16. Thus, details on the GMA is omitted here and can be found in the "*ddhazard*" vignette.

The GMA is sensitive to the choice of $\mathbf{Q}_0$ which works as an inverse penalty. To give an extreme example, suppose we have no events in the first interval and only an intercept. Setting $\mathbf{Q}_0$ to a diagonal matrix with large entries (in this case $\mathbf{Q}_0$ is a scalar) implies almost no restrictions on the intercept. Thus, selecting an intercept tending towards minus infinity is optimal. The computation with the GMA is done in parallel with **OpenMP** (**OpenMP** Architecture Review Board 2013).

The global mode approximation and the EKF with more iterations in the correction step are somewhat similar to the method in Durbin and Koopman (2012, Section 10.6). The major difference is that Durbin and Koopman (2012) make the Taylor expansion *before* running the filter about the current estimate of $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_d$ yielding so called pseudo-observations of an approximating Gaussian model. In contrast, the GMA method makes the expansion at each correction step *within* the filter about the current estimate of $\boldsymbol{a}_{t|t-1}$. The **KFAS** package implements the method in Durbin and Koopman (2012, Section 10.6). Further, **KFAS** uses the sequential method described in Koopman and Durbin (2000). The sequential method in **KFAS** is somewhat like the SMA. Again, the difference is at what point the Taylor expansion is made.

Examples with the SMA and GMA

I will use the hard disk failures data set to compare the SMA and GMA methods with the EKF with a single iteration in the correction step. Below, I estimate the model with the SMA method, and the GMA method. I use the correction step with the Cholesky decomposition with the SMA by setting the argument `posterior_version = "cholesky"` in the `ddhazard_control` call.

```
R> system.time(ddfit_SMA <- ddhazard(formula = frm, data = hd_dat, by = 1,
+    max_T = 60, id = hd_dat$serial_number, Q_0 = diag(1, 23), Q = diag(0.1,
+    23), control = ddhazard_control(eps = 0.001, method = "SMA",
+    posterior_version = "cholesky")))

   user  system elapsed
415.938   0.171 415.513

R> system.time(ddfit_GMA <- ddhazard(formula = frm, data = hd_dat, by = 1,
+    max_T = 60, id = hd_dat$serial_number, Q_0 = diag(1, 23), Q = diag(0.1,
+    23), control = ddhazard_control(eps = 0.001, method = "GMA")))

   user  system elapsed
216.048   0.197  38.697
```

Figure 7 shows the three sets of predicted parameters. The parameters in Figure 7 appear similar. It is clear from the above that the SMA method using the Cholesky decompositions is much slower than the GMA. Further, the GMA has a computation time which is close to the EKF with more iterations shown earlier as expected.

4.5. Constant effects

In some applications, constant (time-invariant) parameters may be relevant. A common way of estimating fixed parameters in filtering (e.g., see Harvey and Phillips 1979) is to set the entries of the rows and columns of $\mathbf{Q}$ for the fixed parameters to zero and the corresponding diagonal entries of $\mathbf{Q}_0$ to large values. This approach is also used by Fahrmeir (1992) with the EKF. An alternative method to estimate the effects in the M-step is also included in the package but it is omitted here for brevity.

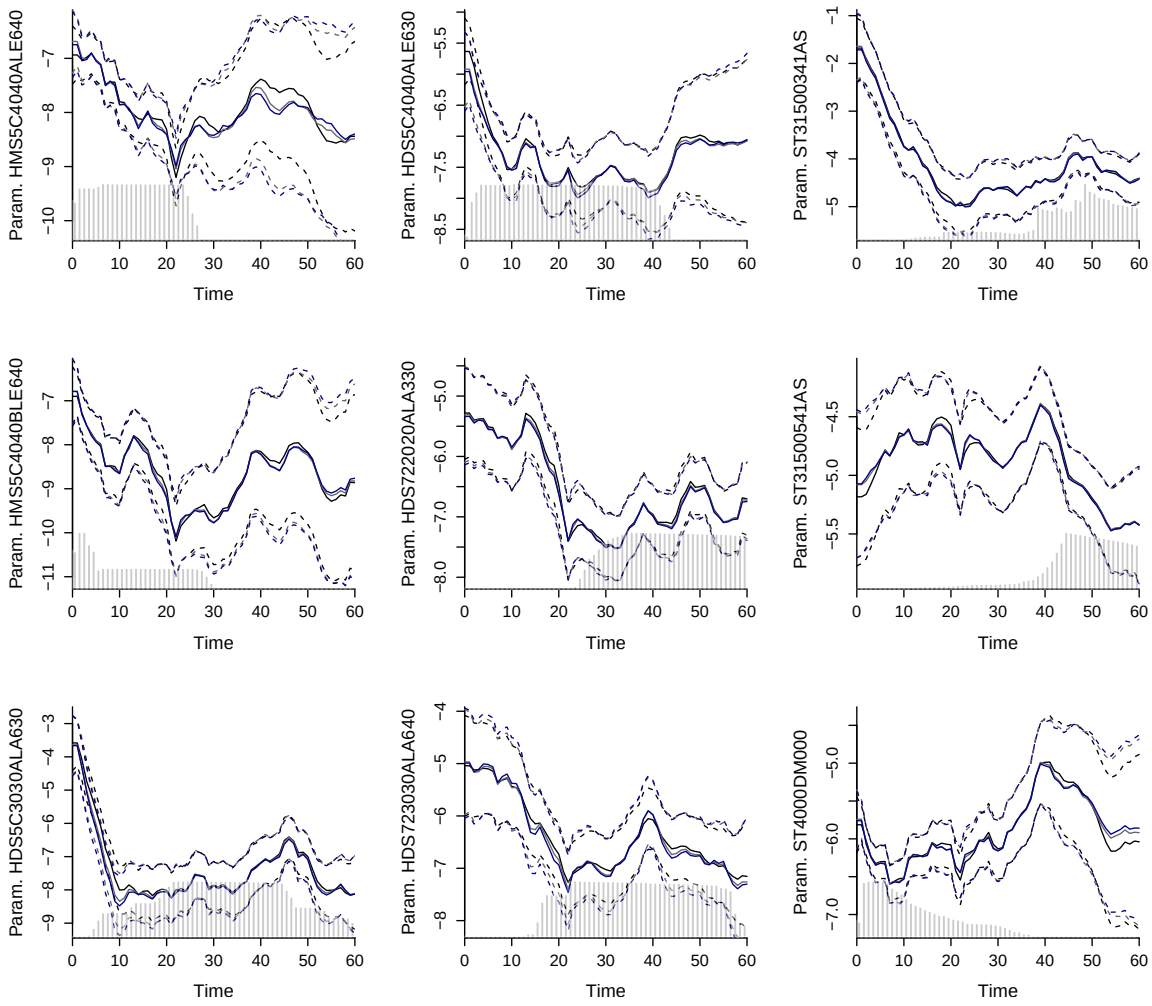


Figure 7: Predicted parameters using the EKF with a single iteration in the correction step, the GMA, and the SMA for the hard disk failure data set. The gray lines are the parameters from the SMA, blue lines are parameters from the GMA, and the black lines are the parameters from the EKF. Grey transparent bars indicate the number of individuals at risk for the specific hard disk version. Heights are only comparable within each frame.

4.6. Second order random walk

I will end this part of the paper by estimating fixed parameters in the E-step as mentioned in the previous section. Further, I will illustrate the use of the second order random walk model. I estimate the model below where the factor levels for the hard disk version follow a second order random walk, and the spline for the SMART 12 attribute is fixed. I specify that I want a second order random walk for the factor levels by setting the argument `order = 2`. I specify which terms are fixed by wrapping the terms in the formula in the `ddFixed` function. The fixed effect estimation method is selected by setting `fixed_terms_method = "E_step"` in the `ddhazard_control` call. To avoid divergence, I decrease the learning rate by setting the `LR` argument the `ddhazard_control` call. Notice that `Q_0`'s dimension is twice that of `Q`.

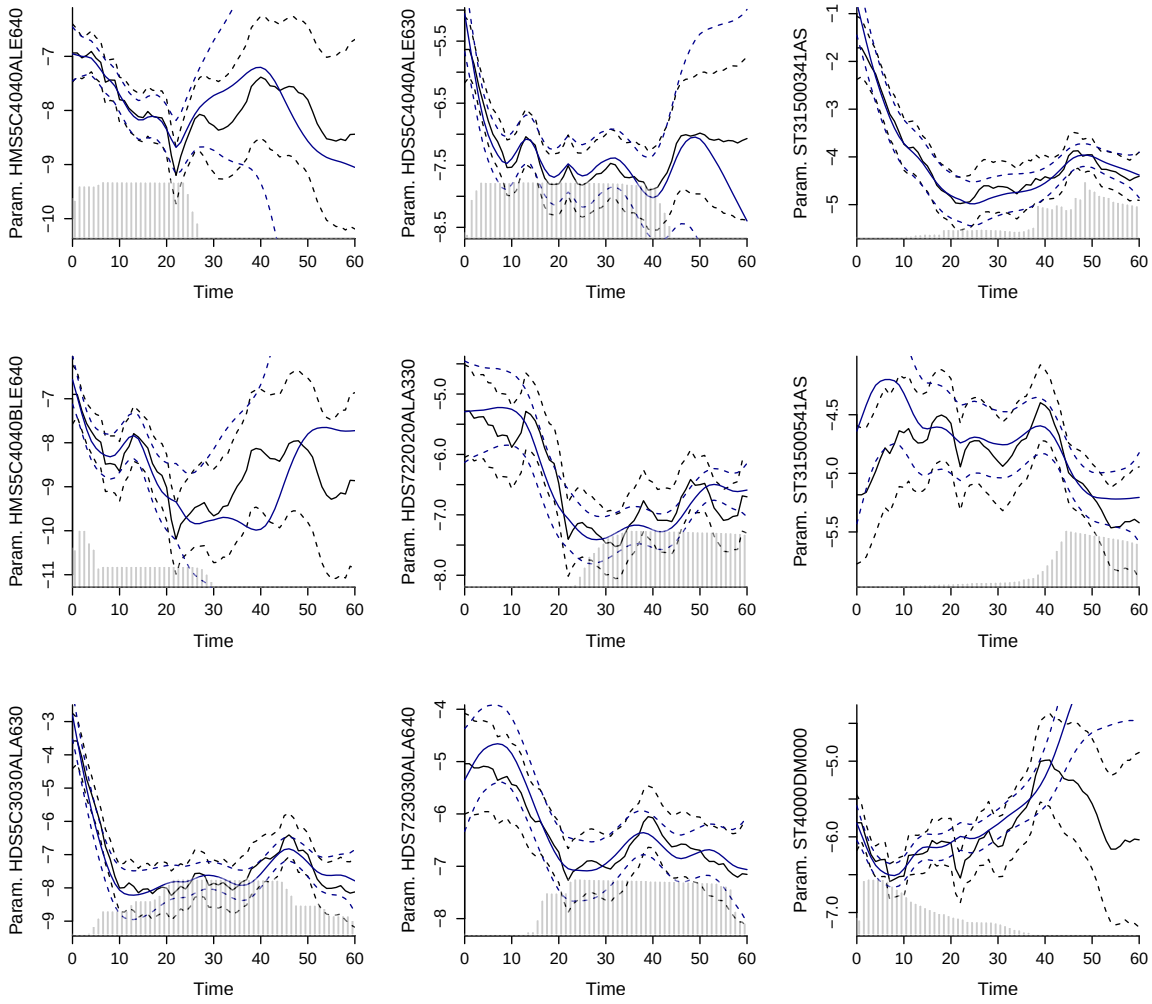


Figure 8: Predicted parameters for some of the factor levels with the second order random walk model. The blue lines are parameters with the second order random walk model with a fixed effect for the power cycle count and the black lines are parameters with the first order random walk model where all parameters are time-varying. Grey transparent bars indicate the number of individuals at risk for the specific hard disk version. Heights are only comparable within each frame.

Lastly, I increase the maximum number of EM-iterations with the `n_max = 250` argument.

```
R> frm_fixed <- Surv(tstart, tstop, fails) ~ -1 + model + ddFixed(ns(
+    smart_12, knots = seq(3, 53, 10), Boundary.knots = c(0, 115)))
R> system.time(ddfit_fixed_E <- ddhazard(formula = frm_fixed,
+    data = hd_dat, by = 1, max_T = 60, order = 2,
+    id = hd_dat$serial_number, Q_0 = diag(1, 32), Q = diag(0.1, 16),
+    control = ddhazard_control(method = "GMA", n_max = 250,
+    NR_eps = 0.00001, eps = 0.001, LR = 0.1, fixed_terms_method = "E_step")))
```

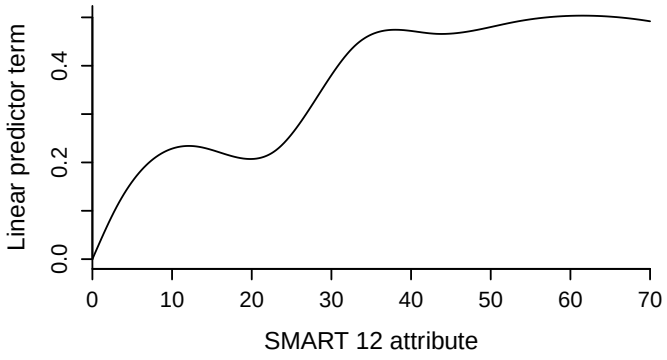


Figure 9: Fixed effects estimates for the SMART 12 attribute on the linear predictor scale.

```
    user   system  elapsed
4091.182    0.714  687.154
```

Figure 8 shows the predicted factor levels for the hard disk version, and Figure 9 shows the spline estimate. The curves are more smooth compared to the first order random walk model as expected. The spline estimate shows a close to monotone increasing effect of the number of power cycle as we may have expected.

5. Discrete versus continuous time

The dynamic discrete time model is where we use the log-likelihood terms, $l_{it}(\boldsymbol{\alpha}_t)$, as shown in Equation 11 where h is the inverse logit function, $h(x) = \exp(x)/(1 + \exp(x))$. This model is suited for situations where the events are observed in intervals and the covariates change at discrete times. However, this is not the case for the hard disk data set. The hard disk data is not reported on monthly precision but on hourly precision. I print the first 10 rows here to illustrate this:

```
R> hd_dat[1:10, c("serial_number", "model", "tstart", "tstop", "smart_12")]
```

```
    serial_number        model tstart  tstop smart_12
505      5XW004AJ ST31500541AS 30.001 40.010        0
506      5XW004AJ ST31500541AS 40.010 43.172       24
507      5XW004AJ ST31500541AS 43.172 56.917       25
508      5XW004Q0 ST31500541AS 40.618 50.962        0
509      5XW004Q0 ST31500541AS 50.962 53.729       54
510      5XW004Q0 ST31500541AS 53.729 54.122       56
511      5XW004Q0 ST31500541AS 54.122 54.424       57
512      5XW004Q0 ST31500541AS 54.424 54.457       58
513      5XW004Q0 ST31500541AS 54.457 54.690       59
514      5XW004Q0 ST31500541AS 54.690 57.193       61
```

I will explain how the `ddhazard` implementation discretizes continuous event times in the following paragraphs. I redefine $\boldsymbol{x}_{ij}$ as the covariate vector for individual i in the period

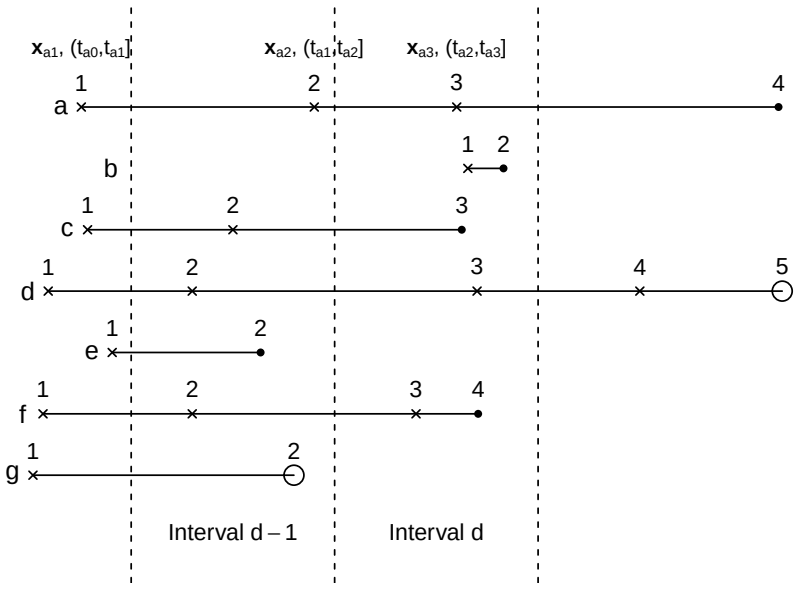


Figure 10: Illustration of a data set with 7 individuals with time-varying covariates. Each horizontal line represents an individual. Each number indicates a start time and stop time in the initial data. A cross indicates that new covariates are observed while a filled circle indicates that the individual has an event. An open circle indicates that the individual is right censored. Vertical dashed lines are time interval borders. The symbols for the covariate vectors and stop times are shown for observation a.

$(t_{i,j-1}, t_{ij}]$. Next, I redefine the discrete time risk set, R_t, as

$$R_t = \{(i,j) : t_{i,j-1} \leq t - 1 < t_{i,j} \wedge D_{ij} = 0\} \tag{18}$$

Further, I redefine

$$y_{ijt} = 1_{\{T_i \in (t-1,t] \wedge t-1 < t_{ij} \leq t\}}$$
$$l_{ijt}(\boldsymbol{\alpha}_t) = y_{ijt} \log h(\boldsymbol{x}_{ij}^\top \boldsymbol{\alpha}_t) + (1 - y_{ijt}) \log \left(1 - h(\boldsymbol{x}_{ij}^\top \boldsymbol{\alpha}_t)\right)$$

y_{ijt} is a generalization of Equation 6 that indicates whether individual i experiences an event with *the jth covariate vector* in interval t. The following example will illustrate the impact of discrete time risk sets in Equation 18. Suppose we look at interval $d-1$ and d (the last two intervals) in a model with time-varying covariates. Further, let both the event times and the point at which we observe new covariates happen at continuous points in time.

Figure 10 illustrates such a situation. Each horizontal line represents an individual. A cross represents when the covariate values jump for the individual, and a filled circle represents an event that has happened for the individual. Lines that end with an open circle are right censored. The vertical dashed lines in the figure represent the time interval borders. The first vertical line from the left is the start of interval $d-1$, the second vertical line is when interval $d-1$ ends, and interval d starts, and the third vertical line is when interval d ends. I will use observation a in Figure 10 to illustrate the risk set in Equation 18. The covariate vector used in interval $d-1$ is $\boldsymbol{x}_{a1}$ as $t_{a0} < d-2 < t_{a1}$. By similar arguments, the covariate vector in interval d is $\boldsymbol{x}_{a2}$.

Because we use the risk set in Equation 18, we use covariates from 1 for individuals a, c, d, and f for the entire period of interval $d-1$, even though the covariates change at 2. Furthermore, g is not in either interval, as we only know that it survives parts of interval $d-1$. Lastly, we never include b as we do not know its covariate vector at the start of interval d.

5.1. Continuous time model

The continuous time model implemented in `ddhazard` is the model shown in Equation 5 in the introduction. The assumptions of the model are:

- The instantaneous hazards are given by $\exp(\boldsymbol{x}_i(t)^\top \boldsymbol{\alpha}(t))$.
- Parameters jump at the end of time intervals, i.e., $\boldsymbol{\alpha}(t) = \boldsymbol{\alpha}_{\lceil t \rceil}$ where $\lceil t \rceil$ gives the ceiling of t. This is illustrated in Figure 10 where the parameters jump at the vertical lines.
- The individuals' covariates jump, i.e., $\boldsymbol{x}_i(t) = \boldsymbol{x}_{ij}$ where $j = \{k : \ t_{i,k-1} < t \leq t_{i,k}\}$. In Figure 10, the covariates jump at the crosses.

The instantaneous hazard jumps when either the individual's covariates jump or the parameters jump. Thus, an individual's event time is piecewise exponentially distributed given the state vectors. The log-likelihood of individual i having an event at time t_i is

$$\log\left(L\left(t_i \mid \boldsymbol{\alpha}_0, \dots, \boldsymbol{\alpha}_d\right)\right) = \boldsymbol{x}_i(t_i)^\top \boldsymbol{\alpha}(t_i) - \int_0^{t_i} \exp\left(\boldsymbol{x}_i(u)^\top \boldsymbol{\alpha}(u)\right) du \tag{19}$$

where $L(\cdot)$ denotes the likelihood. Because of our assumptions, the complete data log-likelihood in Equation 10 simplifies to

$$\begin{aligned}
\mathcal{L}\left(\mathbf{Q}, \mathbf{Q}_0, \boldsymbol{\mu}_0\right) = &-\frac{1}{2}\left(\boldsymbol{\alpha}_0 - \boldsymbol{\mu}_0\right)^\top \mathbf{Q}_0^{-1}\left(\boldsymbol{\alpha}_0 - \boldsymbol{\mu}_0\right) \\
&-\frac{1}{2}\sum_{t=1}^{d}\left(\boldsymbol{\alpha}_t - \mathbf{F}\boldsymbol{\alpha}_{t-1}\right)^\top \mathbf{R}^\top \mathbf{Q}^{-1} \mathbf{R}\left(\boldsymbol{\alpha}_t - \mathbf{F}\boldsymbol{\alpha}_{t-1}\right) \\
&-\frac{1}{2}\log|\mathbf{Q}_0| - \log\frac{d}{2}|\mathbf{Q}| \\
&+\sum_{t=1}^{d}\sum_{(i,j)\in\mathcal{R}_t} l_{ijt}(\boldsymbol{\alpha}_t) + \dots
\end{aligned}$$

where

$$l_{ijt}(\boldsymbol{\alpha}_t) = y_{ijt}\boldsymbol{x}_{ij}^\top \boldsymbol{\alpha}_t - \exp\left(\boldsymbol{x}_{ij}^\top \boldsymbol{\alpha}_t\right)\left(\min\{t, t_{ij}\} - \max\{t-1, t_{i,j-1}\}\right)$$

The l_{ijt} terms are a simplification of Equation 19, where I use the assumption that both the covariates, $\boldsymbol{x}_i(t)$, and parameters, $\boldsymbol{\alpha}(t)$, are piecewise constant. Further, $\mathcal{R}_t$ is the *continuous time risk set* given by

$$\mathcal{R}_t = \{(i,j) : \ t_{i,j-1} < t \wedge t_{ij} \geq t-1 \wedge D_{ij} = 0\}$$

The two conditions in $\mathcal{R}_t$ are that the observation must start before the interval ends ($t_{i,j-1} < t$), and end after the interval starts ($t_{ij} \geq t-1$). I will use observation a in Figure 10 as

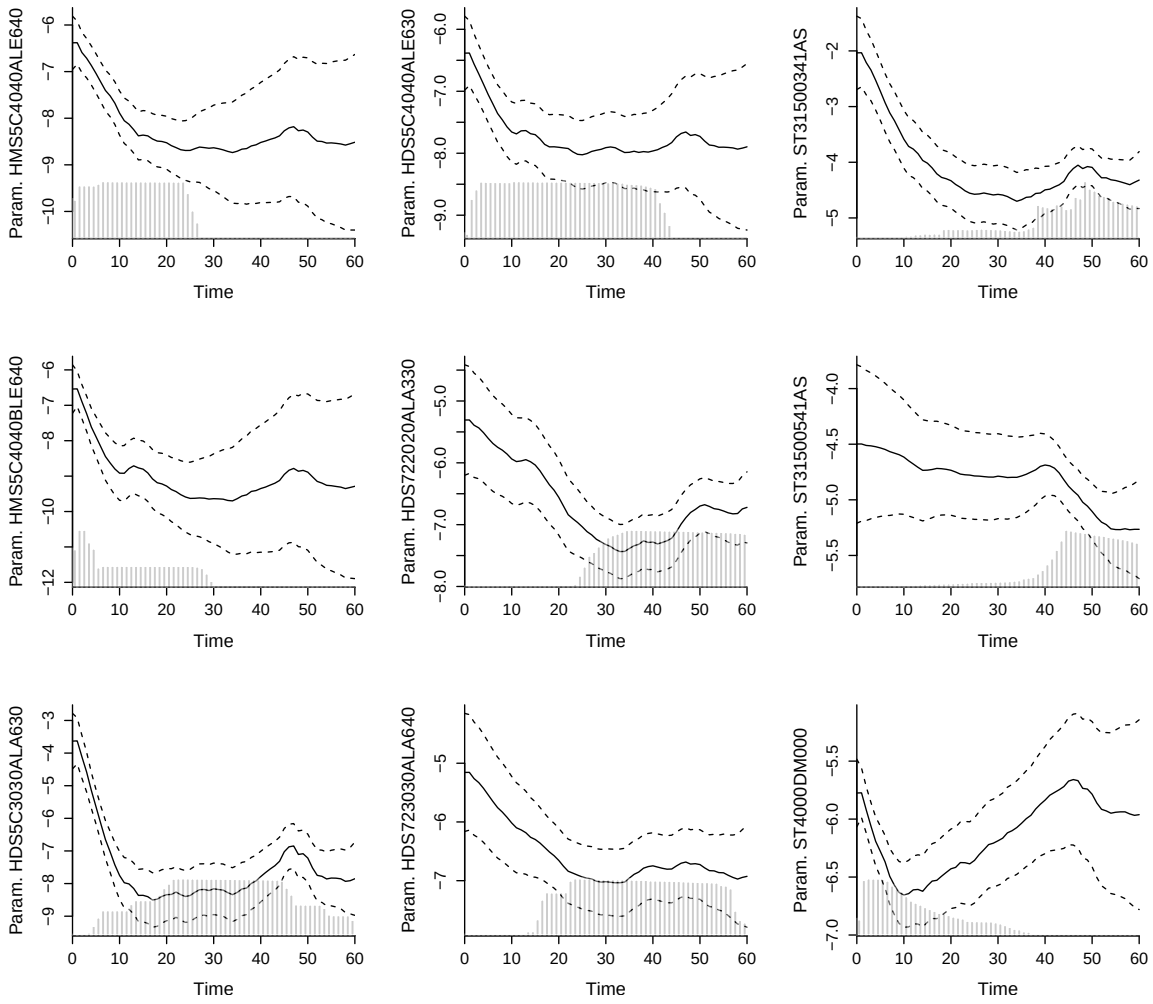


Figure 11: Predicted factor levels parameters with the continuous time model.

an example. Observation a has two covariate vectors in interval $d-1$. The first is $\boldsymbol{x}_{a1}$ as $t_{a0} < d-1$ and $t_{a1} > d-2$. Similar arguments apply for the covariate vector $\boldsymbol{x}_{a2}$.

Example with the continuous model

As mentioned in Section 5, the start and stop times in the hard disk failure data set are in fractions of months on a hourly precision. Thus, I can use the continuous model. I fit the model the EKF with more iterations in the correction step. I get the continuous model by setting `model = "exponential"`:

```
R> system.time(ddfit_cont <- ddhazard(formula = frm, data = hd_dat, by = 1,
+    max_T = 60, model = "exponential", id = hd_dat$serial_number,
+    Q_0 = diag(1, 23), Q = diag(0.1, 23), control = ddhazard_control(
+    NR_eps = 0.0001, eps = 0.001, LR = 0.5, method = "EKF")))

   user  system elapsed
 94.032   0.509  21.061
```

Figure 11 shows the first estimated factor levels' parameters. The results are comparable to what we have seen previously (e.g., see Figure 3 where I also used the EKF with more iterations in the correction step but with the discrete time model).

6. Simulations

In this section, I will simulate data using the first order random walk model and illustrate the computation time, and mean square error (MSE) of the predicted parameters as the number of individuals increases. I use a first order random walk for the parameters with 21 parameters. The intercept starts at -3.5, and the other parameters start at points drawn from the standard normal distribution. I set the intercept to a low value to decrease the baseline likelihood of an event in every interval. I let covariance matrix $\mathbf{Q}$ be a diagonal matrix which has 0.1^2 in the first entry (the intercept) and 0.33^2 for rest of the diagonal entries. The standard deviation is chosen lower for the intercept to ensure that the intercept does not change "too much" with high probability. Figure 12 provides an example of a draw of parameters.

I simulate a different number of individuals with $n = 2^{10}, 2^{11}, \dots, 2^{18}$ in each trial. Each individual is right censored at time 30, and I set the interval lengths to 1. Further, I simulate random delayed entry. We randomly start to observe each individual at time $0, 1, \dots, 29$ with a 50% chance of 0 and uniform chance on the other points. This mimics a situation like corporate default prediction where we use calendar time as the time scale. A firm may first be incorporated a while into the study, in which case the firm is subject to delayed entry.

Each individual has time-varying covariates that change after five periods. Thus, if an individual starts at time 2, his covariate vector changes at time $7, 12, \dots, 27$. The covariates are drawn from an iid standard normal distribution. For each value of n, I make 11 simulation trials. I estimate the UKF model only up to $n = 2^{15}$ because of the computation time. Further, I set the UKF hyperparameters to $(\alpha, \beta, \kappa) = (1, 0, 0.004)$, which yields $W_0^{[m]} = 0.0001$. $\mathbf{Q}_0$ for the EKF with extra iterations, and the GMA is a diagonal matrix with entries 1. The UKF has 0.01 as the diagonal entries. The EKF without extra iterations and the SMA have 10000 in the diagonal entries of $\mathbf{Q}_0$. All the filters have the starting value of $\mathbf{Q}$ as a diagonal matrix with 0.01 in the diagonal elements. All the methods take at most 25 iterations of the EM algorithm if the convergence criteria is not previously met.

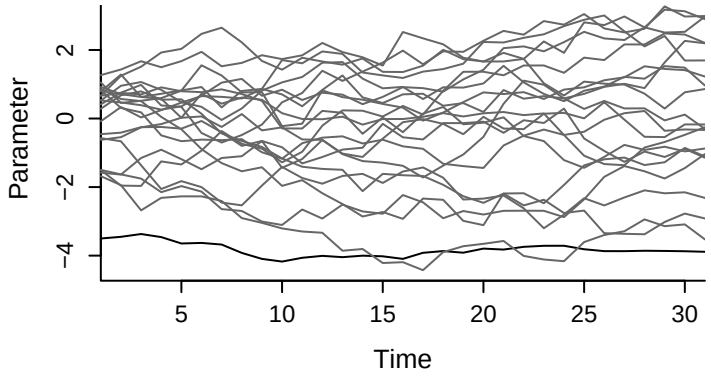


Figure 12: Example of parameters in the simulation experiment. The black curve is the intercept and the gray curves are the parameters for the covariates.

	EKF	EKF with extra iterations	UKF	SMA	GMA
Run time	2.459	5.087	15.003	11.025	5.245
Log-log slope	0.717	0.786	1.026	0.781	0.826

Table 3: Summary information of the computation time in the simulation study. The first row shows the median runtime for largest number of individuals. The UKF is only up to $n = 32768$. The second row shows the slope of the log computation time regressed on the log number of individuals for $n \geq 16384$.

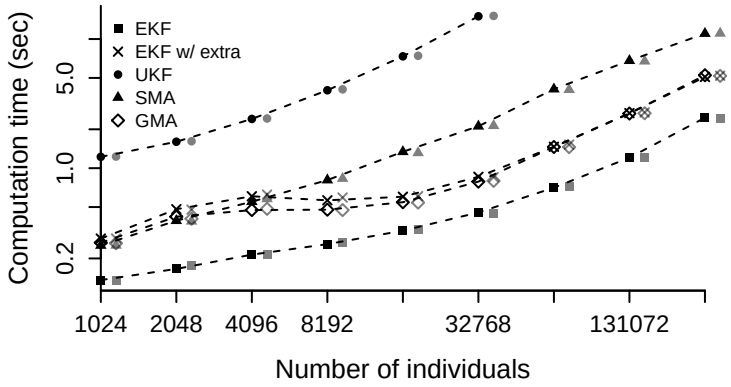


Figure 13: Median computation times of the simulations for each method for different values of n. The gray symbols to the right are the means. The filled squares are the EKF, the crosses are the EKF with extra iteration, the circles are the UKF, the triangles are the SMA, and the open squares are the GMA. The scales are logarithmic so a linear trend indicates that computation time is a power of n.

The simulations are run on a laptop with Ubuntu 18.04 with an Intel core i7-8750H @ 2.20GHz and 16GB ram. Figure 13 shows the medians and means of the computation time. Table 3 displays the median computation time for the largest value of n along with the regression slope of the log computation time regressed on the log number of individuals. All methods have a slope close to or below 1, reflecting the $\mathcal{O}(n_t)$ computational complexity. In fact, the slope is less than 1 for all but the UKF method. This can be explained by the overhead of the parallel computation. Further, the methods tend to use less EM iterations when more data is available. The latter can be seen from Figure 15, which shows the median number of iterations of the EM algorithm. All computation times include the time required to set up the model matrix and fit a weighted GLM to get a starting values for $\boldsymbol{\alpha}_0$. The setup time is equal for all methods.

Figure 14 shows a plot of the MSE for the parameters. The EKF with one iteration in the correction step does not improve much as n increases. Hence, more iterations seem preferable in this example. Some points are worth stressing. First, the computation time of the UKF can be reduced by using a multithreaded **BLAS** library or reimplementing the code. I have seen a reduction up to factor 2 for larger data sets on the setup used in the simulation when **OpenBLAS** (Xianyi, Qian, and Yunquan 2012) is used. Further, one can do more tuning (especially with the UKF) for each data set, which is not done in the present case.

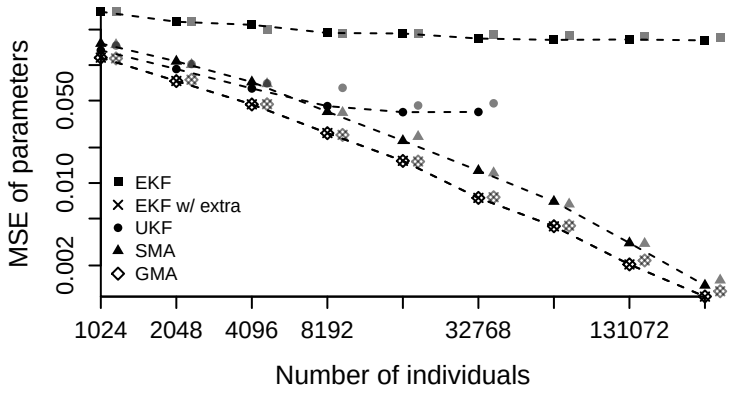


Figure 14: Median mean square error of predicted parameters of the simulations for each method for different values of n. The gray symbols to the right are the means. The filled squares are the EKF, the crosses are the EKF with extra iteration, the circles are the UKF, the triangles are the SMA, and the open squares are the GMA. The axis are on the logarithmic scale.

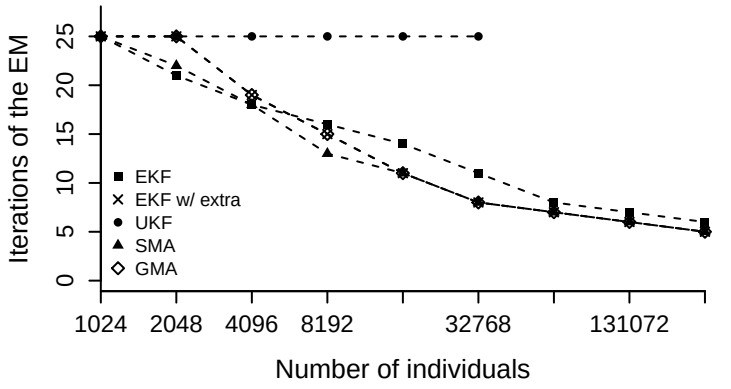


Figure 15: Median number of iterations of the EM algorithm. The filled squares are the EKF, the crosses are the EKF with extra iteration, the circles are the UKF, the triangles are the SMA, and the open squares are the GMA.

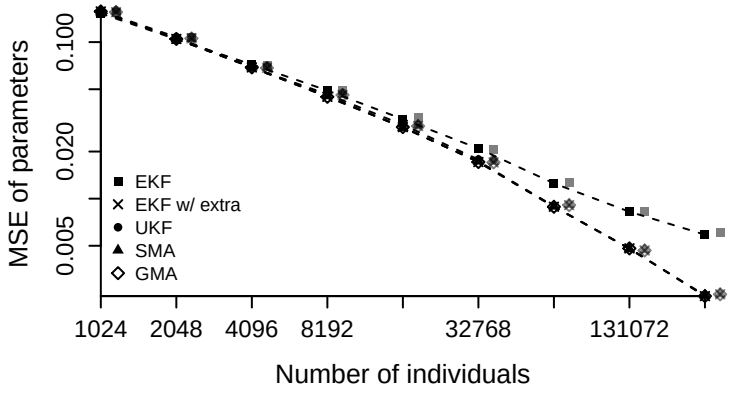


Figure 16: Similar plot to Figure 14 but where each element of the covariate vectors is drawn from $N\left(0, 0.33^2\right)$.

The simulation here is "extreme" in that the linear predictor can take large absolute values in the last intervals with a nonnegligible probability. Thus, I perform a second simulation experiment where I draw the covariates from a normal distribution with zero mean and variance 0.33^2. Figure 16 shows the MSEs. The difference between the filters is small in terms of mean square error. The computation times are similar to before in terms of the relative differences. Still, it seems that the EKF with extra iterations, and the GMA are preferred.

7. Comparison with other packages

I will end by comparing the implemented methods with other packages in R mentioned in the start of the article. Specifically, I will discretize the time line and use the `gam` function in the **mgcv** package to get a time-varying parameter by adding a spline over time for the intercept on the log odds scale as described in Tutz and Schmid (2016, Chapter 5). Further, I use the `pchreg` function from the **eha** package to get a piecewise constant intercept in an exponentially distributed arrival time model. We set each time interval to have length 1. I will also estimate a Cox model with the `coxph` function from the **survival** package and a generalized survival model mentioned in the introduction and shown in Equation 2 with the `gsm` function from the **rstpm2** package. Lastly, I use the discrete model with logit link function from `ddhazard` to get comparable results to the `gam` function. We use interval widths 0.5 for both `ddhazard` and `gam`.

For simplicity, we focus on the ST4000DM000 hard disk version. Further, we take the hard disk information up to month 35. We do not have data for most of the hard disks beyond this time point. We only estimate a linear effect on the link scale in each model for the power cycle count with a time constant parameter.

The left plot of Figure 17 shows the discrete hazard rates with the power cycle count equal to zero. It shows that the result from `gam` and `ddhazard` are similar. The estimate from `pchreg` is less smooth as there is a separate coefficient for each time interval of length 1 without any penalties. We use a natural cubic spline for the intercept in the `gam` fit. Thus, the log odds are linear in time beyond month 35 as these splines are linear beyond the boundary knots. This results in an almost linear increasing discrete hazard as the inverse logit function is almost linear for small inputs. Moreover, the prediction intervals are wider for the `gam` fit than for the `ddhazard` fit. The width of the prediction intervals with `ddhazard` fit mainly depends on the estimate of $\mathbf{Q}$ which is estimated to fit the whole curve.

The right plot of Figure 17 shows the estimated survival curve from the Cox model from the `coxph` call, an estimated generalized survival model using the `gsm` function from the **rstpm2**, and dynamic discrete hazard model from the `ddhazard`. The power cycle count is equal to zero for all the curves. The Cox model survival curve is computed with the `survfit` function while the latter is computed with the `ddsurvcurve` function. Six degrees of freedom is used in the spline for the baseline survival function in the `gsm` call. I also tried a minus logit link function with the `gsm` function but the difference in the log-likelihood at the maximums are negligible.

The three generally agree but only the dynamic discrete hazard model and the generalized survival model allow for extrapolation beyond time 35 as the baseline hazard is nonparametric in the Cox model. The curve from `ddsurvcurve` is illustrated as a step function as the model provides discrete hazard rates.

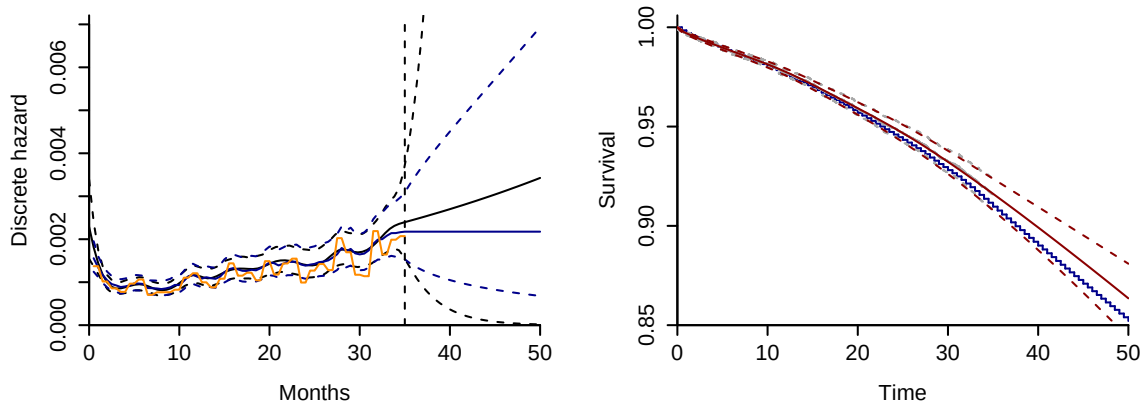


Figure 17: The left plot shows the discrete hazard rate when the power cycle count is equal to zero. The black line is the rate from `mgcv::gam`, the blue line is the rate from `dynamichazard::ddhazard`, and the orange line is the rate from `eha::pchreg`. The dashed lines are 95% back-transformed prediction intervals (confidence bounds are not provided by `eha::pchreg`). The right plot shows the estimated survival curve with the power cycle count equal to zero. The dark gray line is the curve from `survival::coxph` with 95% prediction intervals, the dark red line is the curve from `rstpm2::gsm` with 95% prediction intervals, and the blue curve is from `dynamichazard::ddhazard`.

8. Conclusion

I have covered the EM algorithm used in the `ddhazard` function in **dynamichazard** and the four different filters available with the `ddhazard` function, highlighting the pros and cons of the different filters. Further, I have covered the dynamic discrete time model and the dynamic continuous time model. The simulation study shows that the filters scale well with the number of observations and are fast. Further, the simulation study shows how the mean square error of the predicted parameters behaves with different numbers of observations. The extended Kalman filter has been compared with other methods in R.

I have not covered all the S3 methods provided in the **dynamichazard** package. These include `plot`, `predict`, `hatvalues`, and `residuals`. It is possible to include weights for the observations with all the filters. The details hereof are in the "*ddhazard*" vignette of this package. Furthermore, the `ddhazard_boot` function can be used to perform a nonparametric bootstrap. Weights are used in `ddhazard_boot` with case resampling, which reduces the computation time. Vignettes are provided with the **dynamichazard** package which illustrate the use of the mentioned functions. A demo of the models is available by running `ddhazard_app`. Particle filter and smoothers are provided with the package but are not covered in this paper. I will end by looking at potential further developments.

8.1. Further developments

I will summarize some potential future developments of the **dynamichazard** in this section. First, we can replace the random walk model with another type of multivariate autoregressive model. This will require additional parameter to be estimated which can be done in the M-step of the EM algorithm. See the constrained EM algorithm in the **MARSS** package (Holmes 2013) for update formulas.

Other models can be implemented in survival analysis, such as recurrent events and competing risk (see Fahrmeir and Wagenpfeil 1996). Furthermore, the methods can also be used outside survival analysis. For instance, with panel data with real valued outcomes, multinomial outcomes, or ordinal outcomes for each individual in each interval. The underlying time can depend on the context (e.g., it could be calender time or time since enrollment).

The logistic link function in the discrete model can be changed to other link functions without much work as both the C++ and R code is implemented like the `glm` function in R.

The current implementation of parallel computation is based on shared memory. However, we can extend the implementation to a distributed network. Rigatos (2017, Chapter 3) covers different ways of performing the computation on a distributed network. Two approaches are either to distribute the work in each step of the filter or to run separate filters and aggregate the filters at the end.

An alternative to the filters in the E-step is to use the linearization method described in Durbin and Koopman (2012, Section 10.6) mentioned in Section 4.4. It would be interesting to implement this approach in the package as well. Fahrmeir and Kaufmann (1991) describe an idea similar to the linearization method in Durbin and Koopman (2012, Section 10.6), using a Gauss-Newton and Fisher scoring method.

The methods discussed in this paper can be used as the initial input to the importance sampler with use of antithetic variables and control variables, as suggested by Durbin and Koopman (2000). This approach is implemented in the **KFAS** package. This can be used for approximate likelihood evaluation to perform maximum likelihood estimation as in the **KFAS** package instead of the EM algorithm.

All the models covered in this paper can be estimated with a suitable generalized linear mixed model with correlated random terms. Thus, we can perform approximate maximum likelihood estimation with e.g., the pseudo-likelihood method used in the `GLIMMIX` procedure in SAS, or the Laplace approximation used in the `GLIMMIX` procedure and the **lme4** package (Bates, Mächler, Bolker, and Walker 2015) in R. Alternatively, the implemented particle filters in the **dynamichazard** package can be used for approximate likelihood evaluations and parameter estimation.

Acknowledgments

Thanks to Hans-Rudolf Künsch and Olivier Wintenberger for constructive conversations. Furthermore, thanks to Søren Feodor Nielsen for feedback and ideas on most aspects of the **dynamichazard** package. I am thankful for the feedback from the anonymous reviewers who have helped to improve the article and particularly the comparison of the **dynamichazard** package with other software.

References

Aalen OO (1989). "A Linear Regression Model for the Analysis of Life Times." *Statistics in Medicine*, **8**(8), 907–925. doi:10.1002/sim.4780080803.

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum

A, Hammarling S, McKenney A, Sorensen D (1999). ***LAPACK*** *Users' Guide*. 3rd edition. Society for Industrial and Applied Mathematics, Philadelphia.

BackBlaze (2014). "Hard Drive SMART Stats." Accessed 2017-05-06, URL https://www.backblaze.com/blog/hard-drive-smart-stats/.

BackBlaze (2017). "Hard Drive Data and Stats." Accessed 2017-05-06, URL https://www.backblaze.com/b2/hard-drive-test-data.html.

Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using **lme4**." *Journal of Statistical Software*, **67**(1), 1–48. doi:10.18637/jss.v067.i01.

Broström G (2012). *Event History Analysis with R*. Chapman and Hall/CRC, Boca Raton. doi:10.1201/9781315373942.

Carpenter B, Gelman A, Hoffman MD, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017). *Stan: A Probabilistic Programming Language*. doi:10.18637/jss.v076.i01.

Christoffersen B (2021). ***dynamichazard***: *Dynamic Hazard Models Using State Space Models*. R package version 1.0.0, URL https://CRAN.R-project.org/package=dynamichazard.

Clements M, Liu XR, Christoffersen B (2021). ***rstpm2***: *Smooth Survival Models, Including Generalized Survival Models*. R package version 1.5.2, URL https://CRAN.R-project.org/package=rstpm2.

Cox DR (1972). "Regression Models and Life-Tables." *Journal of the Royal Statistical Society B*, **34**(2), 187–220. doi:10.1111/j.2517-6161.1972.tb00899.x.

Davidson-Pilon C, Kalderstam J, Jacobson N, Reed S, Kuhn B, Zivich P, Williamson M, Abdeali JK, Datta D, Fiore-Gartland A, Parij A, WIlson D, Gabriel, Moneda L, Moncada-Torres A, Stark K, Gadgil H, Jona, JoseLlanes, Singaravelan K, Besson L, Peña MS, Anton S, Klintberg A, Growth J, Noorbakhsh J, Begun M, Kumar R, Hussey S, Seabold S (2021). ***lifelines***: *0.26.0*. doi:10.5281/zenodo.4816284.

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 the Royal Statistical Society B*, **62**(1), 3–56. doi:10.1111/1467-9868.00218.

Durbin J, Koopman SJ (2012). *Time Series Analysis by State Space Methods*, volume 38. Oxford University Press, Oxford.

Fahrmeir L (1992). "Posterior Mode Estimation by Extended Kalman Filtering for Multivariate Dynamic Generalized Linear Models." *Journal of the American Statistical Association*, **87**(418), 501–509. doi:10.1080/01621459.1992.10475232.

Fahrmeir L (1994). "Dynamic Modelling and Penalized Likelihood Estimation for Discrete Time Survival Data." *Biometrika*, **81**(2), 317–330. doi:10.1093/biomet/81.2.317.

Fahrmeir L, Kaufmann H (1991). "On Kalman Filtering, Posterior Mode Estimation and Fisher Scoring in Dynamic Exponential Family Regression." *Metrika*, **38**(1), 37–60. doi:10.1007/bf02613597.

Fahrmeir L, Wagenpfeil S (1996). "Smoothing Hazard Functions and Time-Varying Effects in Discrete Duration and Competing Risks Models." *Journal of the American Statistical Association*, **91**(436), 1584–1594. doi:10.1080/01621459.1996.10476726.

Fine JP, Yan J, Kosorok MR (2004). "Temporal Process Regression." *Biometrika*, **91**, 683–703. doi:10.1093/biomet/91.3.683.

Frumento P (2021). ***pch**: Piecewise Constant Hazards Models for Censored and Truncated Data*. R package version 2.0, URL https://CRAN.R-project.org/package=pch.

Goeman JJ (2010). "L1 Penalized Estimation in the Cox Proportional Hazards Model." *Biometrical Journal*, **52**(1), 70–84. doi:10.1002/bimj.200900028.

Harrell Jr FE (2021). ***rms**: Regression Modeling Strategies*. R package version 6.2-0, URL https://CRAN.R-project.org/package=rms.

Hartikainen J, Solin A, Särkkä S (2011). *Optimal Filtering with Kalman Filters and Smoothers. A Manual for the MATLAB Toolbox EKF/UKF*.

Harvey AC, Phillips GDA (1979). "Maximum Likelihood Estimation of Regression Models with Autoregressive-Moving Average Disturbances." *Biometrika*, **66**(1), 49. doi:10.1093/biomet/66.1.49.

Helske J (2017). "**KFAS**: Exponential Family State Space Models in R." *Journal of Statistical Software*, **78**(10), 1–39. doi:10.18637/jss.v078.i10.

Holmes EE (2013). "Derivation of an EM Algorithm for Constrained and Unconstrained Multivariate Autoregressive State-Space (MARSS) Models." arXiv:1302.3919 [stat.ME], URL https://arxiv.org/abs/1302.3919.

Jackson C (2016). "**flexsurv**: A Platform for Parametric Survival Modeling in R." *Journal of Statistical Software*, **70**(8), 1–33. doi:10.18637/jss.v070.i08.

Johansen A (2009). "**SMCTC**: Sequential Monte Carlo in C++." *Journal of Statistical Software*, **30**(6), 1–41. doi:10.18637/jss.v030.i06.

Julier SJ, Uhlmann JK (1997). "New Extension of the Kalman Filter to Nonlinear Systems." In Ivan Kadar (ed.), *Signal Processing, Sensor Fusion, and Target Recognition VI*, volume 3068, pp. 182–193. Proc. SPIE. doi:10.1117/12.280797.

King A, Nguyen D, Ionides E (2016). "Statistical Inference for Partially Observed Markov Processes via the R Package **pomp**." *Journal of Statistical Software*, **69**(1), 1–43. doi:10.18637/jss.v069.i12.

King AA, Ionides EL, Bretó CM (2021). ***pomp**: Statistical Inference for Partially Observed Markov Processes*. R package version 3.5, URL https://CRAN.R-project.org/package=pomp.

Klein A (2015). "CSI: Backblaze – Dissecting 3TB Drive Failure." Accessed 2017-05-06, URL https://www.backblaze.com/blog/3tb-hard-drive-failure/.

Klein A (2016). "One Billion Drive Hours and Counting: Q1 2016 Hard Drive Stats." Accessed 2017-05-06, URL https://www.backblaze.com/blog/hard-drive-reliability-stats-q1-2016/.

Kooperberg C (2020). ***polspline**: Polynomial Spline Routines*. R package version 1.1.19, URL https://CRAN.R-project.org/package=polspline.

Koopman SJ, Durbin J (2000). "Fast Filtering and Smoothing for Multivariate State Space Models." *Journal of Time Series Analysis*, **21**(3), 281–296. doi:10.1111/1467-9892.00186.

Koopman SJ, Shephard N, Doornik JA (2008). *Statistical Algorithms for Models in State Space Form: **SsfPack** 3.0*. Timberlake Consultants Press.

Lambert PC, Royston P (2009). "Further Development of Flexible Parametric Models for Survival Analysis." *Stata Journal*, **9**(2), 265–290(26). doi:10.1177/1536867x0900900206.

Liu XR, Pawitan Y, Clements M (2016). "Parametric and Penalized Generalized Survival Models." *Statistical Methods in Medical Research*, **27**(5), 1531–1546. doi:10.1177/0962280216664760.

Liu XR, Pawitan Y, Clements MS (2017). "Generalized Survival Models for Correlated Time-to-Event Data." *Statistics in Medicine*, **36**(29), 4743–4762. doi:10.1002/sim.7451.

Martinussen T, Scheike TH (2006). *Dynamic Regression Models for Survival Data*. Springer-Verlag.

Menegaz HMT (2016). *Unscented Kalman Filtering on Euclidean and Riemannian Manifolds*. PhD dissertation, University of Brasilia.

Nordh J (2017). "**pyParticleEst**: A Python Framework for Particle-Based Estimation Methods." *Journal of Statistical Software*, **78**(3), 1–25. doi:10.18637/jss.v078.i03.

OpenMP Architecture Review Board (2013). ***OpenMP** Application Program Interface*. Version 4.0, URL http://www.openmp.org/.

Park MY, Hastie T (2018). ***glmpath**: L1 Regularization Path for Generalized Linear Models and Cox Proportional Hazards Model*. R package version 0.98, URL https://CRAN.R-project.org/package=glmpath.

Peng JY, Aston J (2011). "The State Space Models Toolbox for MATLAB." *Journal of Statistical Software*, **41**(6), 1–26. doi:10.18637/jss.v041.i06.

Petris G, Petrone S (2011). "State Space Models in R." *Journal of Statistical Software*, **41**(1), 1–25. doi:10.18637/jss.v041.i04.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rigatos G (2017). *State-Space Approaches for Modelling and Control in Financial Engineering, Systems Theory and Machie Learning Methods*. Springer-Verlag.

Rossum G (2017). "Python Reference Manual." *Technical report*, Centrum voor Wiskunde en Informatica (CWI). Release 2.7.14, URL https://docs.python.org/2.7/library/.

Sanderson C, Curtin R (2016). "**Armadillo**: A Template-Based C++ Library for Linear Algebra." *The Journal of Open Source Software*, **1**(2). doi:10.21105/joss.00026.

SAS Institute Inc (2017). *SAS/STAT Software, Version 9.4*. Cary. URL http://support.sas.com/.

Shumway RH, Stoffer DS (1982). "An Approach to Time Series Smoothing and Forecasting Using the EM Algorithm." *Journal of Time Series Analysis*, **3**(4), 253–264. ISSN 1467-9892. doi:10.1111/j.1467-9892.1982.tb00349.x.

Simon N, Friedman J, Hastie T, Tibshirani R (2011). "Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent." *Journal of Statistical Software*, **39**(1), 1–13. doi:10.18637/jss.v039.i05.

StataCorp (2017). *Stata Statistical Software: Release 15*. StataCorp LLC, College Station.

Therneau TM (2021). ***survival**: Survival Analysis*. R package version 3.2-13, URL https://CRAN.R-project.org/package=survival.

Therneau TM, Grambsch PM (2000). *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York.

Thomas L, Reyes E (2014). "Tutorial: Survival Estimation for Cox Regression Models with Time-Varying Coefficients Using SAS and R." *Journal of Statistical Software, Code Snippets*, **61**(1), 1–23. ISSN 1548-7660. doi:10.18637/jss.v061.c01.

Tusell F (2011). "Kalman Filtering in R." *Journal of Statistical Software*, **39**(1), 1–27. doi:10.18637/jss.v039.i02.

Tutz G, Schmid M (2016). *Modeling Discrete Time-to-Event Data*. Springer-Verlag. doi:10.1007/978-3-319-28158-2.

Wan EA, Merwe RVD (2000). "The Unscented Kalman Filter for Nonlinear Estimation." In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pp. 153–158. doi:10.1109/asspcc.2000.882463.

Wang W, Chen MH, Wang X, Yan J (2020). ***dynsurv**: Dynamic Models for Survival Data*. R package version 0.4-2, URL https://CRAN.R-project.org/package=dynsurv.

Wood S (2006). *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC.

Xianyi Z, Qian W, Yunquan Z (2012). "Model-Driven Level 3 **BLAS** Performance Optimization on Loongson 3A Processor." In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pp. 684–691. doi:10.1109/icpads.2012.97.

Xiong K, Zhang HY, Chan CW (2006). "Performance Evaluation of UKF-Based Nonlinear Filtering." *Automatica*, **42**(2), 261–270. doi:10.1016/j.automatica.2005.10.004.

Yan J, Fine JP (2004). "Estimating Equations for Association Structures." *Statistics in Medicine*, **23**, 859–880. doi:10.1002/sim.1650.

Zhou Y (2015). "**vSMC**: Parallel Sequential Monte Carlo in C++." *Journal of Statistical Software*, **62**(9), 1–49. doi:10.18637/jss.v062.i09.

Affiliation:

Benjamin Christoffersen
Center for Statistics
Copenhagen Business School
Solbjerg Pl. 3, 2000 Frederiksberg, Denmark
E-mail: benjamin.christoffersen@ki.se
URL: https://staff.ki.se/people/benjamin-christoffersen

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
September 2021, Volume 99, Issue 7 *Submitted:* 2017-09-29
doi:10.18637/jss.v099.i07 *Accepted:* 2020-08-09