Comparing with other packages

Summary

This note will compare the package `dynamichazard` I am working on with current methods avialable in `R`. First, we will look at other packages in `R` that has time variying effects in multiplicative hazard models. Then we will turn to the methods I have implemented

survival::cancer

This section will use the dataset `survival::cancer` to compare the methods. The data set consists of lung cancer patients where some die after `time` column while others are right censored. Dying is coded through `status` with `1` implying right censored and `2` implying a death. For more information on the dataset see `?cancer`. We will focus on the `age` and `sex` variable in the data set. We start by attaching the `survival` package and standardizing the `age` variable. The later is important when we the use the Unscented Kalman filter (UKF)

```
library(survival)
head(cancer)
```

```
##   inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
## 1    3  306      2  74   1       1       90       100     1175      NA
## 2    3  455      2  68   1       0       90        90     1225      15
## 3    3 1010      1  56   1       0       90        90       NA      15
## 4    5  210      2  57   1       1       90        60     1150      11
## 5    1  883      2  60   1       0      100        90       NA       0
## 6   12 1022      1  74   1       1       50        80      513       0
```

```
# Standardize
cancer$age <- (cancer$age - mean(cancer$age)) / sd(cancer$age)

# Add id to keep track later
cancer$id <- seq_len(nrow(cancer))

c("Number of patients" = nrow(cancer),
  "Number of deaths" = sum(cancer$status == 2))
```

```
## Number of patients   Number of deaths
##                228                165
```

mgcv

The first method we will compare with is Generalized Additive Models (GAM) by using the `gam` function in the `mgcv` package. The model we fit is of the form:

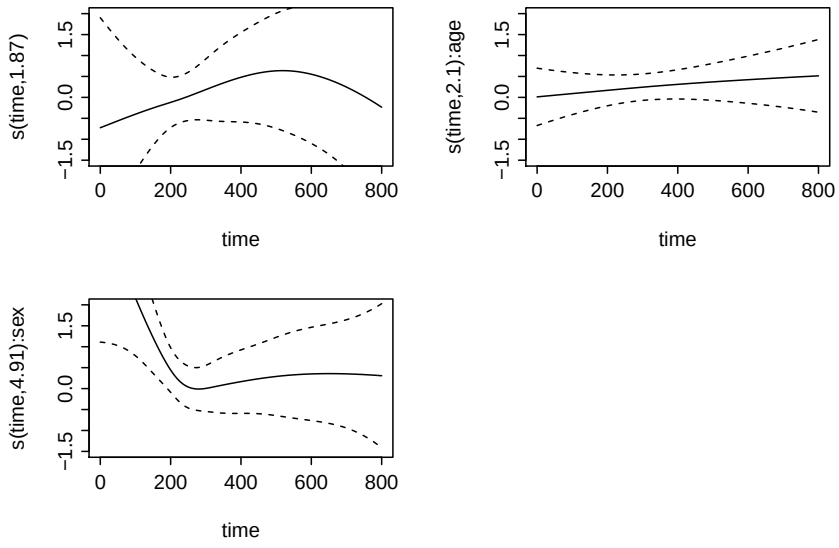
$$\text{logit}(\pi_i) = \vec{\gamma}_{\text{time}}\vec{f}_{\text{time}}(t_i) + \vec{\gamma}_{\text{age}}\vec{f}_{\text{time}}(t_i)a_i + \vec{\gamma}_{\text{sex}}\vec{f}_{\text{sex}}(t_i)s_i$$

where π_i is the probability that the i'th individual dies of cancer, t_i is the stop time of the i'th indvidual, a_i is the age of the i'th individual and s_i is the sex of the i'th individual. $\vec{f}.$ is a basis function. We will use cubic regression splines with knots spread evenly through the covariate values. We fit the model with the following call:

```
library(mgcv, quietly = T)
```

```
## This is mgcv 1.8-15. For overview type 'help("mgcv-package")'.
```

```
spline_fit <- gam(
  formula = status == 2 ~ -1 + s(time, bs = "cr", k = 6) + # cr yields cubic ba-
                                                           # sis with dim of k
    s(time, bs = "cr", k = 6, by = age) +
    s(time, bs = "cr", k = 6, by = sex),
  family = binomial, data = cancer,
  method = "GCV.Cp" # estimate smoothing parameters with generalized cross vali-
                    # dation
                  )

# summary(spline_fit) # TODO: Do we want to see the summary?
plot(spline_fit, xlim = c(0, 800), ylim = c(-1.5, 2), rug = F,
     pages = 1)
```



The plot in the upper right corner is the intercept coeffecient. The lower right corner is the coeffecient of `sex` and the upper left corner is the coeffecient of `age`. The parameters in the sets $\vec{\gamma}_{\text{time}}$, $\vec{\gamma}_{\text{age}}$ and $\vec{\gamma}_{\text{sex}}$ are penalized with a smoothing parameter selected with generalized cross validation. The final plot of the estimates do suggest that there may be curvuture in `sex` variable. The estimate for the `sex` variable seems resonable given that men have higher propensity to die early from cancer then women as illustrated in the cross table below:

```
# sex = 1 is male
# First we look at the proportion that die before time 200
ftable(xtabs(~ I(time <= 200) + I(status == 2) + sex, data = cancer),
       row.vars = 1:2)
```

```
##                               sex  1  2
## I(time <= 200) I(status == 2)
## FALSE          FALSE              20 31
##                TRUE               58 35
## TRUE           FALSE               6  6
##                TRUE               54 18
```

```
# Then we look at the proportion that die before time 800
ftable(xtabs(~ I(time <= 800) + I(status == 2) + sex, data = cancer),
       row.vars = 1:2)
```

```
##                               sex   1   2
## I(time <= 800) I(status == 2)
## FALSE          FALSE               4   2
##                TRUE                2   0
## TRUE           FALSE              22  35
##                TRUE              110  53
```

The model does not take into account that we people who survive to time say 800 also survive at time 200. For instance, if only had right censoring after time 600 then the above model would have no controls before that time. This is not the case and right censoring do occour through the time period we look at. Thus, the above model is chosen to keep the model setup simple

timereg

Another method we can try is a timevariying effects cox model from the package `timereg` based on the book 'Dynamic Regression Models for Survival Data'. The model we fit has an instanous hazard $\lambda(t)$ given by:

$$\lambda(t) = \lambda_0(t) \exp\left(\vec{x}\vec{\beta}(t)\right)$$

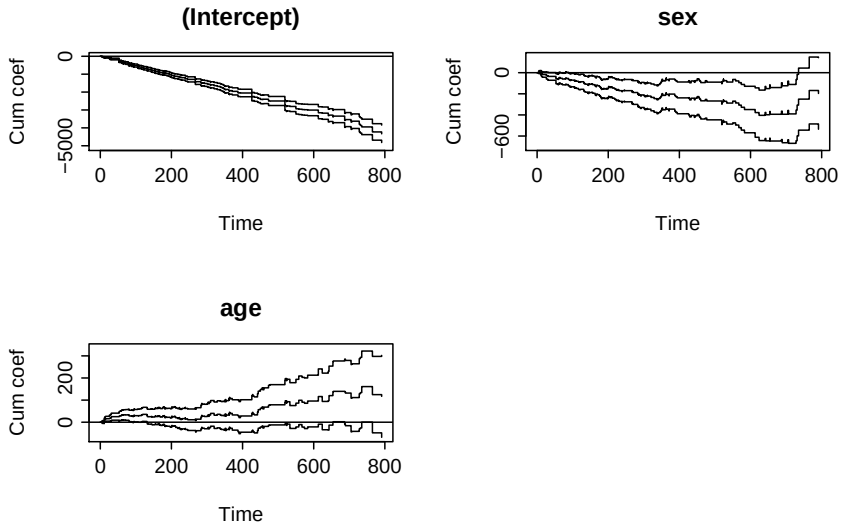
where each margin of $\vec{\beta}(t)$ is estimated with method described shortly. Below we will plot the cummulative regression function $B_i(t) = \int_0^t \beta_i(s)ds$.

```
library(timereg)
library(survival)
arg_list <- list( # We will re-use these parameters later
  formula = Surv(rep(0, nrow(cancer)), time, status == 2) ~ age + sex,
  max_T = 800, data = cancer)

# The Breslow estimate of baseline is not available at this point
tryCatch({
  cox_fit <- timecox(arg_list$formula, data = arg_list$data, max.time = arg_list$max_T,
                     method = "breslow")
}, error = function(e) cat(e$message))
```

```
## Only runs the default method at the moment
```

```
cox_fit <- timecox(arg_list$formula, data = arg_list$data, max.time = arg_list$max_T,
                   method = "basic")
```

```r
# summary(cox_fit) # TODO: Do we want to see the summary?
par(mfcol = c(2, 2), mar = c(5,5,3,0.5))
plot(cox_fit, ylab = "Cum coef")
```



The above code first shows that non-parametric `breslow` estimate is currenlty not supported. Instead the baseline $\lambda_0(t) = \exp(\alpha_0(t))$ where $\alpha_0(t)$ is estimated in a similar to way to $\vec{\beta}(t)$. $\vec{\beta}(t)$ is estimated recursivly with an update equation that is simplified through a first order Taylor expansion and adding a smoothness through weighting the time changes with a uniform contionous kernal. See 'Martinussen, Torben, and Thomas H. Scheike. *Dynamic regression models for survival data*. Springer Science & Business Media, 2007.' for details

Notice that the cummulative coeffecient for the `intercept` and `age` seems close to linear while we do have curvture in the `sex` coeffecient. This is consistent with curvutre we found with our logistic fit using the `mgcv`

Other packages

For completness, there are many other packages that estimate cox regression and GAM models with timevariying effects. For example, we could have used `mgcv::cox.ph` to compare with only one package. The `timereg` was used instead to illustrate two different packages

dynamichazard

This section will show results for the logistic model with timevariying effect that I have implemented. Estimates are shown both for the extended Kalman filter (EKF) and the Unscented Kalman fillter (UKF). First, we will breifly cover the models

The idea is that we discretize the outcomes into $1, 2, \dots, T$ bins. In each bin, we observe whether the indviduals dies or is right censored. The state space model we are applying is of the form:

$$\begin{array}{ll} \vec{y}_t = \vec{z}_t(\vec{\alpha}_t) + \vec{\epsilon}_t & \vec{\epsilon}_t \sim (\vec{0}, \mathbf{H}_t(\vec{\alpha}_t)) \\ \vec{\alpha}_{t+1} = \mathbf{F}\vec{\alpha}_t + \mathbf{R}\vec{\eta}_t & \vec{\eta}_t \sim N(\vec{0}, \mathbf{Q}) \end{array}, \qquad t = 1, \dots, n$$

where $y_{it} \in \{0, 1\}$ is the i'th indviduals outcome at time t. $\cdots \sim (a, b)$ notes a random variable with mean (or mean vector) a and variance (or co-variance matrix) b. It needs not to be normally distributed.

$z_{it}(\vec{\alpha}_t) = h(\vec{\alpha}_t \vec{x}_{it})$ where h is the link function. We use the logit model model in this example. The current implementation supplies $\mathbf{F}$ and $\mathbf{R}$ such that we have first and second order random walk

Firstly, we will need estimate the starting value $\vec{\alpha}_0$ and co-varianec matrix $\mathbf{Q}$. This is done through an EM-algorithm. The E-step either use the Extended Kalman filter or Uncented Kalman Filter. Either methods yields smoothed estimates of $\vec{\alpha}_1, \dots, \vec{\alpha}_T$, smoothed co-variance matrix and smoothed correlation need for the M-step. At this point $\mathbf{Q}_0 = \kappa \mathbf{I}$ is fixed to a large value κ

Extended Kalman filter

The idea behind the EKF is to linearize $\vec{z}_t(\vec{\alpha}_t)$ through a first order Taylor expansion and apply the regular Kalamn filter to the linearized model. The implemented version uses the method described in 'Fahrmeir, Ludwig. *Dynamic modelling and penalized likelihood estimation for discrete time survival data.* Biometrika 81.2 (1994): 317-330.'

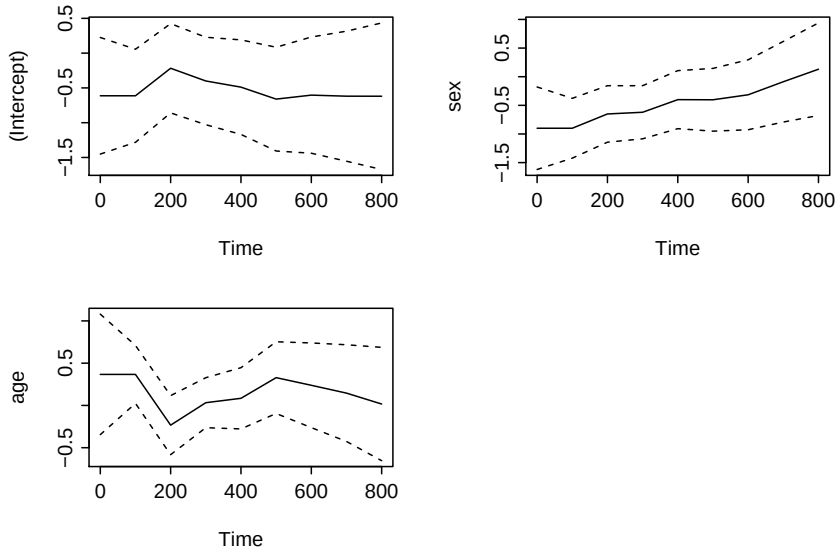
Fahrmeier applies the the Woodbury matrix identity to re-write the filter step of the Kalman filter to gain a method that is linear in time complexity of the dimension of the observational equation. In contrast the orginal formulation in is cubic. Furhter, the filter step can be caried out in parallel make the method more applicable to large data sets. We will return to this later

We can fit the model the following call to `ddhazard`:

```
# # Download the version the code is run with
# devtools::install_github(
#   "boennecd/dynamichazard@6af6b6d3c6807829492b0645bc5ca45c0f8527c4") # TODO: Update link

library(dynamichazard)
arg_list <- c(arg_list, list(
  by = 100,
  Q_0 = diag(rep(1, 3)), est_Q_0 = F,
  verbose = F))
dd_fit_EKF <- do.call(ddhazard, arg_list)
```

```
## a_0 not supplied. One iteration IWLS of static logit model is used
```

```
par(mfcol = c(2, 2))
for(i in 1:3)
  plot(dd_fit_EKF, cov_index = i, type = "cov")
```



The method set $\vec{\alpha}_0$ equal to the estimate from one iteration of the iteratively reweighted least squares model with time in-variante coeffecients. Each observations is weighted according to the number intervals they are in (we will cover more details in a bit). $\mathbf{Q}_0$ is set to diagonal matrix (1, 1, 1). Each time intervals has length 100 and the maximum time T is set to 800

The plots shows estimates of coeffecients $\vec{\alpha}_1, \dots, \vec{\alpha}_T$. The confidence bounds are pointwise estimates using the smoothed variance matrix $\text{Var}\left(\alpha_t | \vec{y}_1, \dots, \vec{y}_T\right)$. We can notice that the `sex` coeffecient seems to have an upward slope

Uncented Kalman Filter

The idea behind UKF is select points from the state equation $\vec{\alpha}_t$ and use these to approximate the distribution of observed outcomes $\vec{y}_t$. To be more concrete, let m denote the dimension of the state equation. Then we select $2m + 1$ $\vec{\mathcal{X}}_{0,t}, \vec{\mathcal{X}}_{1,t}, \dots, \vec{\mathcal{X}}_{2m,t}$. Each sigma point has a sigma weight of W_i where $W_0 = \lambda/(m + \lambda)$ and $W_1 = \cdots = W_{2m} = 1/(2(m + \lambda))$ (λ will be specified shortly). The sigma points are computed in each iteration of the UKF by:

$$
\begin{aligned}
\vec{a}_{t|t} = E\left(\vec{\alpha}_t | \vec{y}_1, \dots, \vec{y}_t\right), \qquad \mathbf{V}_{t|t} &= \text{Var}\left(\vec{\alpha}_t | \vec{y}_1, \dots, \vec{y}_t\right) \\
\vec{\mathcal{X}}_{0,t} &= \vec{a}_{t|t} \\
\vec{\mathcal{X}}_{1,t} &= \vec{a}_{t|t} + \sqrt{\lambda + m}\left(\sqrt{\mathbf{V}_{t|t}}\right)_1 \\
\vec{\mathcal{X}}_{2,t} &= \vec{a}_{t|t} + \sqrt{\lambda + m}\left(\sqrt{\mathbf{V}_{t|t}}\right)_2 \\
&\vdots \\
\vec{\mathcal{X}}_{1+m,t} &= \vec{a}_{t|t} - \sqrt{\lambda + m}\left(\sqrt{\mathbf{V}_{t|t}}\right)_1 \\
&\vdots
\end{aligned}
$$

where $\left(\sqrt{\mathbf{V}_{t|t}}\right)_i$ is i'th column of the Cholesky decomposition of $\mathbf{V}_{t|t}$. λ is set such that $\lambda = a(m + \kappa) - m$ where $a \in (0, 1]$ controls the spread of the sigma point (usually set to 1) and κ is typically set to 0 or $3 - m$. After we have computed the sigma point, we compute the outcomes $\vec{\mathcal{Y}}_{i,t} i = \vec{z}_t(\vec{\mathcal{X}}_{i,t})$ and use the pairs of $(\vec{\mathcal{Y}}_{i,t}, \vec{\mathcal{X}}_{i,t})$ and sigma weights to compute the means, co-variance and correlation matricies needed for the Kalman filter

Two usefull reference for UKF are 'Julier, Simon J., and Jeffrey K. Uhlmann. *Unscented filtering and nonlinear estimation*. Proceedings of the IEEE 92.3 (2004): 401-422.' and 'Julier, Simon J., and Jeffrey K.

Uhlmann. *New extension of the Kalman filter to nonlinear systems.* AeroSense'97. International Society for Optics and Photonics, 1997.'

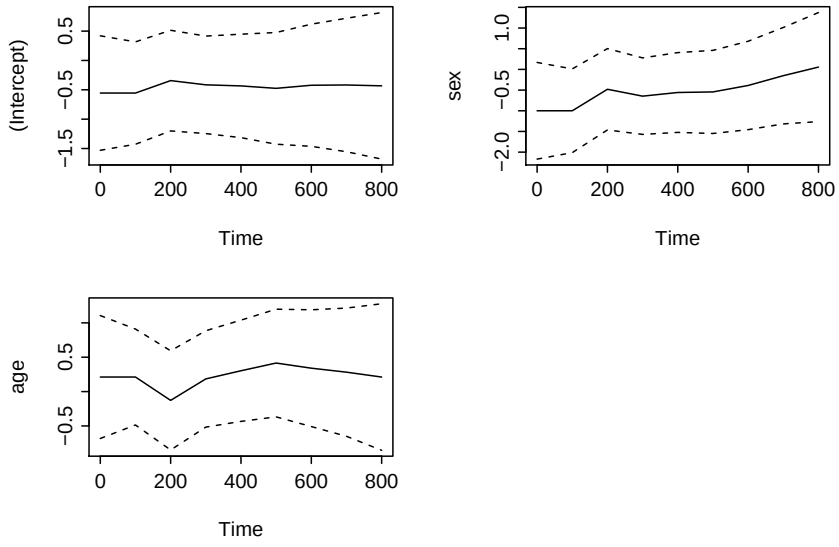
The paramitization for λ differs from these articles but yields the same results when the theoratical optimal values are used. The details are omitted here to keep focus on the idea of the UKF. Lastly, the implementation differs from all describsions I have seen so far by applying the the Woodbury matrix identity to get an algorithm that is linear in the number of observations. We will return to this later

We call `ddhazard` below to estimate with the UKF method. We print the co-variates estimates after the estimation

```
arg_list$method <- "UKF"
arg_list$kappa <- 3 - 3
arg_list$Q_0 <- diag(rep(1, 3))
arg_list$Q <- diag(rep(1e-2, 3))

dd_fit_UKF <- do.call(ddhazard, arg_list)
```

```
## a_0 not supplied. One iteration IWLS of static logit model is used
```

```
par(mfcol = c(2, 2))
for(i in 1:3)
  plot(dd_fit_UKF, cov_index = i, type = "cov")
```



We set the initial **Q** to a diagonal matrix with elements c(0.01, 0.01, 0.01) to avoid issues with the Cholesky decomposition in the first iteration. We find similar estimates as before though the confidence bounds are wider

Regular glm

Another idea is to fit a model with time in-variant effects. A straight forward model would be to to include each row in the with a weight as one. This is done below:

```
glm_args <- list(
  formula = status == 2 & time <= arg_list$max_T ~ age + sex,
  family = "binomial", data = cancer)
glm_fit <- do.call(glm, glm_args)

summary(glm_fit)$coefficients
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    2.320      0.478    4.85 1.23e-06
## age            0.286      0.152    1.88 6.07e-02
## sex           -0.955      0.304   -3.14 1.69e-03
```

However, we may want to take into account that some individuals survives for longer time than others. For this reason I have made the `static_glm` function. It puts weights to each observation according to how many intervals the observations is in. Say for instance that we have an indvidual who dies of cancer at time 750 were we make intervals of length 100. This observation will have two rows in the final model: 1) which it is a control with a weight of 7 (he survived up to time 700) and 2) one where it is a case with a weight of 1 (we observe that he dies at in interval $(700, 800]$). The call to the function is made below:

```
glm_args$formula <- arg_list$formula
glm_args$by <- arg_list$by
glm_args$max_T <- arg_list$max_T

glm_fit_new <- do.call(static_glm, glm_args)

summary(glm_fit_new)$coefficients
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.577     0.2667   -2.16  0.03060
## age            0.152     0.0927    1.64  0.10034
## sex           -0.563     0.1859   -3.03  0.00248
```

The method requires the same `survival::Surv` on the right hand site of the `~` in the formula as the `ddhazard` function. Note that the coeffecient are similar to the estimates from `ddhazard` as expected

We can illustrate the difference between the two `glm` models by noting that the number of addtional rows is equal to the number of deaths that occour after `arg_list$by`:

```
# There is an added row for every case that has an event beyond the first interval
nrow(glm_fit_new$data) # number of rows from static_glm
```

```
## [1] 360
```

```
nrow(glm_fit$data) # number of rows from glm
```

```
## [1] 228
```

```
sum(glm_fit$data$time <= arg_list$max_T & # before max_T
      glm_fit$data$time > arg_list$by & # after first interval
      glm_fit$data$status == 2) # observed to die
```

```
## [1] 132
```

The additional rows has the same co-variate values but the outcome variable `Y` differs as illustrated below:

```
# The weight vector count the number of bins
head(glm_fit$data[, c("status", "time", "sex", "age")]) # data set from glm
```

```
##   status time sex    age
## 1      2  306   1  1.273
## 2      2  455   1  0.612
## 3      1 1010   1 -0.711
## 4      2  210   1 -0.600
## 5      2  883   1 -0.270
## 6      1 1022   1  1.273
```

```
# An extra row is added for events before max_T
glm_fit_new$data[glm_fit_new$data$id %in% glm_fit$data$id[1:5],
                 c("Y", "weights", "sex", "age")]
```

```
##      Y weights sex    age
## 1    0       3   1  1.273
## 2    0       4   1  0.612
## 3    0       8   1 -0.711
## 4    0       2   1 -0.600
## 5    0       8   1 -0.270
## 410  1       1   1 -0.600
## 1100 1       1   1  1.273
## 229  1       1   1  0.612
```

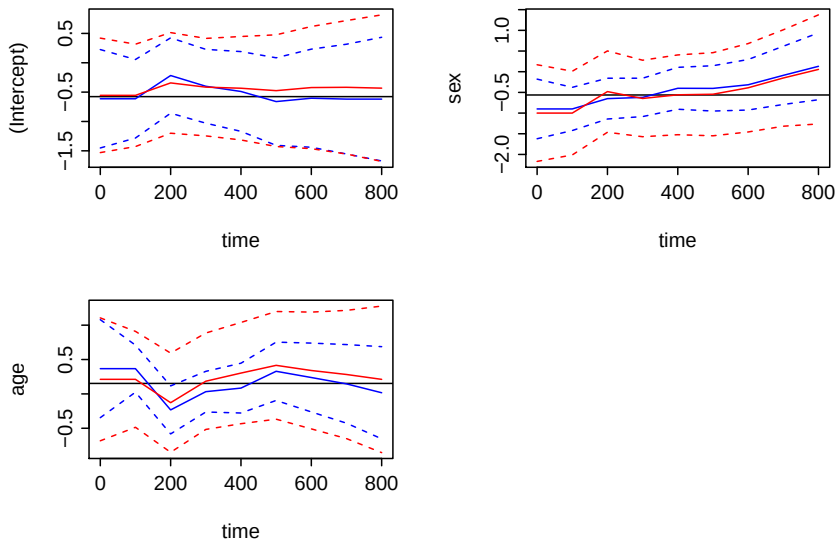
As a last note, it is one iteration from the iterated weighted least square from the `static_glm` that is used when the initial $\vec{\alpha}_0$ is not supplied to `ddhazard`

Comparing all the fits from dynamichazard

We can compare all the estimates from this package in a final plot. The code belows produces the plot. The black line is the estimate from the `static_glm` fit, the blue line is the estimates from EKF fit and the red line is the estimate from the UKF fit:

```
par(mfcol = c(2, 2))
# par(cex.axis = par()$cex.axis * 1.5, cex.lab = par()$cex.lab * 1.5)
for(i in 1:3){
  # glm estimates
  est_glm <- glm_fit_new$coefficients[i]

  # empty plot
  plot(c(0, arg_list$max_T), range(c(
    est_glm,
    dd_fit_EKF$a_t_d_s[, i] +
      sqrt(dd_fit_EKF$V_t_d_s[i, i,]) %*% t(c(1.96, -1.96)),
    dd_fit_UKF$a_t_d_s[, i] +
      sqrt(dd_fit_UKF$V_t_d_s[i, i,]) %*% t(c(1.96, -1.96)))),
```

```
    type = "n", xlab = "time", ylab = names(est_glm))

  # Add estimates from static_glm
  abline(h = est_glm, lty = 1, col = "black")

  # Add estimates from EKF
  plot(dd_fit_EKF, cov_index = i, type = "cov", add = T, col = "blue")

  # Add estimates from UKF
  plot(dd_fit_UKF, cov_index = i, type = "cov", add = T, col = "red")
}
```



We observe that the EKF fit and UKF fit are not far from each other. Though, the confidence bounds are wider for the UKF method. Moreover, the estimates are not far from the `static_glm` fit

dynamichazard implementation

The estimation in `ddhazard` is carried out `c++` using the linear algebra library `Armadillo`. `Armadillo` provides an API for `LAPACK` and `BLAS` which means that an cpu optimized version of the two can decrease the computation time

Further, the EKF implementation use the formulation from Fahrmier (1994) as previosly metioned. The `std` library `thread` is used to compute the filter step in parallel. The implementation do take into account that a multithreaded `BLAS` version may have been used to compile the code. Thus `openblas_set_num_threads` is used to toogle the number of threads `BLAS` will use during and after the filter step

The UKF method is not computed in parallel at this point. Though, parts of the matrix operations will be computed in parallel if a multithreaded `BLAS` is used.

Simulation

We will simulate a series of co-varaites and indviduals in order to show 1) the performance and 2) that the EKF and UKF has a linear time complexity in the number of observations. We will use the function

`test_sim_func_logit` in the `R/test_utils.R` file. First, we source the file and then we print the function. You can skip the function definition if you like and go to the explanation that comes after the print:

```
# We are currenlty in the vignettes folder
gsub("(^.*)(dynamichazard.*)", ".../\\2", getwd())
```

```
## [1] ".../dynamichazard/vignettes"
```

```
source("../R./test_utils.R")
test_sim_func_logit
```

```
## function (n_series, n_vars = 10, t_0 = 0, t_max = 10, x_range = 0.1,
##     x_mean = -0.1, re_draw = T, beta_start = 3, intercept_start,
##     sds = rep(1, n_vars + (!missing(intercept_start))))
## {
##     n_row_max <- n_row_inc <- 10^5
##     res <- matrix(NA_real_, nrow = n_row_inc, ncol = 4 + n_vars,
##         dimnames = list(NULL, c("id", "tstart", "tstop", "event",
##             paste0("x", 1:n_vars))))
##     cur_row <- 1
##     if (re_draw) {
##         get_unif_draw(re_draw = T)
##         get_exp_draw(re_draw = T)
##         get_norm_draw(re_draw = T)
##     }
##     use_intercept <- !missing(intercept_start)
##     betas <- matrix(get_norm_draw((t_max - t_0 + 1) * (n_vars +
##         use_intercept)), ncol = n_vars + use_intercept, nrow = t_max -
##         t_0 + 1)
##     betas <- t(t(betas) * sds)
##     betas[1, ] <- if (use_intercept)
##         c(intercept_start, rep(beta_start, n_vars))
##     else beta_start
##     betas <- apply(betas, 2, cumsum)
##     for (id in 1:n_series) {
##         tstart <- tstop <- t_0
##         repeat {
##             tstop <- tstart + get_exp_draw(1) + 1
##             x_vars <- x_range * get_unif_draw(n_vars) - x_range/2 +
##                 x_mean
##             l_x_vars <- if (use_intercept)
##                 c(1, x_vars)
##             else x_vars
##             tmp_t <- tstart
##             while (tmp_t < tstop && tmp_t < t_max) {
##                 exp_eta <- exp((betas[floor(tmp_t - t_0) + 2,
##                   ] %*% l_x_vars)[1, 1])
##                 event <- exp_eta/(1 + exp_eta) > get_unif_draw(1)
##                 if (event) {
##                   tstop <- tmp_t + 1
##                   break
##                 }
##                 tmp_t <- tmp_t + 1
```

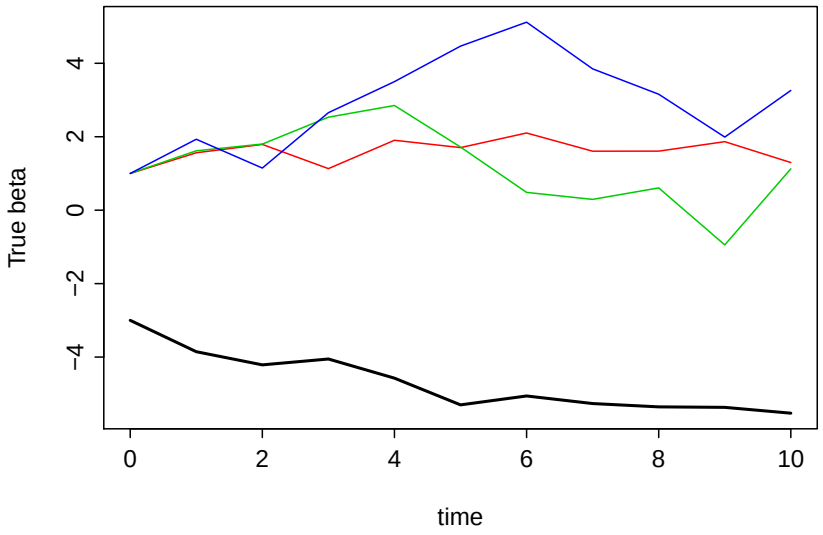
```
##             }
##             res[cur_row, ] <- c(id, tstart, tstop, event, x_vars)
##             if (cur_row == n_row_max) {
##                 n_row_max <- n_row_max + n_row_inc
##                 res = rbind(res, matrix(NA_real_, nrow = n_row_inc,
##                   ncol = 4 + n_vars))
##             }
##             cur_row <- cur_row + 1
##             if (event || tstop > t_max)
##                 break
##             tstart <- tstop
##         }
##     }
##     list(res = as.data.frame(res[1:(cur_row - 1), ]), betas = betas)
## }
## <bytecode: 0x000000001a94d650>
```

The function start by simulating $\vec{\beta}_t = \vec{\beta}_{t-1} + \vec{\eta}_t$ where $\vec{\eta}_t \sim N(\vec{0}, \mathbf{Q})$. We then model the death of individual i in period t by $\pi_{it} = \exp(\vec{\beta}^T \vec{x}_{it})/(1 + \exp(\vec{\beta}^T \vec{x}_{it}))$. We update the co-variate vector for the i'th individual with gabs of $1 + z$ where $z \sim \text{Exp}(1)$. We let $x_{itk} \sim \text{Unif}(a, b)$ for given values a and b (if the co-variates are updated in period t for the i'th individual). Below we define a list of arguments for `test_sim_func_logit` and simulate the series:

```
set.seed(20160921) # a resonably productive day
beta_start <- 1
n_vars <- 3
sims_args <- list(n_series = (n_series <- 1e4),
                  n_vars = n_vars,
                  t_0 = (t_0 <- 0),
                  t_max = (t_max <- 10),
                  x_range = (x_range <- 1),
                  x_mean = (x_mean <- 0),
                  beta_start = beta_start,
                  intercept_start = (intercept_start <- -3),
                  re_draw = T,
                  sds = (sds <- c(sqrt(.2), rep(1, n_vars))))

sims <- do.call(test_sim_func_logit, sims_args)
```

We have a total of 10^{4} series, with 3 parameters and an intercept. The simulated parameters are printed below. The thick line is the intercept:

```
matplot(seq_len(nrow(sims$betas)) - 1, sims$betas, type = "l", lty = 1,
        lwd = c(2, rep(1, n_vars)), xlab = "time", ylab = "True beta")
```



Further, we start at time 0 and end at time 10 giving us 10 intervals. The $\vec{\beta}_0$ starts at (-3, 1, 1, 1). We set the variances of state space variables to [0.447, 1, 1, 1]. The variance of the intercept is lower to ensure that it does not wonder of too much. Thus, we end with a lower base line risk of dying with greater certainty Lastly, the co-variates are simulated to be uniformly distributed within [-0.5, 0.5]. The first 10 rows of the final data frame and number of deaths are printed below

```
head(sims$res, 10)
```

```
##    id tstart tstop event      x1      x2      x3
## 1   1   0.00  1.86     0  0.3613  0.1551  0.0154
## 2   1   1.86  2.94     0  0.3424  0.0519 -0.2350
## 3   1   2.94  4.89     0 -0.1973 -0.0359 -0.0277
## 4   1   4.89  6.62     0  0.3989  0.1604  0.0515
## 5   1   6.62  8.67     0 -0.3168  0.0236 -0.3321
## 6   1   8.67  9.71     0  0.1656  0.3840  0.1751
## 7   1   9.71 13.75     0 -0.4571  0.3456 -0.2635
## 8   2   0.00  1.11     0  0.2627 -0.1496 -0.3809
## 9   2   1.11  2.52     0  0.1282 -0.1571  0.0326
## 10  2   2.52  4.49     0 -0.0175  0.3215  0.2799
```

```
sum(sims$res$event) # number of indvidiauls who dies
```

```
## [1] 1991
```

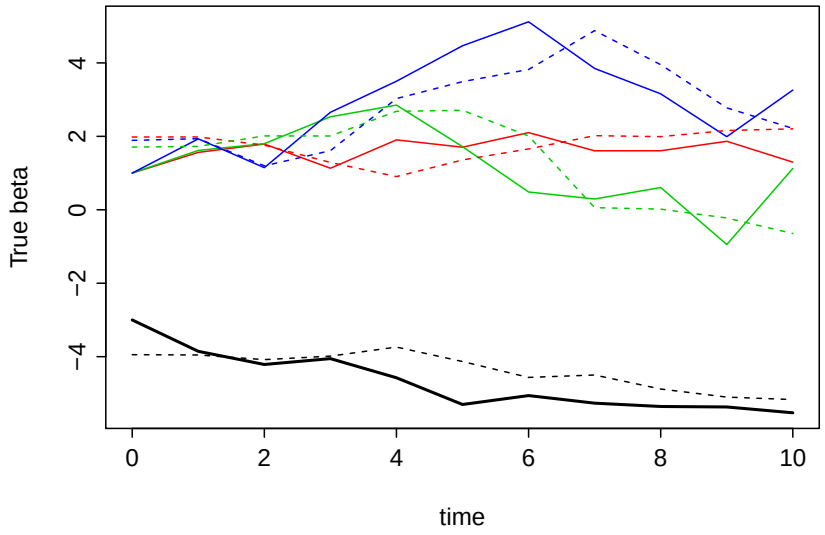
```
arg_list <- list(
  formula = Surv(tstart, tstop, event) ~
            . - tstart - tstop - event - id,
  data = sims$res,
  Q_0 = diag(rep(10, 4)),
  id = sims$res$id,
  method = "EKF",
  max_T = 10,
  by = 1)
```

```
system.time(fit_EKF <- do.call(ddhazard, arg_list))
```

```
## a_0 not supplied. One iteration IWLS of static logit model is used
```

```
##    user  system elapsed
##    1.03    0.08    0.80
```

```
matplot(seq_len(nrow(sims$betas)) - 1, sims$betas, type = "l", lty = 1,
        lwd = c(2, rep(1, n_vars)), xlab = "time", ylab = "True beta")
matplot(seq_len(nrow(sims$betas)) - 1, fit_EKF$a_t_d_s, add = T,
        type = "l", lty = 2)
```

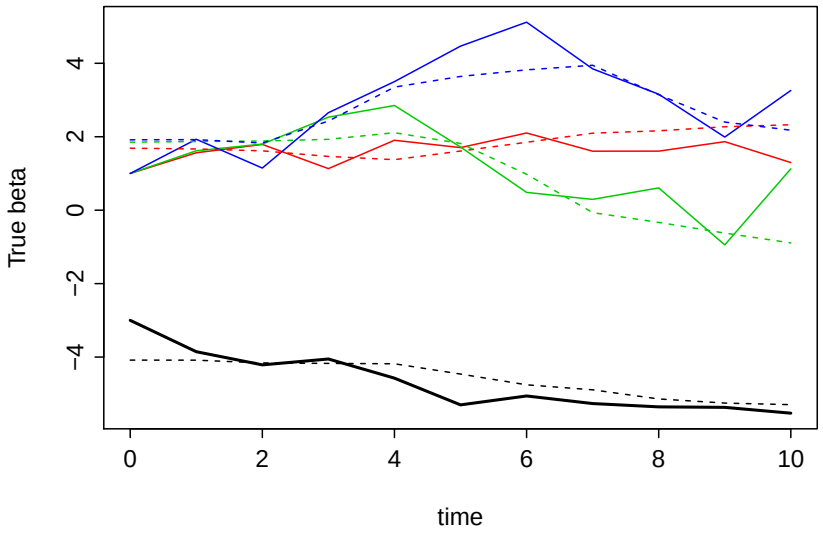



```
arg_list$method <- "UKF"
arg_list$k <- 0

system.time(fit_UKF <- do.call(ddhazard, arg_list))
```

```
## a_0 not supplied. One iteration IWLS of static logit model is used
```

```
##    user  system elapsed
##    2.86    0.24    3.11
```

```
matplot(seq_len(nrow(sims$betas)) - 1, sims$betas, type = "l", lty = 1,
        lwd = c(2, rep(1, n_vars)), xlab = "time", ylab = "True beta")
matplot(seq_len(nrow(sims$betas)) - 1, fit_UKF$a_t_d_s, add = T,
        type = "l", lty = 2)
```




```
fit_EKF$Q
```

```
##             (Intercept)       x1      x2       x3
## (Intercept)      0.0755 -0.05158  0.0349  0.07438
## x1              -0.0516  0.13416 -0.1394 -0.00129
## x2               0.0349 -0.13942  0.6038 -0.10726
## x3               0.0744 -0.00129 -0.1073  0.77904
```

```
fit_UKF$Q
```

```
##             (Intercept)      x1      x2      x3
## (Intercept)      0.0911 -0.0662  0.1871  0.0335
## x1              -0.0662  0.0946 -0.2091 -0.0524
## x2               0.1871 -0.2091  0.7430  0.0937
## x3               0.0335 -0.0524  0.0937  0.8657
```