Bootstrap illustration

Benjamin Christoffersen

2017-06-17

Introduction

This vignette will show how to bootstrap the confidence intervals of a `ddhazard` call. This vignette builds on the vignettes 'ddhazard' and 'Comparing methods for time varying logistic models'. Thus, it is recommended to read these first. You can get the version used to make this vignette by calling:'

```
current_version # The string you need to pass devtools::install_github
```

```
## [1] "boennecd/dynamichazard@c579fc49ceca796b281478a364c57a58b13e43cb"
```

```
devtools::install_github(current_version)
```

You can also get the latest version on CRAN by calling:

```
install.packages("dynamichazard")
```

PBC data set

We start by settings up the data set. We will use the pbc2 data set from the `survival` package as in the vignette 'Comparing methods for time varying logistic models':

```
# PBC data set from survival with time variying covariates
# Details of tmerge are not important in this scope. The code is included
# to make you able to reproduce the results
# See: https://cran.r-project.org/web/packages/survival/vignettes/timedep.pdf
library(survival)
temp <- subset(pbc, id <= 312, select=c(id, sex, time, status, edema, age))
pbc2 <- tmerge(temp, temp, id=id, death = event(time, status))
pbc2 <- tmerge(pbc2, pbcseq, id=id, albumin = tdc(day, albumin),
               protime = tdc(day, protime), bili = tdc(day, bili))
pbc2 <- pbc2[, c("id", "tstart", "tstop", "death", "sex", "edema",
                 "age", "albumin", "protime", "bili")]
```

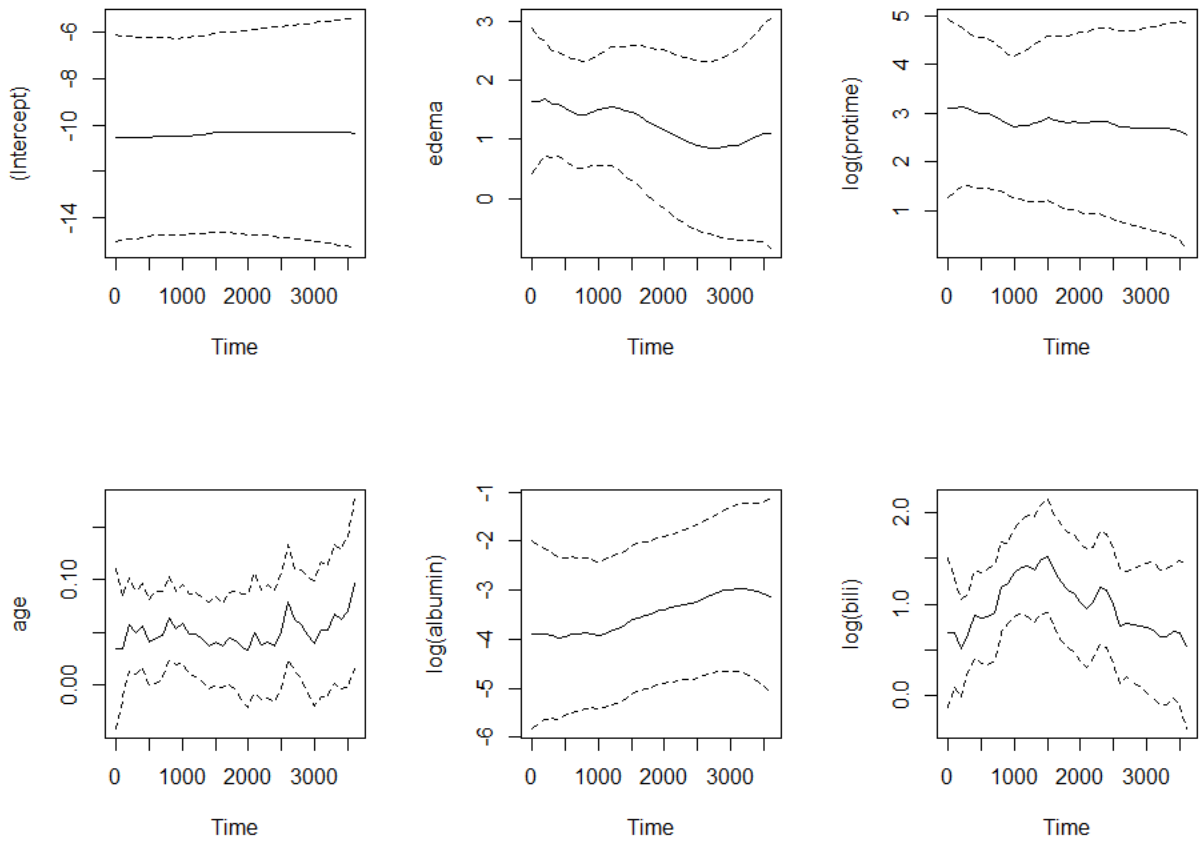
Next, we fit the model as in the vignette 'Comparing methods for time varying logistic models':

```
library(dynamichazard)
dd_fit <- ddhazard(Surv(tstart, tstop, death == 2) ~ age + edema +
                       log(albumin) + log(protime) + log(bili), pbc2,
                   id = pbc2$id, by = 100, max_T = 3600,
                   Q_0 = diag(rep(10000, 6)), Q = diag(rep(0.001, 6)),
                   control = list(save_risk_set = T, save_data = T, eps = .1))
```

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

A plot of the estimates is given below. The dashed lines are 95% point-wise confidence intervals using the variances estimates from the Extended Kalman filter with smoothing:

```
plot(dd_fit)
```



Sampling individuals

We can bootstrap the estimates in the model by using the `ddhazard_boot` function as done below:

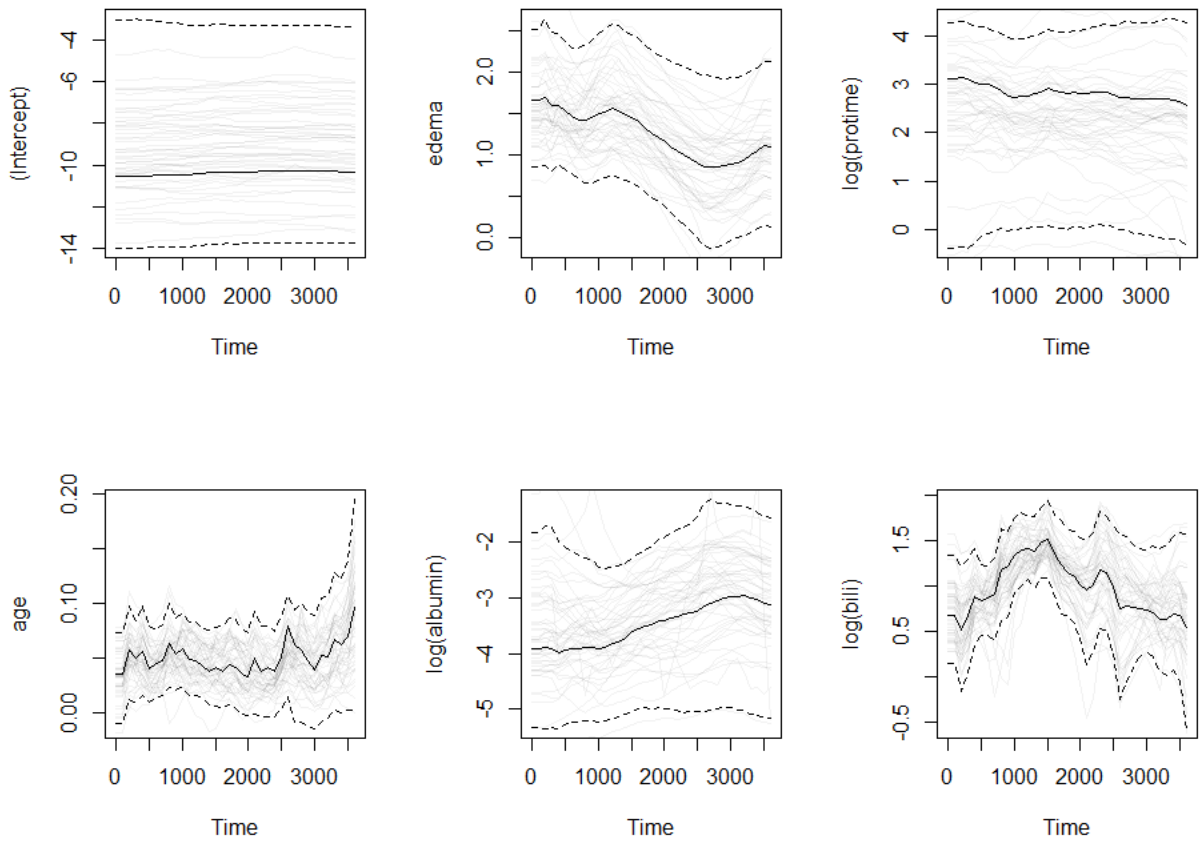
```
set.seed(7451)
R <- 10000
boot_out <- ddhazard_boot(
  dd_fit,
  do_sample_weights = F,       # should re-sampeling be by weights or by
                               # sampling each individual discreetly
  R = R                        # Number of bootstrap samples
  )

# The list has the same structure and class as the list returned by boot::boot
# Though, a few elements are added
class(boot_out)
```

```
## [1] "ddhazard_boot" "boot"
```

Above, we bootstrap the model by sampling the individuals. I.e. individuals will have weights of $0, 1, 2, \ldots$ in the estimation. We can plot 95% confidence bounds from the bootstrap coefficients with the Percentile Bootstrap method as follows:

```
plot(dd_fit, ddhazard_boot = boot_out, level = 0.95)
```

```
## Only plotting 50 of the boot sample estimates
```



The completely black line is the original estimates, the dashed lines are 2.5% and 97.5% quantiles of the bootstrap coefficient taken at each point and the transparent black lines each represent a bootstrap estimate. Linear interpolation on the normal quantile scale is used if we do not have a quantile that match exactly.

Strata

You can provide a strata variable to perform stratified sampling with. This is done by setting the `strata` argument in the call to `ddhazard_boot`. Notice that this has to be on an individual level (one indicator variable per individual) not observation level (not one indicator variable per row in the data set). Further, you can use the `unique_id` argument to match the individual entries with the entries in `strata`. As an example, we stratify by the `age` at the start of the study period with the code below:

```
# Individuals have different number of rows in the dataset
xtabs(~xtabs(~pbc2$id))
```

```
## xtabs(~pbc2$id)
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14
## 27 27 34 48 32 30 18 22 21 22  9  9  8  5
```

```r
# Though all the individual have the same age for all periods
# This age is the age at the start of the study
unique(tapply(pbc2$age, pbc2$id, function(x) length(unique(x))))
```

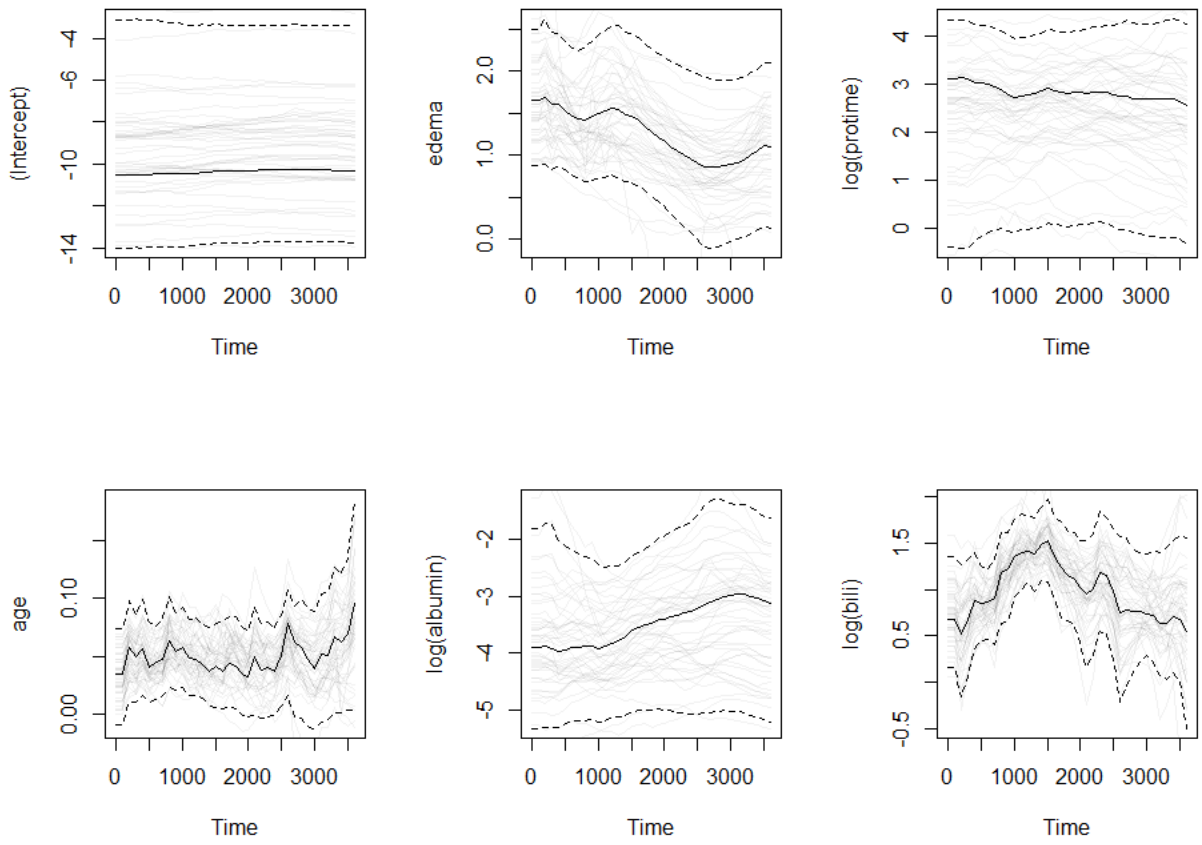
```
## 1
## 1
```

```r
# Next, we find the age for each individual
unique_id <- unique(pbc2$id)
age <- sapply(unique_id, function(x) pbc2$age[pbc$id == x][1])
summary(age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   33.63   44.52   52.04   50.69   56.22   70.56
```

```r
# We define a strata variable for those less than age 50
is_less_than_50 <- age < 50

# We perform stratified sampling over this variable as follows
set.seed(101)
boot_out_with_strata <- ddhazard_boot(
  dd_fit,
  unique_id = unique_id,
  strata = is_less_than_50,
  R = R)

plot(dd_fit, ddhazard_boot = boot_out_with_strata)
```

```
## Only plotting 50 of the boot sample estimates
```



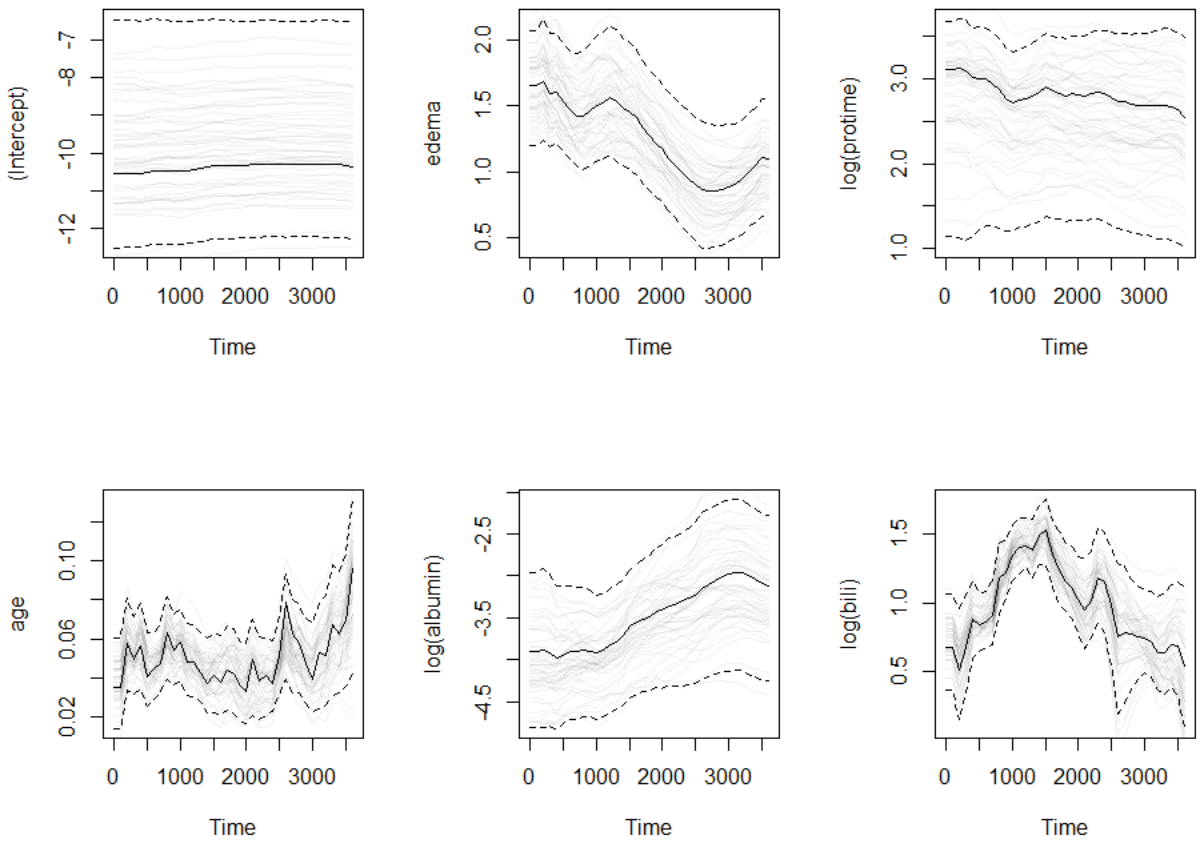
The above code is only provided for illustrative purposes. There is no reason to do stratified sampling over the age variable (as far as I gather). However, it may be useful if you have e.g. categorical variables in your model and want to ensure that each bootstrap sample has a given amount of observation in each category

Sampling weights

We can also sample the weights. This is done as follows: within each stratum j (e.g. males or females) let r_j denote the number of individuals. Then we sample r_j uniform variables $l_i \sim \text{Unif}(0, 1)$ for $i = 1, \ldots, r_j$ and normalize with a constant c such that $\sum_{i=1}^{r_j} l_i/c = r_j$. The code below will sample the weights as described above:

```
set.seed(401)
boot_out_by_weights <- ddhazard_boot(
  dd_fit,
  do_sample_weights = T, # changed
  R = R)

plot(dd_fit, ddhazard_boot = boot_out_by_weights)
```

```
## Only plotting 50 of the boot sample estimates
```



Fixed effects

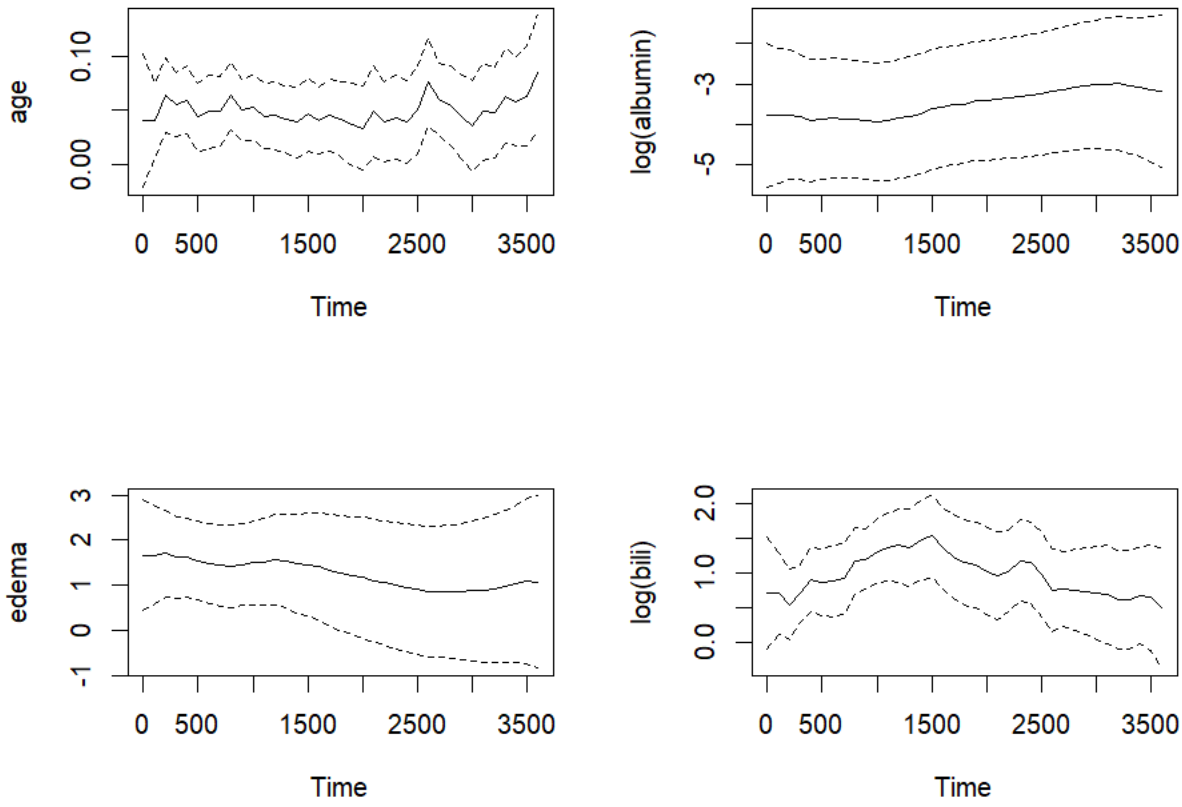
Fixed effects (time invariant effects) can also be bootstrap to get confidence bounds. The fixed effects bootstrap coefficients are added as the last entries of the element `t` of the returned object by `ddhazard_boot`. As an example we will estimate a model below where `log(protime)` and the intercept are fixed

```
dd_fit <- ddhazard(Surv(tstart, tstop, death == 2) ~
                     ddFixed(1) + ddFixed(log(protime)) # changed to fixed
                     + age + edema + log(albumin) +  + log(bili), pbc2,
                   id = pbc2$id, by = 100, max_T = 3600,
                   Q_0 = diag(rep(10000, 4)), Q = diag(rep(0.001, 4)),
                   control = list(
                     save_risk_set = T, save_data = T, eps = .1,
                     fixed_terms_method = "E_step")      # Fixed effects are
                                                         # esimated in E-step
                   )
```

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

The time varying effects are plotted below:

```
plot(dd_fit)
```



The fixed effects are estimated to:

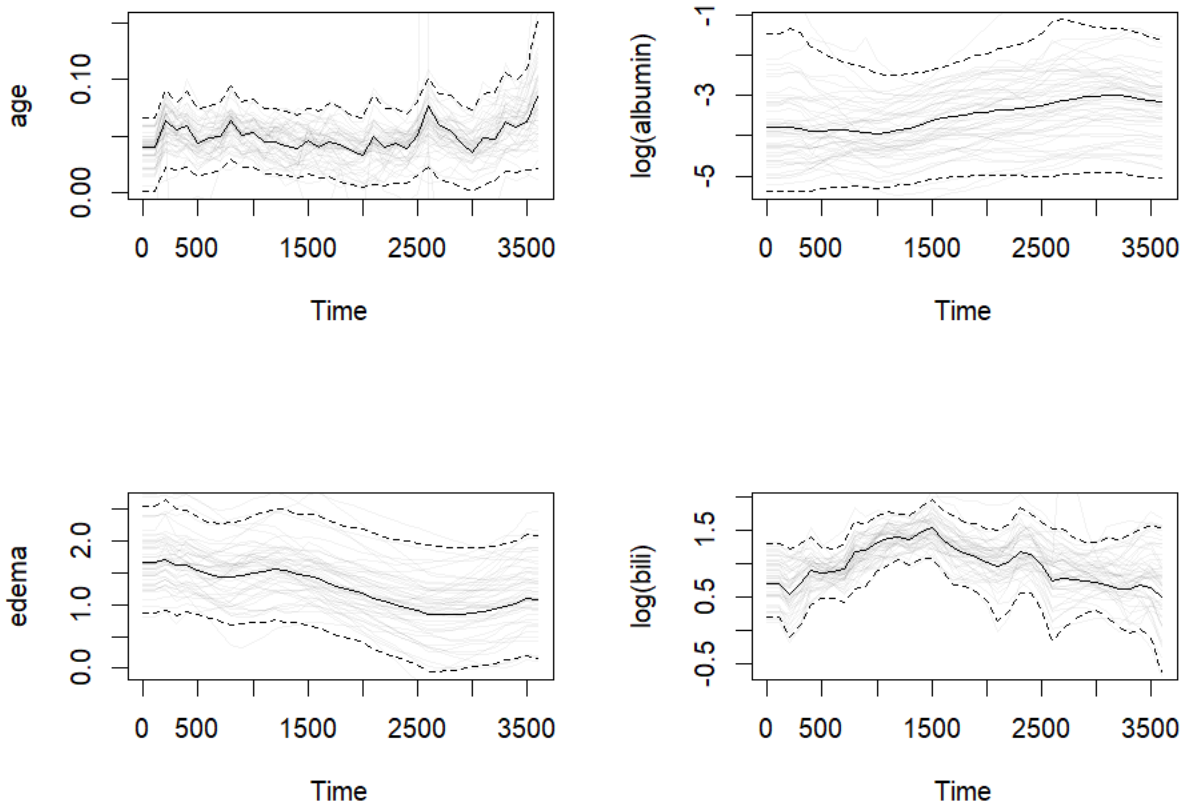
```
dd_fit$fixed_effects
```

```
##  (Intercept) log(protime)
##   -10.449270     2.859554
```

We can bootstrap the estimates with a call similar to those we made before:

```
set.seed(9001)
boot_out <- ddhazard_boot(
  dd_fit,
  do_sample_weights = F,  # dont sample weights
  R = R)

# Plot time varying effects
plot(dd_fit, ddhazard_boot = boot_out)
```

```
## Only plotting 50 of the boot sample estimates
```



We then turn the bootstrap confidence intervals of the fixed effects. These can be computed with the `boot.ci` function from the `boot` library as shown below:

```
library(boot)

# We start by printing confidence intervals for
colnames(boot_out$t)[ncol(boot_out$t) - 1]
```

```
## [1] "(Intercept)"
```

```
boot.ci(boot_out, index = ncol(boot_out$t) - 1,
        # We specify the types of confidence intervals estimates here:
        type = c(
          "norm",  # A matrix of intervals calculated using the normal
                   # approximation.
          "basic", # The intervals calculated using the basic bootstrap method.
          "perc",  # The intervals calculated using the bootstrap percentile
                   # method.
          "bca")   # The intervals calculated using the adjusted bootstrap
                   # percentile (BCa) method.
        )
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
```

```
## CALL :
## boot.ci(boot.out = boot_out, type = c("norm", "basic", "perc",
##     "bca"), index = ncol(boot_out$t) - 1)
##
## Intervals :
## Level      Normal              Basic
## 95%   (-38.02,  14.08 )   (-17.76,  -7.11 )
##
## Level     Percentile            BCa
## 95%   (-13.79,  -3.15 )   (-16.66,  -7.05 )
## Calculations and Intervals on Original Scale
```

```
# Then we print confidence intervals for
colnames(boot_out$t)[ncol(boot_out$t)]
```

```
## [1] "log(protime)"
```

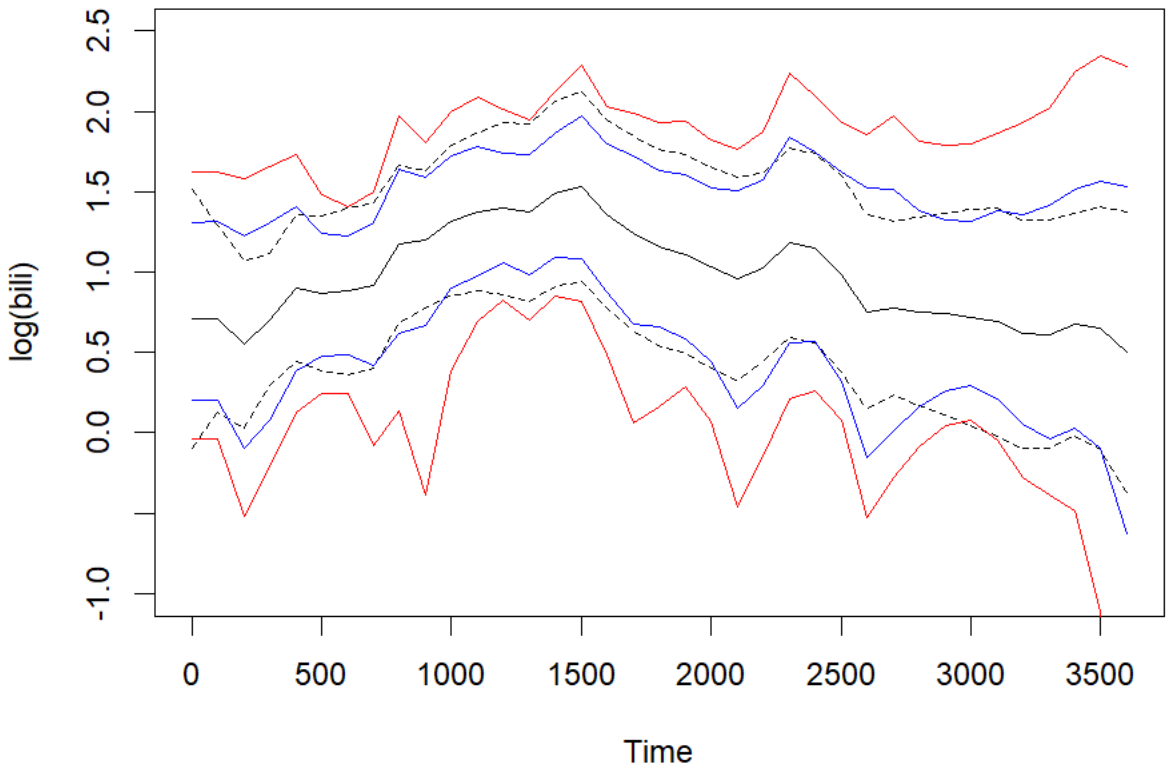
```
boot.ci(boot_out, index = ncol(boot_out$t) - 0, type = c(
  "norm", "basic", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_out, type = c("norm", "basic", "perc",
##     "bca"), index = ncol(boot_out$t) - 0)
##
## Intervals :
## Level      Normal              Basic
## 95%   (-0.682,  7.466 )   ( 1.703,  5.710 )
##
## Level     Percentile            BCa
## 95%   ( 0.001,  4.008 )   ( 1.681,  5.004 )
## Calculations and Intervals on Original Scale
```

Boot envelope

We may also want to get simultaneous confidence intervals. An easy way to get such confidence intervals is with the `envelope` function in the `boot` library. For instance, we can simultaneous confidence intervals for the `bili` covariate as follows:

```
# Find the indices that correspondents to the log(bili) variable
is_bili_coef <- grep("^log\\(bili\\):", colnames(boot_out$t))

# Use the envelope
envelopes <- envelope(boot_out, level = 0.95 ,index = is_bili_coef)

# Plot curves
plot(dd_fit, cov_index = 4, ylim = c(-1, 2.5))
lines(dd_fit$times, envelopes$point[1, ], col = "blue")
lines(dd_fit$times, envelopes$point[2, ], col = "blue")

lines(dd_fit$times, envelopes$overall[1, ], col = "red")
lines(dd_fit$times, envelopes$overall[2, ], col = "red")
```



The dashed black lines are from the smoothed covariance matrix. The blue lines are pointwise confidence intervals using the percentile method from the `envelope` function. The red line is the simultaneous confidence bounds using the envelope method in equation (4.17) of Davison & Hinkley (1997). The latter curves are formed by creating an envelope over each of the pointwise confidence intervals and hence the name

How good is the coverage

In this section, we will test the coverage of the pointwise confidence intervals using the smoothed covariance matrix and the bootstrap percentile method. We will test these in a simulation study where:

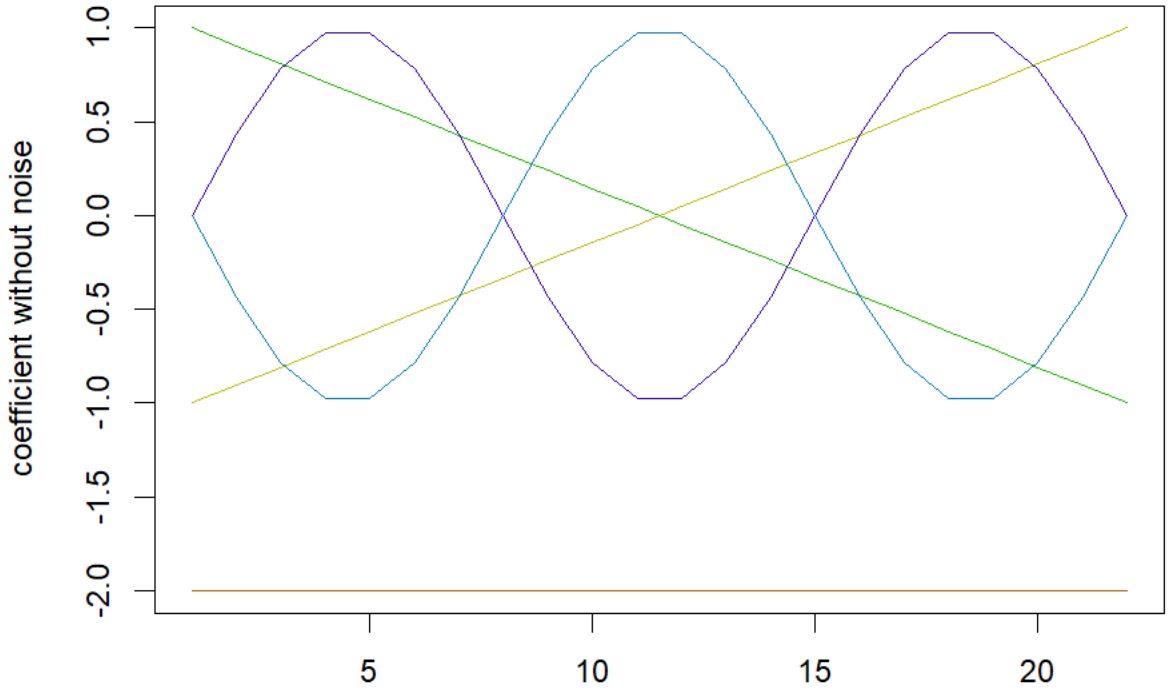
- The coefficients are drifting deterministically with a some normal noise added to them
- Individuals have time invariant covariates

The simulation is to mimic a situation where we assume that the coefficients are not random (as the model implies) but we do not know the shape of the coefficient curves across time. We setup the parameters for the experiment below and plot the coefficients without noise:

```
tmax <- 22                            # Number of periods
n_start_grps <- 3                     # Number of "start group" - see text
                                      # Number of multiple of tmax - 1 in each
mlt <- 30                             # start group
n <- (tmax - 1) * mlt * n_start_grps # Total number of individuals
n
```

```
## [1] 1890
```

```r
# Define the noise free coefficients
beta <- cbind(
  x1 = rep(-2, (tmax - 1) + 1),
  x2 = (0:(tmax - 1) - (tmax - 1)/2) / ((tmax - 1) / 2),
  x3 = ((tmax - 1):0 - (tmax - 1)/2) / ((tmax - 1) / 2),
  x4 = - sin(pi / 7 * (0:(tmax - 1))),
  x5 = sin(pi / 7 * (0:(tmax - 1))))

# Plot noise free coefficients
cols <- c("#BC5C00", "#BEBE00", "#23BC00", "#0082BC", "#3500C1")
matplot(beta, type = "l", lty = 1, ylab = "coefficient without noise",
        col = cols)
```



There will be a total of `n = 1890` individuals in groups of three. We start observing each group at time `0`, `7` and `14`. We do so to have a "stable" number of individual through the experiment. The experiment ends after `tmax = 22`.

We add a bit of normally distributed noise to the coefficients with mean zero and standard deviation 0.1. The individuals' covariates are simulated from the uniform distribution from the range [-1, 1]. The function `sim_func` is used to make the simulation. The definition of the function can be found in the markdown file for this vignette on the github site. We simulate a series below, illustrate the data matrix and plot the coefficients with noise added to them:

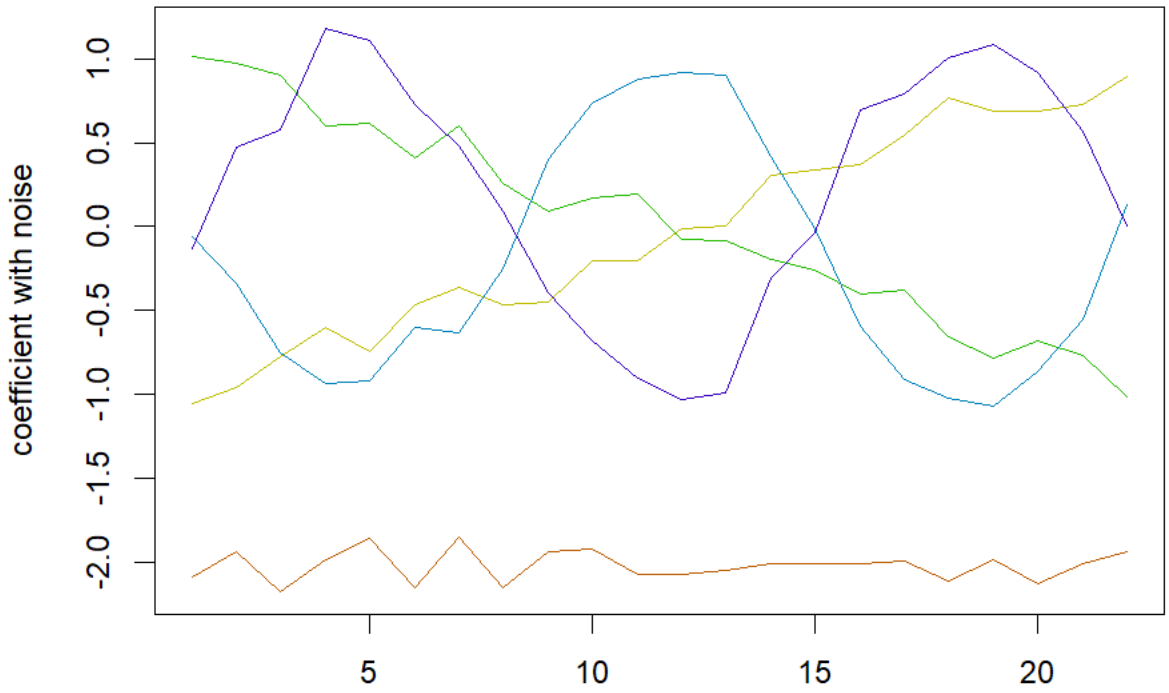
```r
# Simulate
set.seed(122044)
sim_list <- sim_func()

# Show data matrix
head(sim_list$sims, 10)
```

```
##    id tstart tstop x1    x2    x3     x4    x5  eta  dies
## 1   1     14    16  1  0.18 -0.64  0.195  0.14 -1.8  TRUE
## 2   2     14    22  1  0.76 -0.14 -0.882 -0.80 -1.7 FALSE
## 3   3     14    16  1  0.58  0.48  0.921  0.50 -2.0  TRUE
## 4   4     14    22  1 -0.78  0.69  0.995  0.29 -2.5 FALSE
## 5   5     14    22  1 -0.51  0.80  0.290  0.79 -2.4 FALSE
## 6   6     14    22  1 -0.27  0.62  0.962  0.94 -2.3 FALSE
## 7   7     14    19  1 -0.84  0.96  0.816 -0.50 -2.5  TRUE
## 8   8     14    22  1 -0.58  0.62  0.969 -0.34 -2.4 FALSE
## 9   9     14    15  1  0.70 -0.87  0.763 -0.70 -1.5  TRUE
## 10 10     14    17  1  0.59  0.23  0.063 -0.20 -1.9  TRUE
```

```r
tail(sim_list$sims, 10)
```

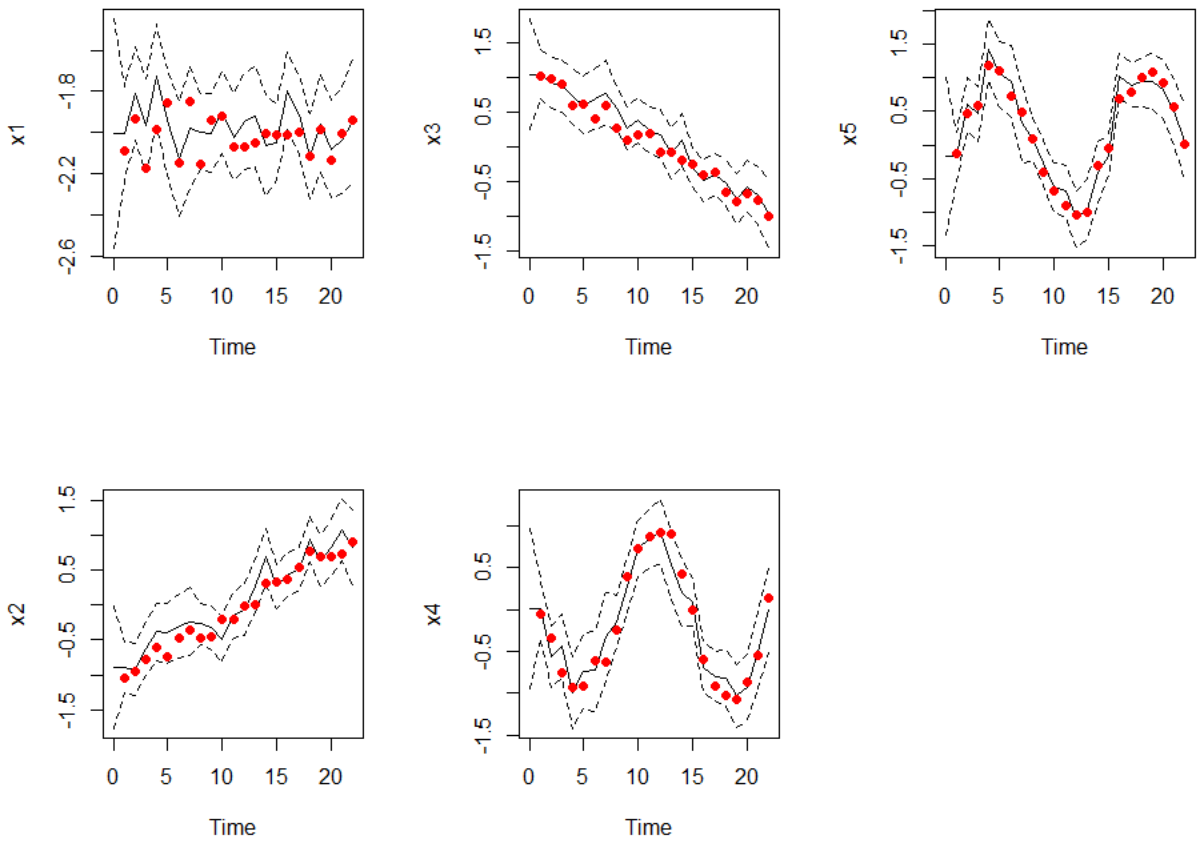
```
##        id tstart tstop x1    x2     x3     x4    x5   eta dies
## 1881 1881      0     3  1  0.48  0.728 -0.834  0.91 -1.92 TRUE
## 1882 1882      0    12  1 -0.66 -0.385  0.084 -0.97 -1.66 TRUE
## 1883 1883      0     3  1  0.25 -0.908  0.246  0.40 -3.34 TRUE
## 1884 1884      0    10  1  0.92 -0.243 -0.544 -0.98 -3.15 TRUE
## 1885 1885      0    16  1 -0.93  0.514 -0.878 -0.65 -0.45 TRUE
## 1886 1886      0    12  1 -0.90 -0.230  0.744  0.38 -1.47 TRUE
## 1887 1887      0     2  1 -0.21 -0.393 -0.621 -0.71 -2.14 TRUE
## 1888 1888      0    10  1 -0.50 -0.088  0.352 -0.87 -1.56 TRUE
## 1889 1889      0     5  1 -0.29  0.123 -0.142  0.55 -1.72 TRUE
## 1890 1890      0    14  1  0.83  0.294  0.643  0.02 -2.71 TRUE
```

```r
# Plot coefficients with noise
matplot(sim_list$beta_w_err, type = "l", lty = 1, ylab = "coefficient with noise",
        col = cols)
```



We are now able to estimate the model as follows:

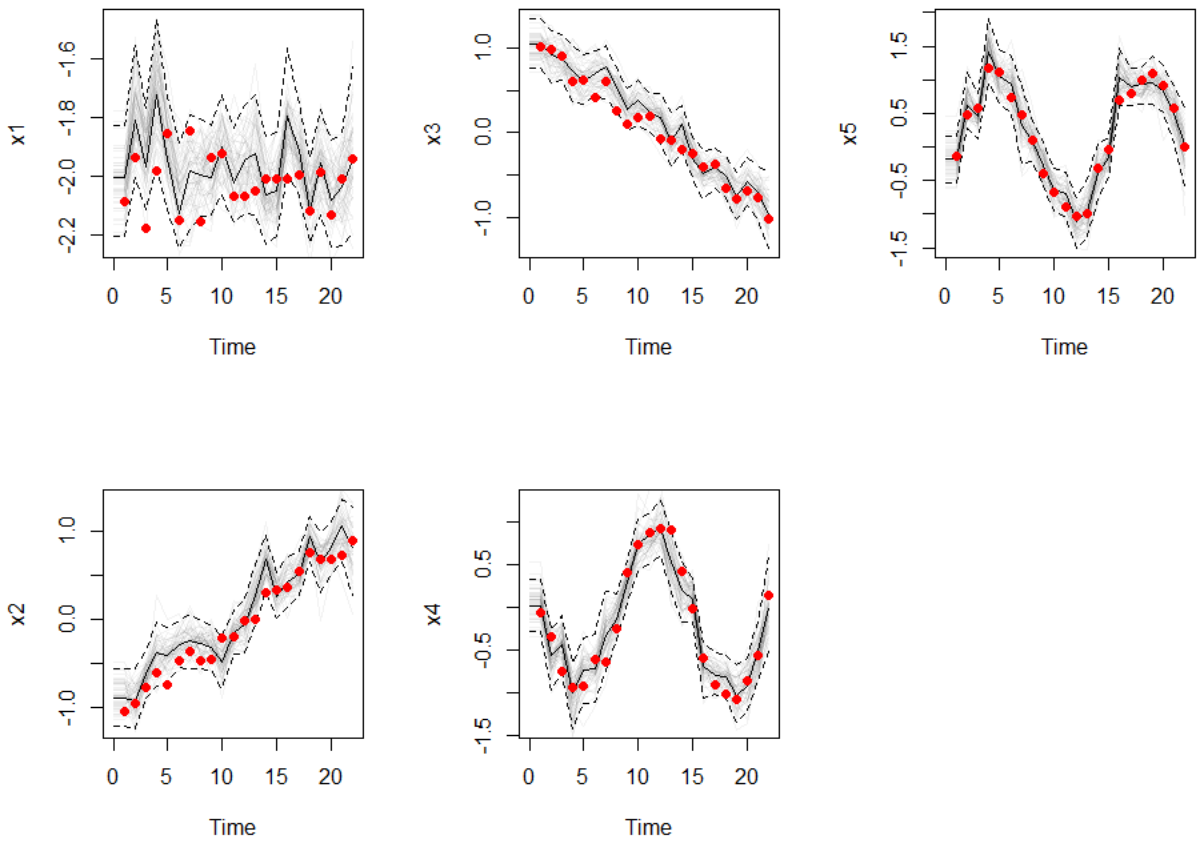
```
# Estimate model
fit_expression <- expression({
  fit <- ddhazard(Surv(tstart, tstop, dies) ~ -1 + x1 + x2 + x3 + x4 + x5,
                  data = sim_list$sims, id = sim_list$sims$id, max_T = tmax,
                  by = 1, Q_0 = diag(1e4, 5), Q = diag(1, 5),
                  a_0 = rep(0, 5), control = list(
                    denom_term = 1e-3, eps = .01, criteria = "delta_likeli"))
})
eval(fit_expression)

# Plot estimates with pointwise confidence bounds from smoothed covariance
# matrix
for(i in 1:5){
  plot(fit, cov_index = i)
  points(sim_list$beta_w_err[, i], pch = 16, col = "red")
}
```



The plots shows the estimated coefficient with 95% pointwise confidence intervals from the smoothed covariance matrix. The dots are the actual values (i.e. those with noise added to them). A bootstrap estimate of the confidence bounds is made below:

```
# Bootstrap with resampling individuals
boot_out <- ddhazard_boot(fit,
                          do_sample_weights = F, R = 999,
                          do_stratify_with_event = F)

# Plot estimated confidence bounds
for(i in 1:5){
  plot(fit, cov_index = i, ddhazard_boot = boot_out)
  points(sim_list$beta_w_err[, i], pch = 16, col = "red")
}
```

```
## Only plotting 50 of the boot sample estimates
## Only plotting 50 of the boot sample estimates
## Only plotting 50 of the boot sample estimates
## Only plotting 50 of the boot sample estimates
## Only plotting 50 of the boot sample estimates
```



We can now pose the question how the pointwise coverage is for each coefficient. For this reason, we have defined the function `compute_coverage` which is not included but can be found in the markdown for this vignette on the github site:

```
compute_coverage(fit, boot_out, sim_list$beta_w_err)
```

```
## $smooth
##        x1        x2        x3        x4        x5
## 0.9090909 1.0000000 1.0000000 1.0000000 0.9545455
##
## $boot
##        x1        x2        x3        x4        x5
## 0.8181818 0.9090909 0.9545455 0.9545455 1.0000000
```

`compute_coverage` outputs a list of the true coverage of the 95% confidence intervals from the smoothed covariance matrix and the percentile method from the bootstrap. That is, the fractions of red dots from the previous plot that are within the 95% confidence interval. The two elements of the list is for the the percentile method from the bootstrap. These are respectively the `smooth` and `boot` elements of the list. We can now repeat the above `M` times (defined below) as follows:

```
set.seed(520920)
R <- 999  # Number of bootstrap estimates in each trials
M <- 100  # Number of trials

# Define matrix for output
```

```r
coverage_boot <- coverage_smooth <- matrix(
  NA_real_, nrow = M, ncol = ncol(fit$state_vecs))

# Sometimes estimations fails. We use this counter to keep track of the number
# of times
n_fails <- 0
LRs <- 1.1^(0:(-6)) # Learning rates to try in order to get a fit

# We save this as an expression as we will re-run it later
boot_exp <- expression({
  ########
  # Progress bar for inpatient people (me)
  pb <- winProgressBar(
    "Running simulation", "", 0, M, 50)
  ########

  for(i in 1:M){
    ########
    info <- sprintf("%.2f%% done", 100 * (i - 1) / M)
    setWinProgressBar(pb, i - 1, "Running simulation", info)
    ########

    # Simulate data set
    sim_list <- sim_func()

    # Fit on whole data set
    did_succed <- F
    try({
      eval(fit_expression)
      did_succed <- T
    })
    if(!did_succed){
      n_fails <- n_fails + 1
      next
    }

    # Bootstrap fits
    boot_out <- ddhazard_boot(fit,
                              strata = as.factor(sim_list$sims$tstart),
                              do_stratify_with_event = F,
                              do_sample_weights = F, R = R,
                              LRs = LRs)

    # Compute coverage and add to output
    coverage <- compute_coverage(fit, boot_out, sim_list$beta_w_err)
    coverage_smooth[i, ] <- coverage$smooth
    coverage_boot[i, ] <- coverage$boot
  }

  ########
  close(pb)
  ########
```

```
  })
eval(boot_exp)
```

```
## NULL
```

```
n_fails # number of failed estimations
```

```
## [1] 0
```

The mean of the fraction of the overages for the two methods are printed below. That is, the mean of the fraction for each coefficient from each run that did not fail:

```
colMeans(coverage_smooth, na.rm = T)
```

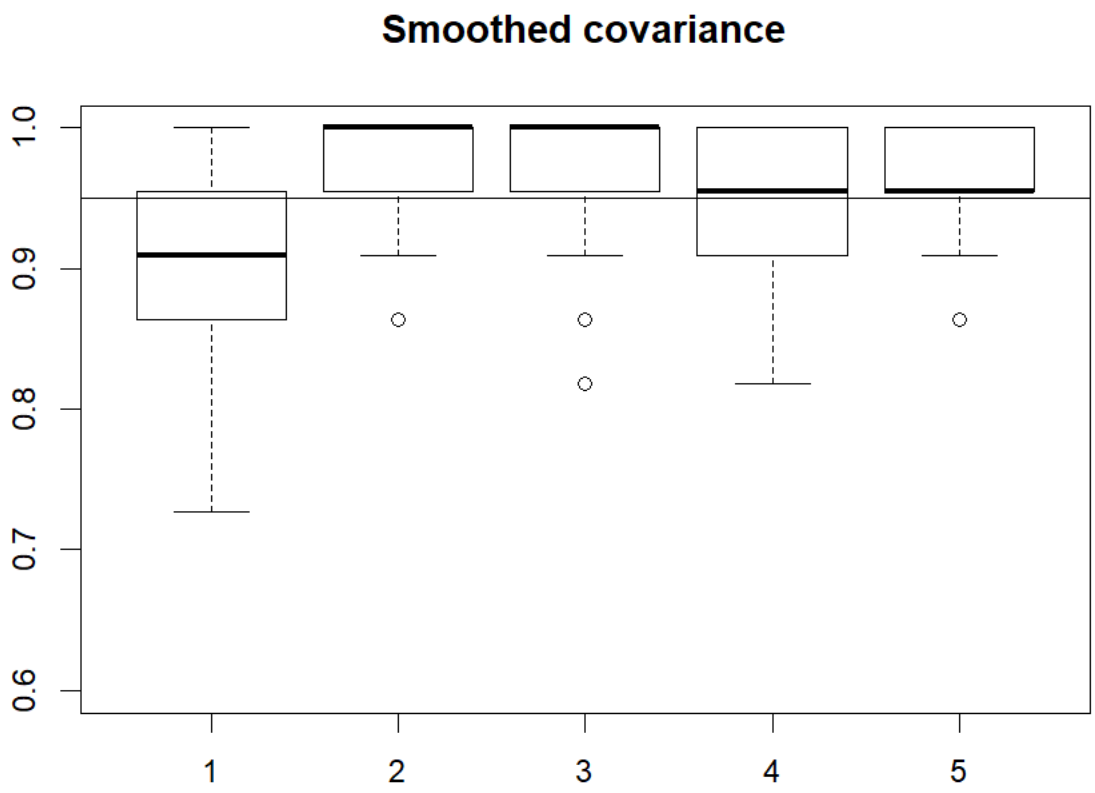
```
## [1] 0.9090909 0.9695455 0.9681818 0.9613636 0.9622727
```

```
colMeans(coverage_boot, na.rm = T)
```

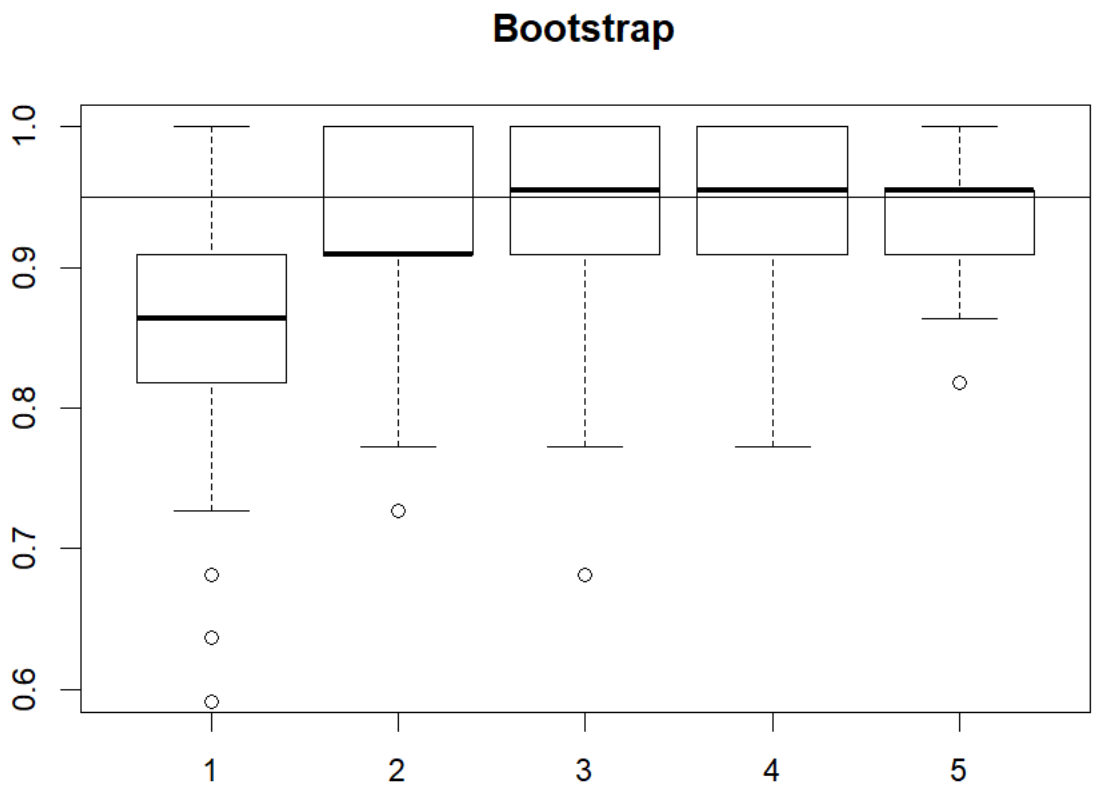
```
## [1] 0.8390909 0.9277273 0.9340909 0.9313636 0.9354545
```

Finally, we can make a boxplot of the fraction of coverage in each trail as follows:

```
boxplot(coverage_smooth, ylim = c(.6, 1), main = "Smoothed covariance")
abline(h = .95, lty = 1)
```




```
boxplot(coverage_boot, ylim = c(.6, 1), main = "Bootstrap")
abline(h = .95, lty = 1)
```



We do alter the learning rate in the previous simulation in order to get a fit when we bootstrap. An alternative could be not to allow for this as done below where failed fits are excluded:

```
n_fails <- 0
LRs <- 1     # Changed to one value only

eval(boot_exp)
```

```
## NULL
```

```
n_fails # number of failed estimations
```

```
## [1] 0
```

The means and box plot are given below:

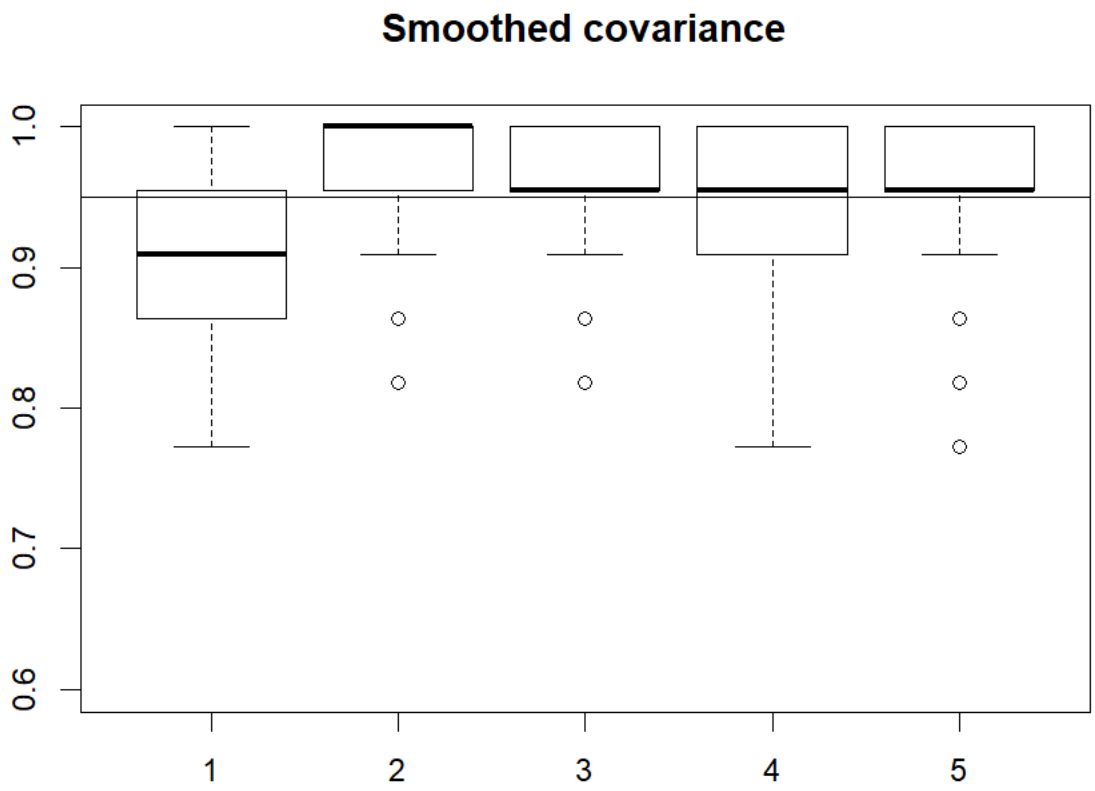
```
colMeans(coverage_smooth, na.rm = T)
```

```
## [1] 0.9054545 0.9668182 0.9636364 0.9536364 0.9577273
```

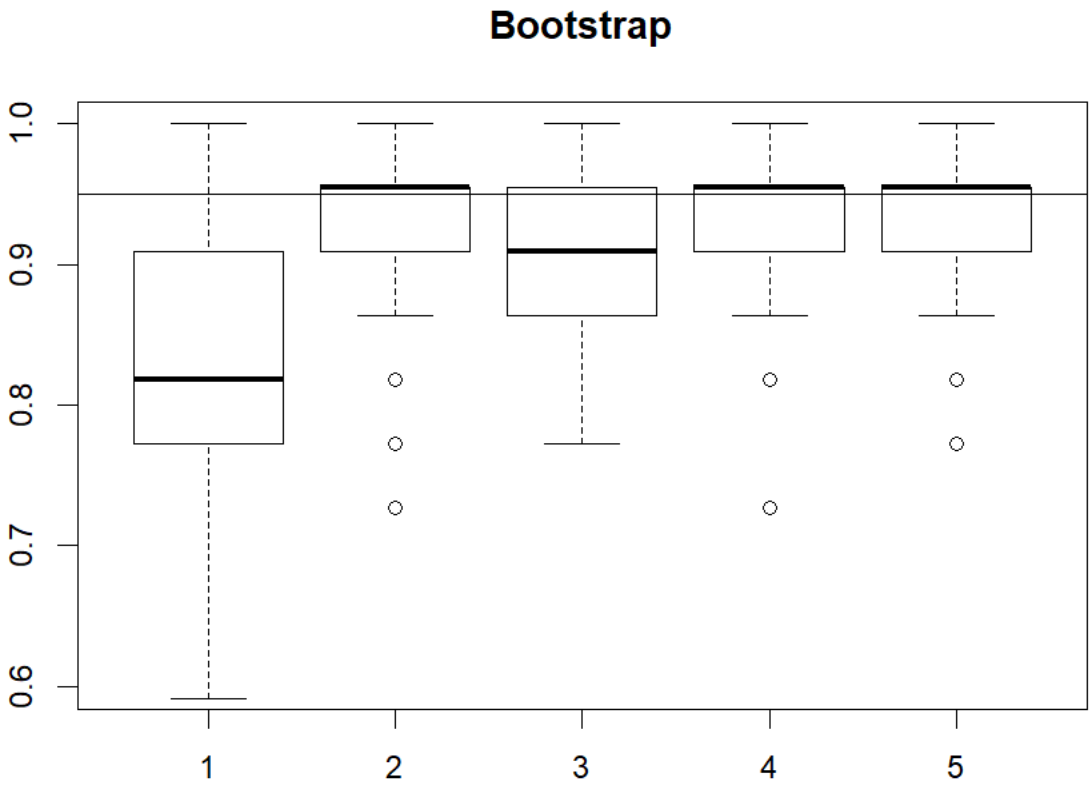
```
colMeans(coverage_boot, na.rm = T)
```

```
## [1] 0.8336364 0.9290909 0.9145455 0.9272727 0.9272727
```

```
boxplot(coverage_smooth, ylim = c(.6, 1), main = "Smoothed covariance")
abline(h = .95, lty = 1)
```




```
boxplot(coverage_boot, ylim = c(.6, 1), main = "Bootstrap")
abline(h = .95, lty = 1)
```



References

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap methods and their application* (Vol. 1). Cambridge university press.