Assessing Vaccine Durability Following Placebo Crossover

Jon Fintzi and Dean Follmann Biostatistics Research Branch, NIAID, NIH

12/28/2020

Overview

Randomized vaccine trials are used to assess vaccine efficacy and to characterize the durability of vaccine induced protection. There is a broad consensus that placebo volunteers in COVID-19 vaccine trials should be offered a vaccine once efficacy has been established. This will likely lead to most placebo volunteers crossing over to the vaccine arm, thus complicating the assessment of long term durability.

In this vignette, we provide code for simulating COVID-19 vaccine trials and assessing vaccine efficacy (VE) and durability of vaccine induced protection within a Cox proportional hazards framework. The code supplied in this vignette reproduces the example trials in Fintzi and Follmann (2021), but is flexible and can be modified to accommodate staggered entry, frailties, waning or constant VE, and various study designs including placebo-crossover, continuous crossover, or standard parallel arm trials.

Setup

We will simulate a COVID-19 vaccine trial where the true VE waned linearly on the log–hazard scale from 85% to 35% over 1.5 years. The trial enrolls 30,000 participants with linear accrual over three months in a 1:1 randomization to vaccine or placebo. The baseline attack rate is piece–wise constant, with change–points every three months, and is calibrated to yield 50, 75, 50, and 25 cases on the placebo arm in each period in the first year, and half the expected number of cases per period on the placebo arm in year two. In this example, interim analyses are planned at 150 cases, which ultimately result in crossover at the end of year one following evaluation and vetting of the efficacy by a regulatory agency. Placebo crossover occurs over a four week period. Each volunteer is followed for a total of two years.

Let $t \geq 0$ index time since study initiation, and let $\boldsymbol{\tau} = \left\{ \boldsymbol{\tau}_i = \left(\tau_i^{(e)}, \tau_i^{(v)} \right); \tau_i^{(v)} \geq \tau_i^{(e)} > 0,\ i = 1, \ldots, N \right\}$ denote the times of study entry and vaccination for the study participants. Define $Z_i(t)$ to be an indicator for whether $t > \tau_i^{(v)}$. We will suppose that vaccine efficacy decays linearly on the log-hazard scale. The hazard for participant i is,

$$h_i(t) = \begin{cases} 0 & ,\ t \leq \tau_i^{(e)}, \\ h_0(t) \exp\left[Z_i(t) \left(\theta_1 + \theta_2 \left(t - \tau_i^{(v)} \right) \right) \right] & ,\ t > \tau_i^{(e)}. \end{cases} \tag{1}$$

Here, $VE(t) = 1 - \exp\left[\theta_1 + \theta_2 \left(t - \tau^{(v)} \right) \right]$. The baseline hazards, $h_0(t) = \gamma_\ell,\ t \in (t_{\ell-1}, t_\ell]$, are piecewise constant over three month periods, $(t_0, t_1],\ (t_1, t_2], \ldots,\ (t_7, t_8]$, and follow a seasonal pattern calibrated so we have an average of 50 events on the placebo arm in the first three months, 75 in months 4-6, 50 in months 7-9, and 25 in months 10-12 of the first study year. The baseline hazards in the subsequent study year are halved.

We integrate the hazard and obtain the survival function. Event times can then be simulated via the inverse CDF method. This is facilitated via wrappers implemented in the `simsurv` package (Brilleman, 2019).

Functions to Simulate COVID-19 Trials

We implement the functions for simulating trials below. Note that these functions will do more than we need in this vignette, but are general purpose wrappers that cover all of the simulations and examples in the manuscript. We first implement a function that returns the cumulative hazard in a time-interval over which the hazards are homogeneous.

```
# cumulative hazard in a time-homogeneous interval
c_haz_int =
  function(lambda_t,        # baseline hazard
           t_l,             # left endpoint
           t_r,             # right endpoint
           trt,             # treatment indicator (0 if not yet vaccinated)
           VE_max = NULL,   # VE(0), theta_1
           VE_decay = NULL, # slope parameter, theta_2
           t_trt = NULL,    # vaccination time
           frailty = 1) {

  # cumulative hazard over any interval for unvaccinated, always
  if(trt == 0) {
    c_haz = frailty * lambda_t * (t_r - t_l)
  } else {
    if(VE_decay == 0) {
      c_haz = frailty * lambda_t * exp(VE_max) * (t_r - t_l)
    } else {
      c_haz =
        frailty * lambda_t * exp(VE_max) / VE_decay *
        (exp(VE_decay * (t_r - t_trt)) - exp(VE_decay * (t_l - t_trt)))
    }
  }
  return(c_haz)
}
```

We now define a wrapper that calculates the cumulative hazard for a period, (0,t], over which the hazard is time-varying by aggregating the contributions from piecewise-homogeneous intervals.

```
# cumulative hazard function
# t: vector of times
# x: matrix with covariates
# betas: vector with parameters
cumul_haz = function(t, x, betas, ...) {

  # initialize the hazard vector
  c_haz_t = rep(0.0, length(t))

  # get time when hazard starts accumulating
  tstart = max(betas[["t0"]], x[["entry"]])

  # baseline hazard intervals
  haz_ints = seq(0, 2.25, by=0.25)
  times = sort(c(unique(c(betas[["t0"]], x[["entry"]], x[["crossover"]],
                          haz_ints, t))))
  endpoints_l = head(times, length(times)-1)
```

```
  endpoints_r = tail(times, length(times)-1)

  # which intervals are contributing
  contribs = endpoints_l >= tstart & endpoints_r <= min(t, 2.25)

  # treatment indicators at left endpoints
  trt_inds = ifelse((x[["trt"]] == 1 & endpoints_l >= x[["entry"]]) |
                      (x[["trt"]] == 0 & endpoints_l >= x[["crossover"]]), 1, 0)

  # now compute the cumulative hazards
  for(s in seq_along(t)) {

    # initialize vector of cumulative hazards
    c_haz_ints_s = rep(0.0, length(endpoints_l))

    # compute the hazards for each interval
    for(i in seq_along(endpoints_l)) {
      if(contribs[i]) {

        # grab the baseline hazard
        lambda_t =
          betas[[paste0("lambda_",
                        findInterval(endpoints_l[i],
                                     haz_ints, all.inside = T))]]

        # set time of treatment administration
        t_trt = ifelse(x[["trt"]] == 1, x[["entry"]], x[["crossover"]])

        # cumulative hazard contribution
        c_haz_ints_s[i] =
          c_haz_int(lambda_t = lambda_t,
                    t_l = endpoints_l[i],
                    t_r = endpoints_r[i],
                    trt = trt_inds[i],
                    VE_max = betas[["VE_max"]],
                    VE_decay = betas[["VE_decay"]],
                    t_trt = t_trt,
                    frailty = x[["frailty"]])
      }
    }

    c_haz_t[s] = sum(c_haz_ints_s)
  }

  return(c_haz_t)
}
```

Finally, a wrapper for simulating a COVID-19 vaccine trial.

```
sim_one <- function(n,                  # number of participants
                    pars,               # vector of simulation parameters
                    cumul_haz,          # cumulative hazard function
                    n_cross,            # number of events to start crossover
```

```
                    t_cross,           # time of crossover
                    enrolldur = 12/52, # enrollment period
                    crossdur = 4/52,   # crossover period
                    pvacc = 0.5,       # probability of vaccination
                    frailty_pars = NULL) { # gamma frailty dist params

  if(!is.null(t_cross)) {

  # create a data frame with subject IDs and treatment indicators
  covars <- data.frame(id = seq_len(n),
                       trt = rbinom(n, 1, pvacc),
                       entry = runif(n, 0, enrolldur),
                       crossover = 0,
                       frailty = 1)

  if(!is.null(frailty_pars)) {
    covars$frailty = rgamma(n, shape = frailty_pars[1], rate = frailty_pars[2])
  }

  # simulate crossover times for placebo group
  covars$crossover[covars$trt == 0] =
    t_cross + runif(sum(covars$trt == 0), 1e-5, 1e-5 + crossdur)

  # simulate event times
  dat <- simsurv::simsurv(betas = c(pars, t0 = 0),
                          x = covars,
                          cumhazard = cumul_haz,
                          maxt = 2.25,
                          rootfun = qlogis,
                          ids = covars$id,
                          idvar = "id")

  dat <- merge(covars, dat) %>% arrange(eventtime)

} else if(!is.null(n_cross)) {

  # create a data frame with subject IDs and treatment indicators
  covars_1 <- data.frame(id = seq_len(n),
                         trt = rbinom(n, 1, pvacc),
                         entry = runif(n, 0, enrolldur),
                         crossover = Inf,
                         frailty = 1)

  if(!is.null(frailty_pars)) {
    covars$frailty = rgamma(n, shape = frailty_pars[1], rate = frailty_pars[2])
  }

  # simulate event times
  dat_1 <-
    simsurv::simsurv(betas = c(pars, t0 = 0),
                     x = covars_1,
                     cumhazard = cumul_haz,
                     maxt = 2.25,
```

```
                      rootfun = qlogis,
                      ids = covars_1$id,
                      idvar = "id")

    # sort by event times
    dat_1 <- dat_1 %>% arrange(eventtime)
    n_events = cumsum(dat_1$status)

    # grab the crossover time
    t_cross = dat_1$eventtime[n_events == n_cross][1]

    # cache the pre-crossover data
    dat_1 <- dat_1 %>% filter(dat_1$eventtime <= t_cross)
    dat_sim_1 = merge(covars_1, dat_1) %>%  arrange(eventtime)

    # subset to get covariates for next part of the simulation
    covars_2 = covars_1[!covars_1$id %in% dat_sim_1$id, ]

    # generate crossover times
    covars_2$crossover[covars_2$trt == 0] =
      t_cross + runif(sum(covars_2$trt == 0), 1e-5, 1e-5 + crossdur)

    # keep simulating
    dat_2 <-
      simsurv::simsurv(betas = c(pars, t0 = t_cross),
                      x = covars_2,
                      cumhazard = cumul_haz,
                      maxt = 2.25,
                      rootfun = qlogis,
                      ids = covars_2$id,
                      idvar = "id")

    # sort by event time
    dat_2 <- dat_2 %>% arrange(id)

    # merge covars_2 and dat_2
    dat_sim_2 = merge(covars_2, dat_2) %>% arrange(eventtime)

    # combine simulated datasets
    dat = rbind(dat_sim_1, dat_sim_2)
    dat = dat %>% arrange(eventtime)
  }

  dat$id = seq_along(dat$id)

  # censor people at two years post-study entry
  cens_inds = dat$eventtime > (dat$entry + 2)
  dat$eventtime[cens_inds] = dat$entry[cens_inds] + 2
  dat$status[cens_inds] = 0

  return(dat)
}
```

We will also find it useful to have functions for censoring a trial at crossover and reshaping a simulated

dataset to get it into start-stop format.

```
# to censor people at crossover
cens_cross <- function(dat, n_cross, t_cross) {

    if(!is.null(n_cross)) {

        # get the crossover time
        t_cross = dat[n_cross, "eventtime"]

        # censor events after crossover
        dat$status[(n_cross + 1):nrow(dat)] = 0
        dat$eventtime[(n_cross + 1):nrow(dat)] = t_cross

        # get rid of the crossover indicator
        dat$crossover = NULL

    } else {

        # get the crossover index
        cross_ind = which(dat[,"eventtime"] > t_cross)[1]

        # censor events after crossover
        dat$status[cross_ind:nrow(dat)] = 0
        dat$eventtime[cross_ind:nrow(dat)] = t_cross

        # get rid of the crossover indicator
        dat$crossover = NULL
    }

    # drop records where people were censored prior to entry
    dat <- subset(dat, eventtime > entry)

    # censor people at two years post-study entry
    cens_inds = dat$eventtime > (dat$entry + 2)
    dat$eventtime[cens_inds] = dat$entry[cens_inds] + 2
    dat$status[cens_inds] = 0

    # rename columns =
    names(dat)[1:5] = c("id", "vacc", "tstart", "tstop", "status")

    return(dat)
}

reshape_dat <- function(dat, align_entry = FALSE) {

    # align times so that entry times correspond to time zero for each participant
    if(align_entry) {
        dat <-
            dat %>%
            group_by(id) %>%
            mutate(crossover = ifelse(trt == 1, 0, crossover - entry),
                   eventtime = eventtime - entry) %>%
            mutate(entry = 0) %>%
```

```
        ungroup() %>%
        as.data.frame()
}

# two rows for each placebo, one for each in the treatment arm
dat_new <-
    data.frame(id = unlist(mapply(rep, x = dat$id,
                                  each = ifelse(dat$trt == 1, 1, 2))),
               tstart = unlist(lapply(seq_len(nrow(dat)),
                                      function(x) {
                                          if(dat$trt[x] == 1) {
                                              dat$entry[x]
                                          } else {
                                              c(dat$entry[x], dat$crossover[x])
                                          }})),
               tstop = NA,
               status = NA,
               assg = 0,
               vacc = 0,
               tvacc = 0)

# fill out the interval endpoints and status
tmax = max(dat$eventtime)
for(s in seq_along(unique(dat_new$id))) {

    inds <- which(dat_new$id == dat$id[s])

    if(dat$trt[s] == 1) {

        dat_new$assg[inds] = 1
        dat_new$vacc[inds] = 1
        dat_new$tvacc[inds] = dat$entry[s]
        dat_new$tstop[inds] = dat$eventtime[s]
        dat_new$status[inds] = dat$status[s]

    } else {

        dat_new$assg[inds] = c(0,0)
        dat_new$vacc[inds] = c(0,1)
        dat_new$tvacc[inds] = c(dat$crossover[s], dat$crossover[s])

        if(dat$eventtime[s] < dat$crossover[s]) {

            dat_new$tstop[inds]  = c(dat$eventtime[s], NA)
            dat_new$status[inds] = c(dat$status[s], NA)

        } else {

            dat_new$tstop[inds] = c(dat$crossover[s], dat$eventtime[s])
            dat_new$status[inds] = c(0, dat$status[s])
        }
    }
}
```

```
    dat_new =
        dat_new[complete.cases(dat_new),] %>%
        arrange(id, tstop)

    return(dat_new)
}
```

Simulating a COVID-19 Vaccine Trial

We start by setting up the parameters for a COVID-19 vaccine trial. The time of crossover is 1 year and we'll have a seasonally varying attack rate. VE is calibrated to wane linearly on the log-hazard scale from 85% to 35% over 1.5 years.

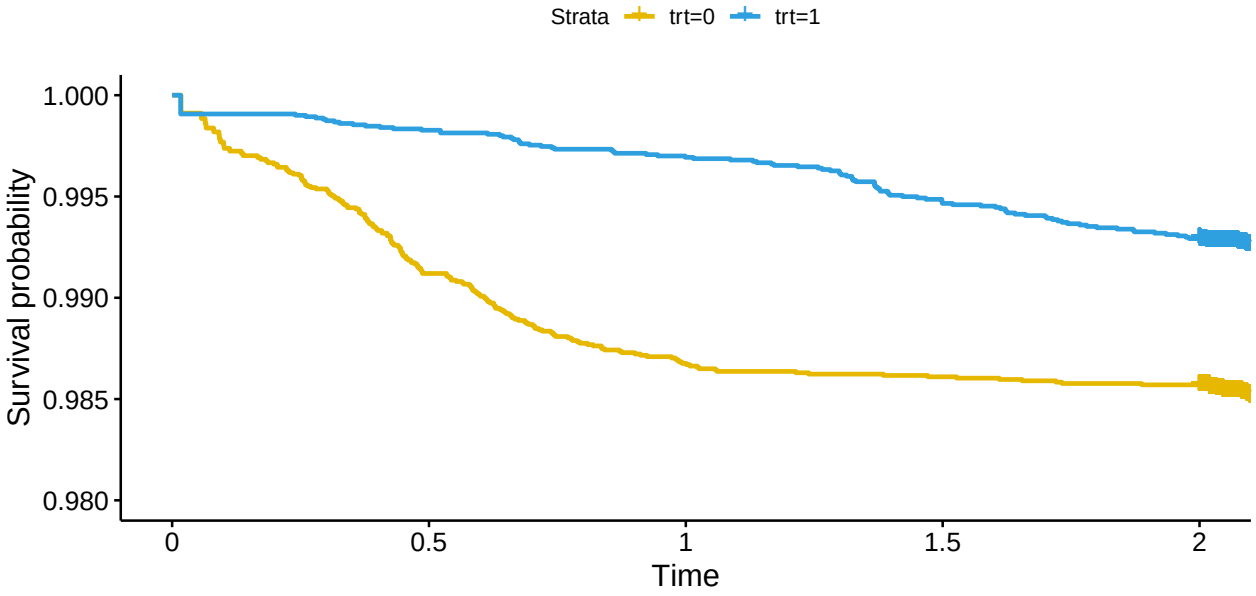
```
# set up parameters
t_cross = 1    # time of crossover
n_cross = NULL # we're not crossing at some number of events, this is NULL

# To get the baseline hazards
# h = function(r) (pexp(0.5, r) * 1.5e4 - 50)^2
# optimize(h, interval = c(0, 1))

sim_pars_waning =
    c(VE_max = log(1 - 0.85), # theta_1
      VE_decay = 0.977558,    # theta_2
      lambda_1 = 0.0134,      # 50 / 3 months
      lambda_2 = 0.02,        # 75 / 3 months
      lambda_3 = 0.0134,      # 50 / 3 months
      lambda_4 = 0.0067,      # 25  / 3 months
      lambda_5 = 0.0067,      # 25 / 3 months
      lambda_6 = 0.01,        # 37.5 / 3 months
      lambda_7 = 0.0067,      # 25 / 3 months
      lambda_8 = 0.0033,      # 12.5 / 3 mo
      lambda_9 = 0.0067)      # 25 / 3 mo
```

Let's go ahead and simulate a trial. We'll set the parameters and RNG seed, then simulate a trial.

```
# set.seed
set.seed(1834)

# simulate trial
sim_waning <-
    sim_one(n = 3e4,
            pars = sim_pars_waning,
            n_cross = n_cross,
            t_cross = t_cross,
            cumul_haz = cumul_haz)

# reshape the simulated dataset
simr_waning <- reshape_dat(sim_waning, align_entry = FALSE)

# save the datasets
```

```
saveRDS(sim_waning, file = "sim_waning.Rds")
saveRDS(simr_waning, file = "simr_waning.Rds")
```

Here's what our simulated data looks like. Note that it will be convenient to have vaccination coded as a time-varying covariate, which for the placebo group will change values after the placebo participants are crossed over.



id	tstart	tstop	status	assg	vacc	tvacc
1	0.0137061	0.0165915	1	1	1	0.0137061
2	0.0030863	0.0174004	1	0	0	1.0491566
3	0.0149288	0.0568718	1	0	0	1.0281499
4	0.0305169	0.0643275	1	0	0	1.0725880
5	0.0431431	0.0659803	1	0	0	1.0205335
6	0.0574278	0.0809023	1	0	0	1.0264042
7	0.0573545	0.0912254	1	0	0	1.0395848
8	0.0112373	0.0917694	1	0	0	1.0379315
9	0.0453881	0.0938827	1	0	0	1.0557889
10	0.0988826	0.1003066	1	0	0	1.0564997

Inference

We'll fit two models. The first is the true data generating model, where VE decays linearly on the log hazard scale. Second, we'll model the VE decay profile using a P-spline. In practice, it is important in both of these models to recognize that we need to evaluate the value of the time-varying VE profile for each participant at each event time, not just at the failure time of that particular participant. The `tt()` function in the `survival` package generates an expanded dataset that evaluates the value of a time-varying functional, in this case the VE profile, at each event time. An in-depth explanation on use of the `tt()` function is provided in the survival vignette.

We'll start by fitting the log-linear decay model. Note that the `vacc` variable is a time-varying covariate for vaccination status, and that the decay profile is a function of the time since vaccination, `tvacc`. The `tt()` function then maps time since vaccination to either zero, or the time elapsed between when a subject was vaccinated and the current event time. The model is fit as follows:

```
# Cox PH model with log-linear decay
# note that time since vaccination is a time-varying parameter.
# also, vacc is a time-varying covariate.
fit_lin =
  coxph(formula =
                  Surv(time = tstart,
                       time2 = tstop,
                       event = status) ~ vacc + tt(tvacc),
              tt = function(tvacc, t, ...) {
                  pmax(0, t - tvacc)
              },
              data = simr_waning)

fit_lin
```

```
## Call:
## coxph(formula = Surv(time = tstart, time2 = tstop, event = status) ~
##     vacc + tt(tvacc), data = simr_waning, tt = function(tvacc,
##     t, ...) {
##     pmax(0, t - tvacc)
## })
##
##               coef exp(coef) se(coef)      z        p
## vacc      -2.1890    0.1120   0.2220 -9.862  < 2e-16
## tt(tvacc)  1.3291    3.7777   0.2577  5.158 2.49e-07
##
## Likelihood ratio test=132.5  on 2 df, p=< 2.2e-16
## n= 44890, number of events= 292
```

And now we fit the P-spline decay model.

```
# p-spline decay model. again, time since vaccination is a time-varying
# parameter and vacc is a time-varying covariate.
fit_ps =
  coxph(formula =
                  Surv(time = tstart,
                       time2 = tstop,
                       event = status) ~ vacc + tt(tvacc),
              tt = function(tvacc, t, ...) {
                  pspline(pmax(0, t - tvacc), df = 3, nterm = 8)
              },
              data = simr_waning)

fit_ps
```

```
## Call:
## coxph(formula = Surv(time = tstart, time2 = tstop, event = status) ~
##     vacc + tt(tvacc), data = simr_waning, tt = function(tvacc,
##     t, ...) {
##     pspline(pmax(0, t - tvacc), df = 3, nterm = 8)
## })
##
```

```
##                     coef se(coef)    se2  Chisq   DF       p
## vacc              -2.256    0.456  0.386 24.452 1.00 7.6e-07
## tt(tvacc), linear  1.285    0.262  0.262 23.948 1.00 9.9e-07
## tt(tvacc), nonlin                         5.312 2.05   0.074
##
## Iterations: 4 outer, 11 Newton-Raphson
##      Theta= 0.809
## Degrees of freedom for terms= 0.7 3.1
## Likelihood ratio test=139  on 3.77 df, p=<2e-16
## n= 44890, number of events= 292
```

The `survival` package nicely partitions the P-spline into a linear and non-linear trend. The non-linearity is not statistically significant. We also can test for time-varying VE in each model using a likelihood ratio test. We refit the models without the decay terms, and then calculate our LRT test statistics which are compared to chi-square distributions with degrees of freedom equal to the change in the number of parameters (or effective number of parameters, in the case of the P-spline) between the unrestricted and restricted models. Both tests emphatically reject the null hypothesis of constant VE.

```
# model without waning VE
lin_nowaning = update(fit_lin, .~. - tt(tvacc))
ps_nowaning = update(fit_ps, .~. - tt(tvacc))

# likelihood ratio test for waning VE
lin_lrt_pval =
    1 - pchisq(2 * (fit_lin$loglik - lin_nowaning$loglik)[2], df = 1)

ps_lrt_pval =
  1 - pchisq(2 * (fit_lin$loglik - lin_nowaning$loglik)[2], df = fit_ps$df[2] - 1)

# print p-values
lin_lrt_pval
```

```
## [1] 2.718607e-08
```

```
ps_lrt_pval
```

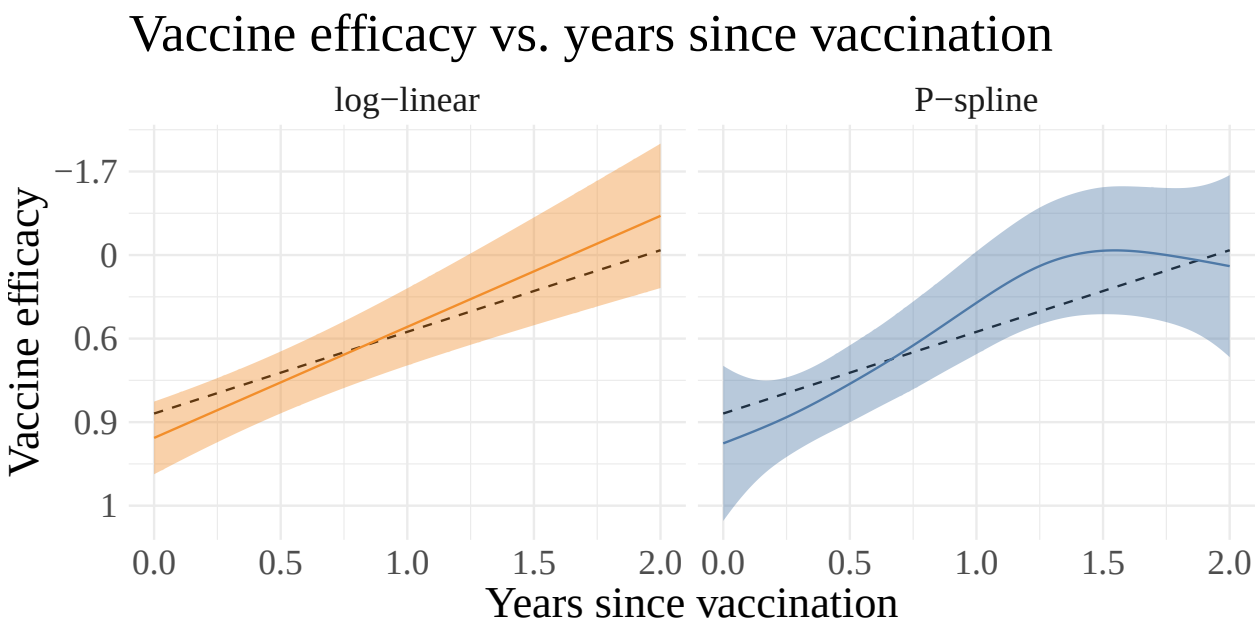
```
## [1] 2.138699e-07
```

We next extract the model estimates and covariance matrices and plot VE(t). The parameterization of P-splines in the survival package is a bit odd so we have to recenter the spline term.

```
# estimates of VE(t)
VE_ests =
  expand.grid(time = seq(0,2,by=0.01),
              Model = c("log-linear", "P-spline"))

VE_ests$truth = log(0.15) + 0.977558 * VE_ests$time

times_lin = cbind(1, seq(0,2,by=0.01))
times_ps = cbind(1, pspline(seq(0,2,by=0.01), df = 3, nterm = 8))
centvec = rep(0, ncol(times_ps)); centvec[2] = 1 # for centering the spline
centvec = rep(1, ncol(times_ps)) %*% diag(centvec)
```

```
VE_ests$est =
  c(times_lin %*% coef(fit_lin),
    apply(times_ps, 1, function(x) (x - centvec) %*% coef(fit_ps)))

VE_ests$lower =
  VE_ests$est - 1.96 * c(
    sqrt(apply(times_lin, 1, function(x) t(x) %*% vcov(fit_lin) %*% x)),
    sqrt(apply(times_ps, 1, function(x) (x - centvec) %*% vcov(fit_ps) %*% t(x - centvec))))

VE_ests$upper =
  VE_ests$est + 1.96 * c(
    sqrt(apply(times_lin, 1, function(x) t(x) %*% vcov(fit_lin) %*% x)),
    sqrt(apply(times_ps, 1, function(x) (x - centvec) %*% vcov(fit_ps) %*% t(x - centvec))))
```

```
require(ggthemes)
VE_ests %>%
  ggplot(aes(x = time)) +
  geom_line(aes(y = truth, linetype = "Truth"), linetype = "dashed") +
  geom_ribbon(aes(ymin = lower, ymax = upper, fill = Model), alpha = 0.4) +
  geom_line(aes(y = est, colour = Model)) +
  facet_grid(. ~ Model, scales = "free") +
  theme_minimal() +
  scale_color_tableau(palette = "Tableau 10", direction = -1) +
  scale_fill_tableau(palette = "Tableau 10", direction = -1) +
  scale_y_continuous(breaks = seq(-3, 2.5, by = 1),
                     labels = round(1 - exp(seq(-3,2.5,by=1)), digits = 1)) +
  labs(x = "Years since vaccination", y = "Vaccine efficacy",
       title = "Vaccine efficacy vs. years since vaccination") +
  theme(legend.position = "none",
        text = element_text(family = "serif", size = 20))
```



References

- Brilleman S (2019). simsurv: Simulate Survival Data. R package version 0.2.3. https://CRAN.R-project.org/package=simsurv
- Therneau T (2020). *A Package for Survival Analysis in R*. R package version 3.2-7, https://CRAN.R-project.org/package=survival.