Bootstrap illustration

Benjamin Christoffersen

2018-09-16

Introduction

This vignette will show how to use bootstrap a `ddhazard` object. It is recommended to read or skim `vignette("ddhazard", "dynamichazard")` 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@83d73fbcde352e023c4ea89259f9cbdab9e5e409"
```

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

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

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

TRACE

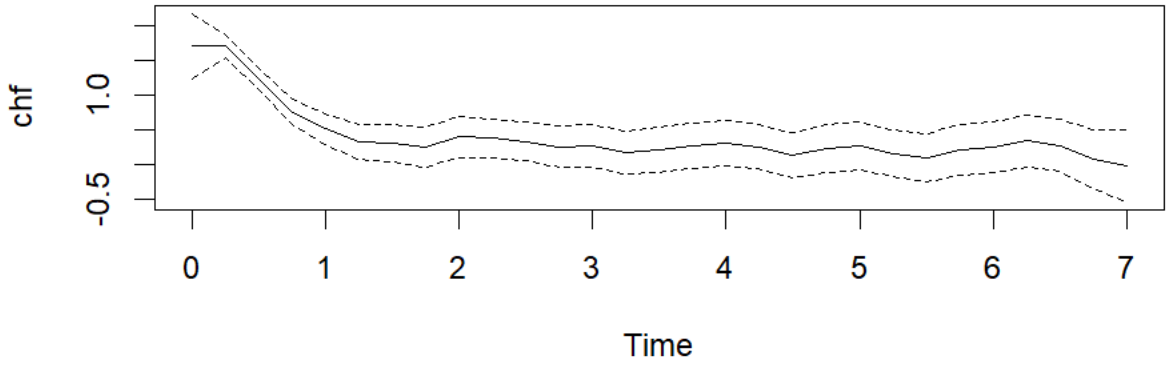
We will use the `TRACE` data set from the `timreg` package. See `?TRACE` for a description of the dataset and `?timereg::aalen` for a additive hazard models used with data set – at least in version 1.9.1. Some of them are (kinda) like the models we fit next in the sequal. The `TRACE` data set is used here to illustrate the bootstrap methods and not as example of how to analysis the data set. I have not looked at the details of the data set or the model fits. We fit the model as follows:

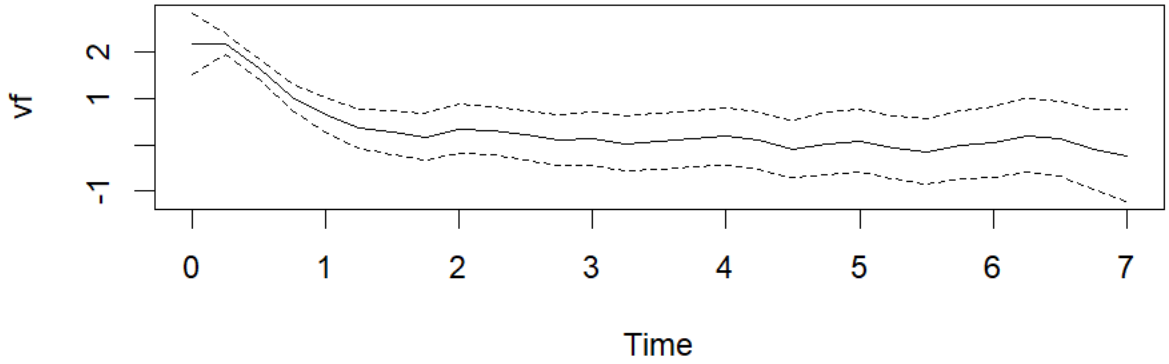
```
library(dynamichazard)
data(TRACE, package = "timereg")
dd_fit <- ddhazard(
  Surv(time, status == 9) ~ ddFixed_intercept() +
    ddFixed(age) + ddFixed(sex) + ddFixed(diabetes) + chf + vf,
  TRACE, max_T = 7, by = .25, model = "exponential",
  Q_0 = diag(10, 2), Q = diag(.1^2, 2),
  control = ddhazard_control(eps = .001, n_max = 25))
```

```
## a_0 not supplied. IWLS estimates of static glm model is used for random walk models. Otherwise the va
```

We use the exponential arrival times models with the extended Kalman filter (the default) estimation method. 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)
```






```
summary(dd_fit)
```

```
## Call:
## ddhazard(formula = Surv(time, status == 9) ~ ddFixed_intercept() +
##     ddFixed(age) + ddFixed(sex) + ddFixed(diabetes) + chf + vf,
##     data = TRACE, model = "exponential", by = 0.25, max_T = 7,
##     Q_0 = diag(10, 2), Q = diag(0.1^2, 2), control = ddhazard_control(eps = 0.001,
##         n_max = 25))
##
## 'exponential' model fitted with the 'EKF' method in 19 iterations of the EM algorithm.
##
## Smoothed time-varying coefficients are:
##               chf        sd           vf       sd
## 0.00  1.70950193 0.24337394  2.15710647 0.3415211
## 0.75  0.76306761 0.09438822  1.02734147 0.1455356
## 1.50  0.30781226 0.13981459  0.26484347 0.2418010
## 2.25  0.38133351 0.14453683  0.29675565 0.2639489
## 3.00  0.26332240 0.15698279  0.12343800 0.2920732
## 4.00  0.31451983 0.16521643  0.17376623 0.3111066
## 4.75  0.22839484 0.17777955  0.02797998 0.3372086
## 5.50  0.09962127 0.17998873 -0.15218903 0.3572790
## 6.25  0.34226679 0.19162187  0.19571998 0.3963496
## 7.00 -0.01399385 0.26664463 -0.23268303 0.5006394
```

```
##
## The estimated diagonal entries of the covariance matrix in the state equation are:
##       chf        vf
## 0.2147913 0.4115283
##
## The estimated fixed effects are:
## (Intercept)         age         sex    diabetes
## -6.76556920  0.06120493  0.17884482  0.48937436
##
## 1878 individuals used in estimation with 938 observed events.
```

Sampling individuals

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

```
set.seed(7451)
R <- 999 # number of bootstrap samples
boot_out <- ddhazard_boot(dd_fit, R = R)
```

The list has the same structure and class as the list returned by `boot::boot` a few other elements:

```
class(boot_out)
```

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

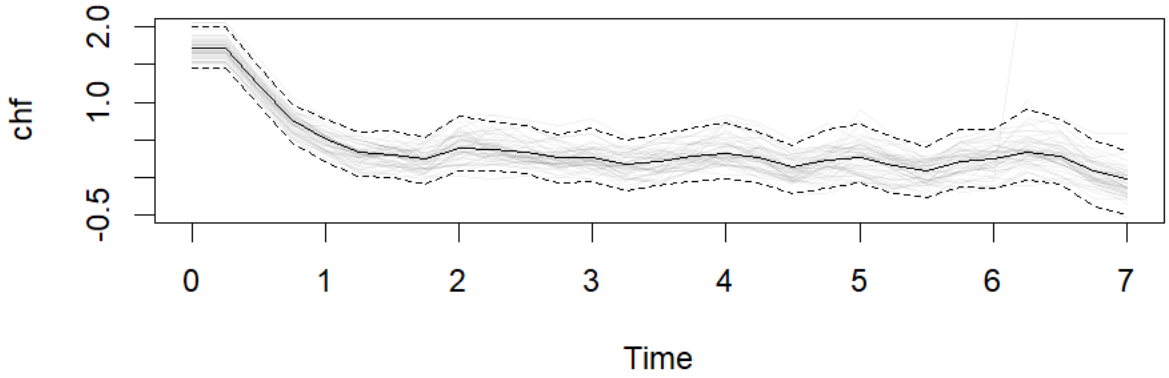
```
str(boot_out)
```

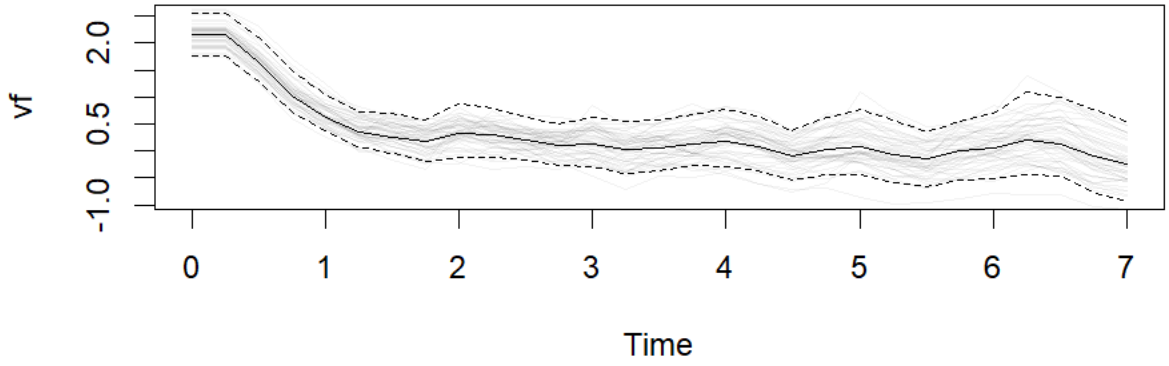
```
## List of 12
##  $ t0       : Named num [1:62] 1.714 1.714 1.231 0.759 0.513 ...
##   ..- attr(*, "names")= chr [1:62] "chf:t0" "chf:t1" "chf:t2" "chf:t3" ...
##  $ t        : num [1:999, 1:62] 1.73 1.85 1.52 1.8 1.77 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:62] "chf:t0" "chf:t1" "chf:t2" "chf:t3" ...
##  $ R        : num 999
##  $ data     : Named num [1:1878] 1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "names")= chr [1:1878] "1" "2" "3" "4" ...
##  $ seed     : int [1:626] 403 624 -1318928793 -836455556 -1543698867 826078378 2061930019 -137608575
##  $ statistic:function (data, ran.gen)
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 104 16 195 3 16 3 1331 1422
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x0000000022e99348>
##  $ sim      : chr "ordinary"
##  $ call     : language boot(data = data, statistic = statistic, R = R, sim = ifelse(do_sample_weights
##  $ stype    : chr "i"
##  $ strata   : Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ weights  : num [1:1878] 0.000532 0.000532 0.000532 0.000532 0.000532 ...
##  $ t_names  : chr [1:62] "chf:t0" "chf:t1" "chf:t2" "chf:t3" ...
##  - attr(*, "class")= chr [1:2] "ddhazard_boot" "boot"
##  - attr(*, "boot_type")= chr "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 5% and 95% 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.

Fixed effects

Recall that the fixed effects are estimated to be:

```
dd_fit$fixed_effects
```

```
## (Intercept)         age         sex    diabetes
## -6.76556920  0.06120493  0.17884482  0.48937436
```

We can get confidence bounds for these with the `boot.ci` function from the `boot` library as shown below:

```
library(boot)

# print confidence intervals for
colnames(boot_out$t)[ncol(boot_out$t)] # this variable
```

```
## [1] "diabetes"
```

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

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_out, type = c("norm", "basic", "perc"),
##     index = ncol(boot_out$t))
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   ( 0.2765,  0.6876 )   ( 0.2706,  0.6821 )   ( 0.2971,  0.7086 )
## Calculations and Intervals on Original Scale
```

Strata

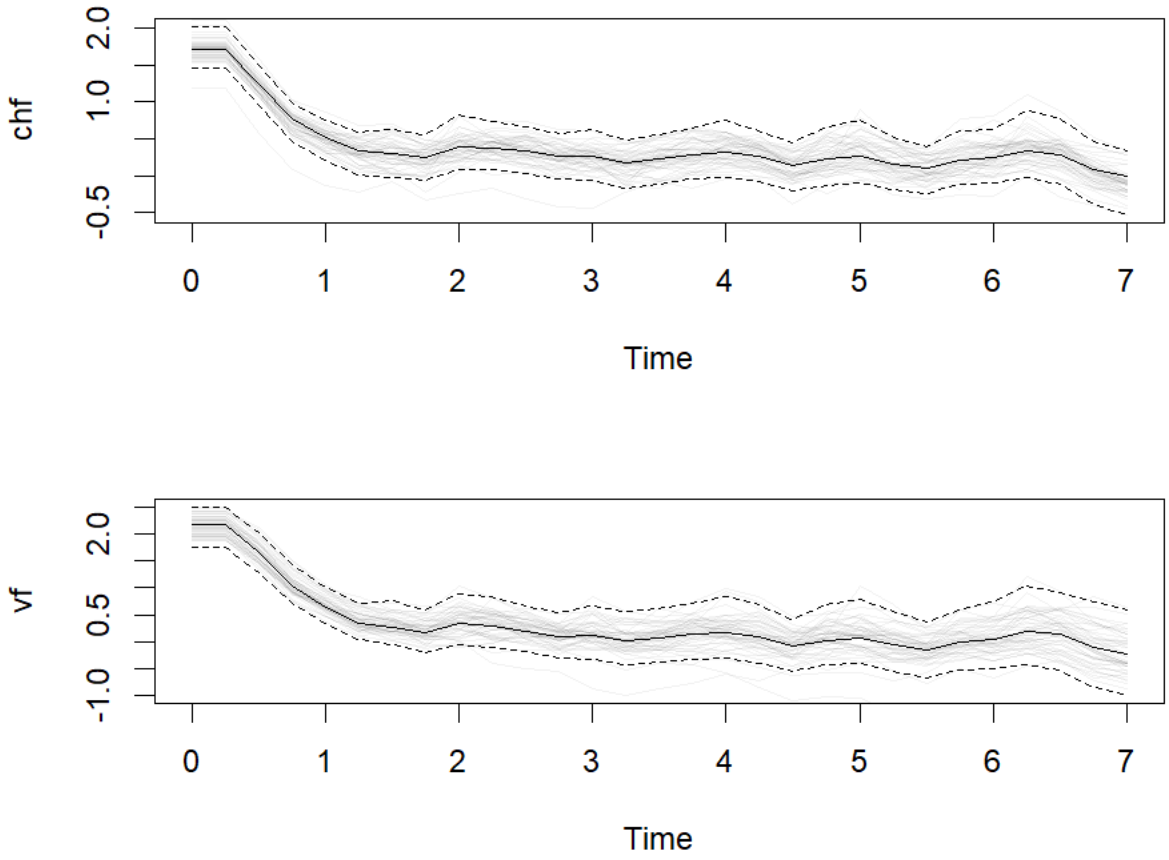
You can provide a strata variable to perform stratified sampling. This is done by setting the `strata` argument in the call to `ddhazard_boot`. Notice that this has to be on an individual level (one indicator variable per individual) not observation level (not one indicator variable per row in the data set). You can use the `unique_id` argument to match the individual entries with the entries in `strata`. Though, this is not needed for this data set as we do not have time-varying covariates. As an example, we stratify by the `chf` value below:

```
# all observations are unique. I.e. all other individuals have one record.
# Otherwise we had to make a strata with an entry for each individual -- not
# each record in the data.frame used in the estimation
sum(duplicated(TRACE$id))
```

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

```
# use strafied bootstrap
set.seed(101)
boot_out_with_strata <- ddhazard_boot(
  dd_fit, R = R, unique_id = TRACE$id, strata = TRACE$chf)

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

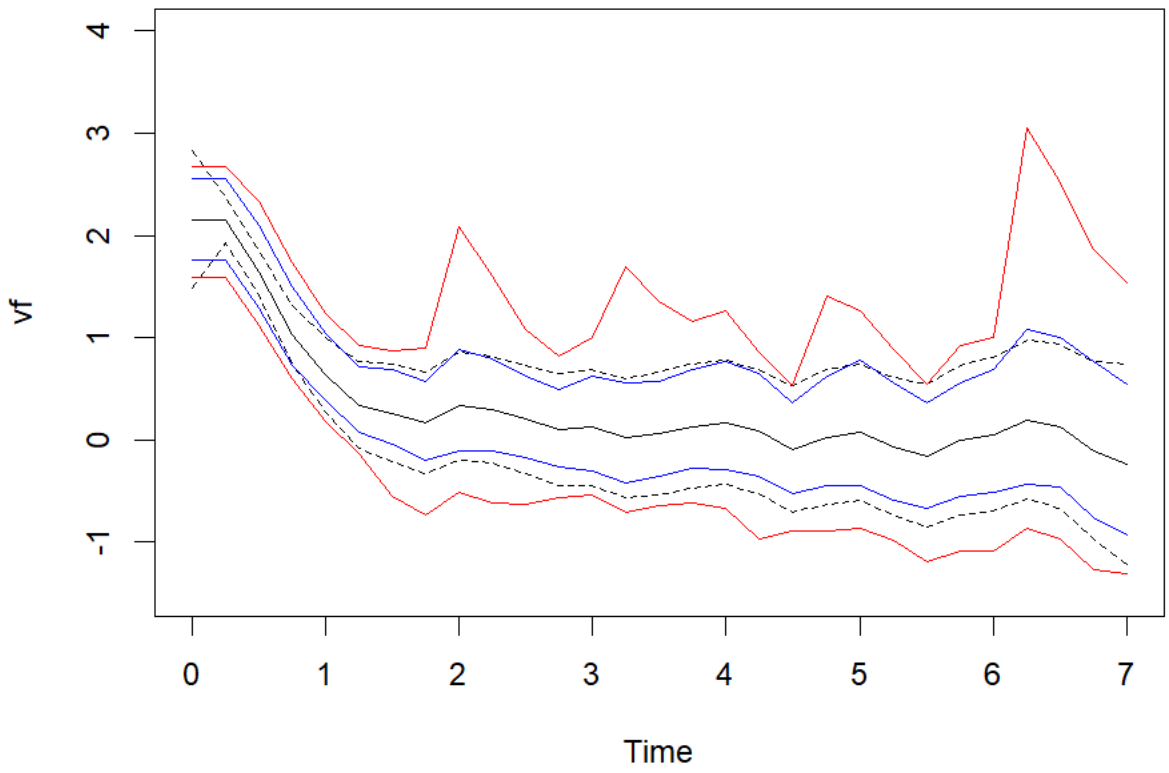
```
## Only plotting 50 of the boot sample estimates
```



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 get simultaneous confidence intervals for the `vf` coefficient as follows:

```
# find the indices that correspondents to the coefficient we want
is_vf <- grep("^vf:", colnames(boot_out$t))

# use the envelope
envelopes <- envelope(boot_out, level = 0.95 ,index = is_vf)

# plot curves
plot(dd_fit, ylim = c(-1.5, 4),
     cov_index = grep("^vf$", colnames(dd_fit$state_vecs)))
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:

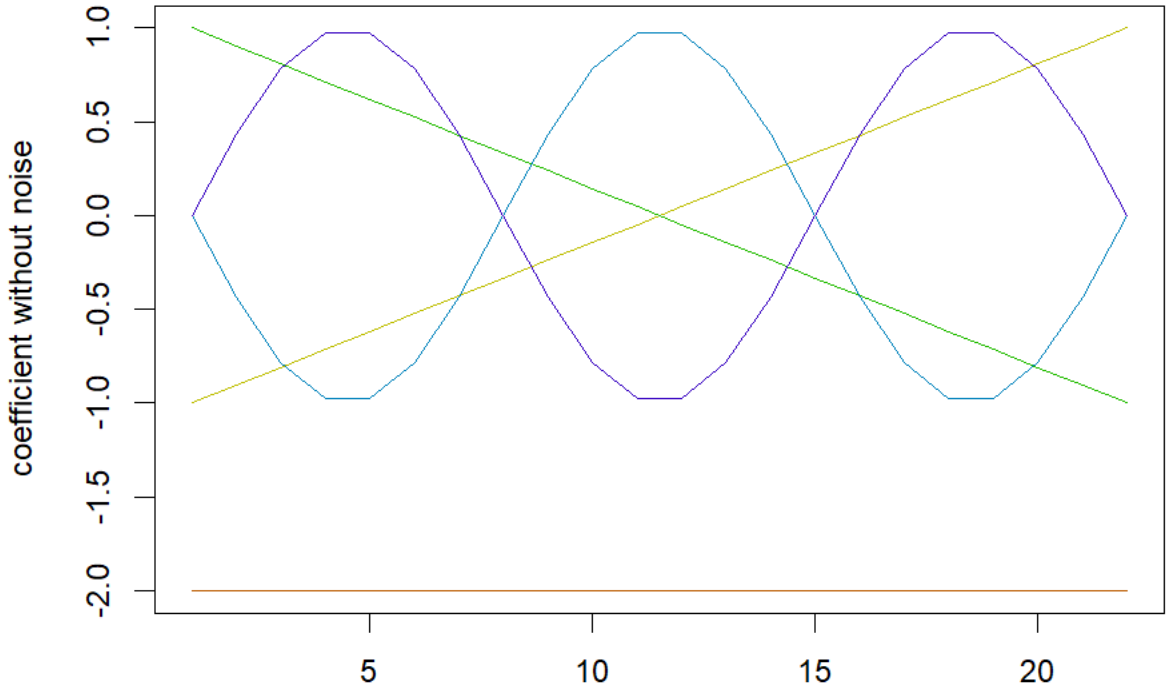
- The coefficients are drifting deterministically with a some normal noise added to them.
- Individuals have time invariant covariates.

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`. I.e. we have random random delayed entry. 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:

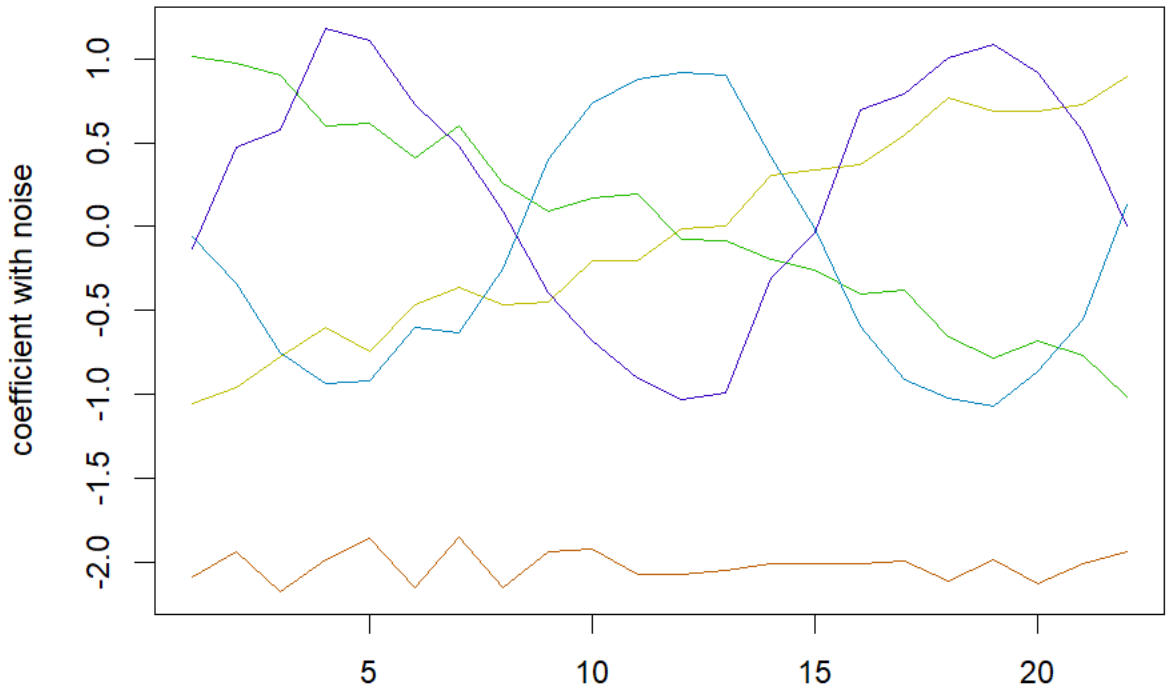
```r
# Simulate
set.seed(122044)
sim_list <- sim_func()

# 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)
```

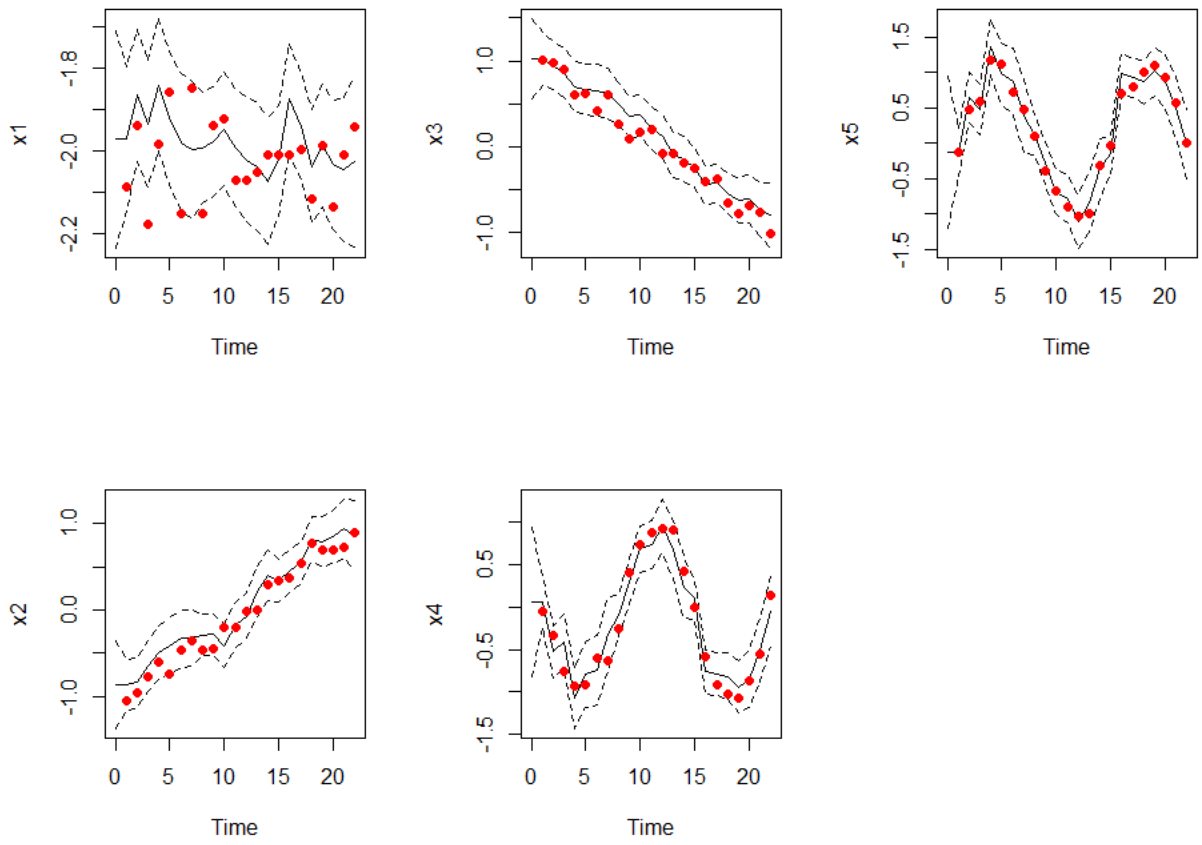
```
##        id tstart tstop x1    x2     x3     x4    x5   eta dies
## 1881 1881      0     3  1  0.48  0.728 -0.834  0.91 -1.92 TRUE
## 1882 1882      0    12  1 -0.66 -0.385  0.084 -0.97 -1.66 TRUE
## 1883 1883      0     3  1  0.25 -0.908  0.246  0.40 -3.34 TRUE
## 1884 1884      0    10  1  0.92 -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:

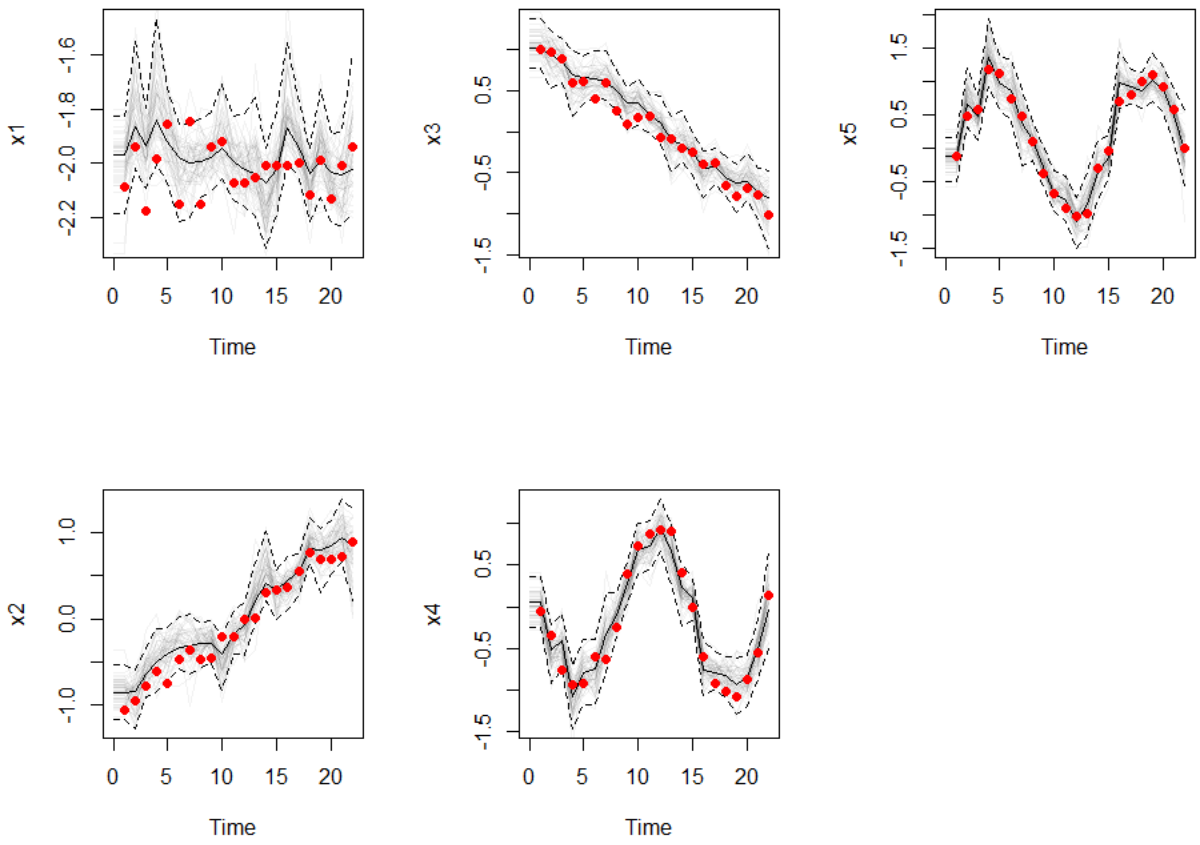
```
# Estimate model
fit_expression <- expression({
  fit <- ddhazard(
    Surv(tstart, tstop, dies) ~ -1 + x1 + x2 + x3 + x4 + x5,
    data = sim_list$sims, id = sim_list$sims$id, max_T = tmax,
    by = 1, Q_0 = diag(1e4, 5), Q = diag(.1, 5),
    a_0 = rep(0, 5), control = ddhazard_control(eps = .001, n_max = 25))
})
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, R = 999)

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

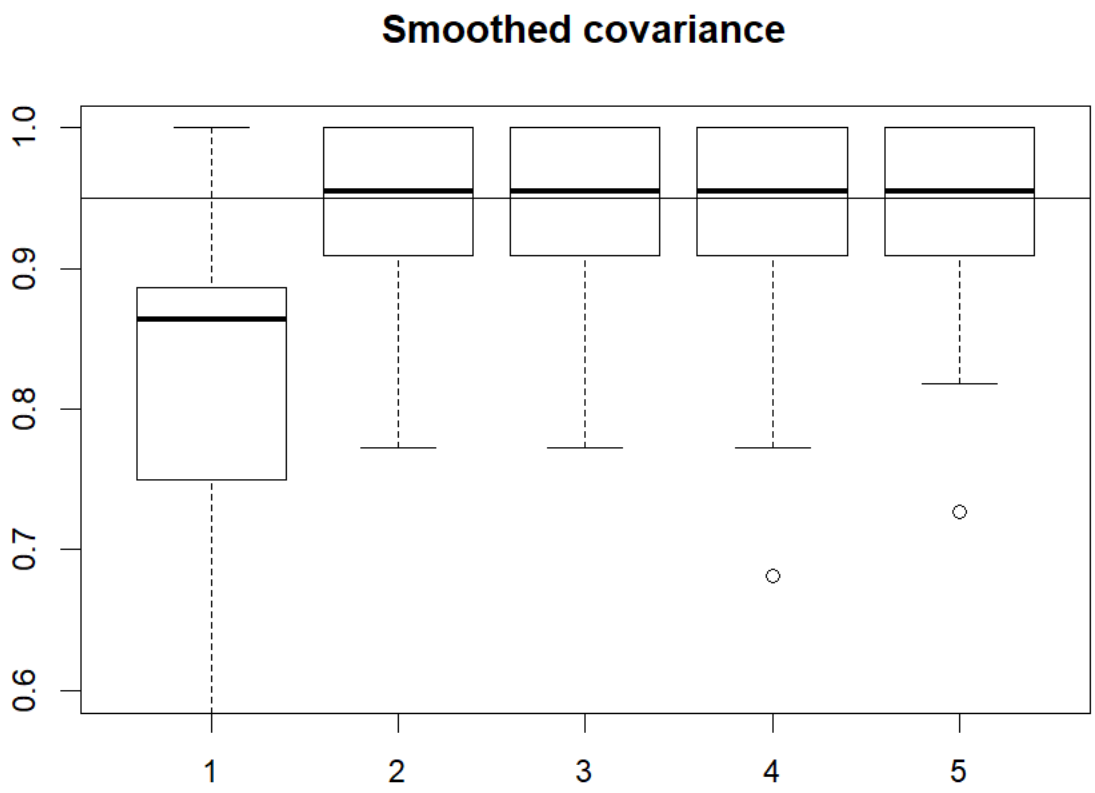
```
## [1] 0.8122727 0.9481818 0.9500000 0.9468182 0.9400000
```

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

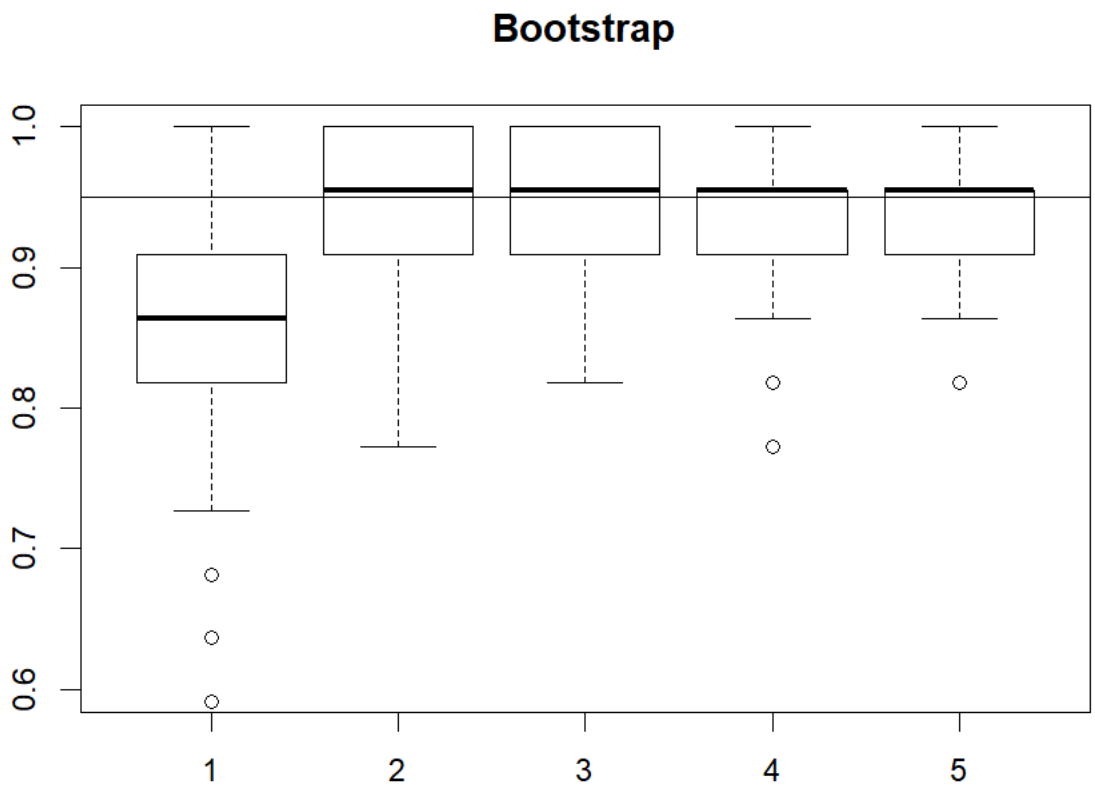
```
## [1] 0.8422727 0.9404545 0.9445455 0.9386364 0.9368182
```

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:

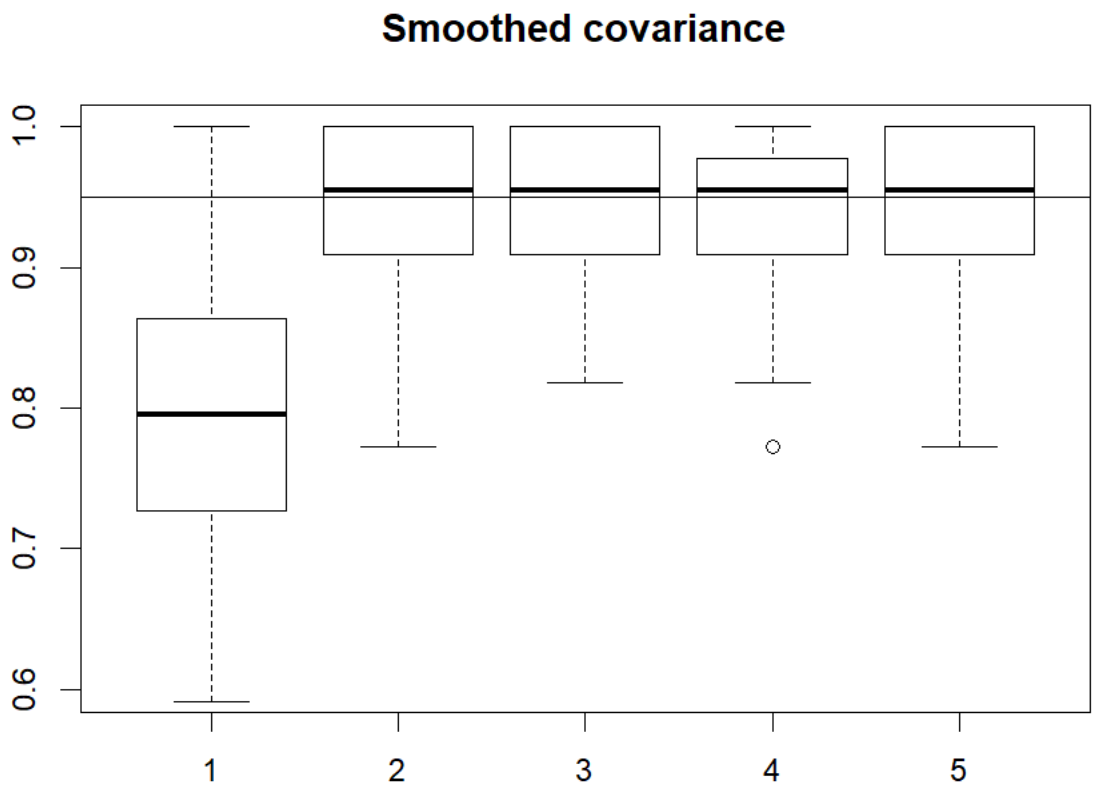
```
colMeans(coverage_smooth, na.rm = T)
```

```
## [1] 0.7895455 0.9513636 0.9468182 0.9400000 0.9418182
```

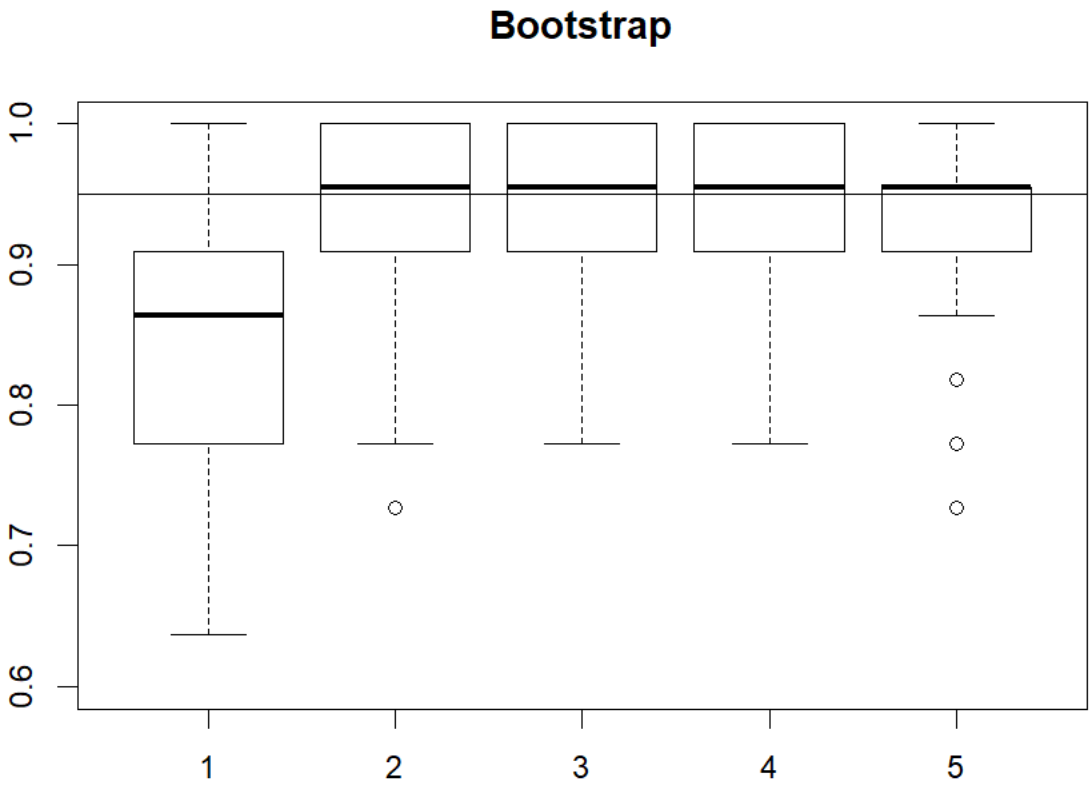
```
colMeans(coverage_boot, na.rm = T)
```

```
## [1] 0.8309091 0.9400000 0.9386364 0.9390909 0.9313636
```

```
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.