Solutions: Chapter 8

Exercise 1

See Chapter 2, below Eq. (2.32)

Exercise 2

Function that compute event time and type for one competing risks experiment:

Input

`data`: data in the mvna form (id, entry, exit, from, to)
`sum.ch`: A function giving the all-cause cumulative hazard. Used for the inversion procedure
`prob`: A function of t giving the probability of observing an event of type 1, say
`from`: In the illness-death model, what is the initial state of the competing risks experiment

Output

A data set of the same type.

```
> block.cp <- function(data, sum.ch, prob, from, max.int = 500) {
+
+     n <- nrow(data)
+     inc <- 1; n.failure <- 0
+     stime <- numeric(n)
+
+     while (inc <= n) {
+         u <- runif(1)
+         if (sum.ch(0, log(1 - u)) * sum.ch(max.int, log(1 - u)) < 0) {
+             res <- uniroot(sum.ch, c(0, max.int), tol = 0.0001, y = log(1 - u))
+             ## if (res$root == 0) browser()
+             stime[inc] <- res$root
+             inc <- inc + 1
+         }
+         else {
+             n.failure <- n.failure + 1
```

```
+         }
+     }
+     time <- stime + data$exit
+
+     to <- rbinom(n, 1, prob = prob(time))
+     to <- switch(from,
+                  "0" = {
+                      ifelse(to == 0, 2, 1)
+                  },
+                  "1" = {
+                      ifelse(to == 0, 2, 0)
+                  })
+
+     data.frame(id = data$id, entry = data$exit, exit = time,
+                from = data$to, to = to)
+ }
```

Function: all cause cumulative hazard from state 0

```
> ch.from.0 <- function(x, y) {
+     return((1/9) * x^3 + (1 / 5) * x + 3 * log(x + 1) + y)
+ }
```

Function: all cause cumulative hazard from state 1

```
> ch.from.1 <- function(x, y) {
+     return(x + (4/3) * x^(3/2) + y)
+ }
```

Function: probability(t) of getting to state 1, starting from state 0

```
> prob.from.0 <- function(tt) {
+     return(((1/3) * tt^2 + (1/5)) / (3 / (1 + tt) + (1/3) * tt^2 + 1/5))
+ }
```

Function: probability(t) of getting to state 0, starting from state 1

```
> prob.from.1 <- function(tt) {
+     return(1 / (1 + 2 * sqrt(tt)))
+ }
```

Function for simulating a data set

Input

`ll`: a list. Just keep it as it is in the call. That's for the recursion
`n`: sample size
`p.init`: probability of being initially in state 1
`ch.from.0`,`ch.from.1`: functions returning all-cause hazards from state 0 or 1

`prob.from0`, `prob.from1`: functions giving probabilities
`cens`: a list of 2: yes: logical, censoring yes or no,
`param`: parameters for the uniformly distributed censoring times `max.int`: parameter for `uniroot()`

Output

A data set in the `mvna/etm` format.

```
> sim.chap8.exo2 <- function(ll = list(), n, p.init,
+                            ch.from0, ch.from1,
+                            prob.from0, prob.from1,
+                            cens = list(yes = FALSE, param = NULL),
+                            max.int = 500) {
+
+     if (length(ll) == 0) {
+         ## create some empty data for a start
+         from <- to <- rbinom(n, size = 1, p.init)
+         id <- seq_len(n)
+         entry <- exit <- numeric(n)
+         temp <- data.frame(id, from, to, entry, exit)
+         ll[[1]] <- rbind(block.cp(temp[temp$from == 0, ], ch.from0,
+                                   prob.from0, "0"),
+                          block.cp(temp[temp$from == 1, ], ch.from1,
+                                   prob.from1, "1"))
+         sim.chap8.exo2(ll, n, p.init, ch.from0, ch.from1,
+                        prob.from0, prob.from1, cens,
+                        max.int)
+     }
+
+     ## then...
+     tmp <- ll[[length(ll)]]
+
+     if (!all(tmp$to == 2)) {
+         ll[[length(ll) + 1]] <- rbind(block.cp(subset(tmp, to == 0), ch.from0,
+                                                prob.from0, "0"),
+                                       block.cp(subset(tmp, to == 1), ch.from1,
+                                                prob.from1, "1"))
+         sim.chap8.exo2(ll, n, p.init, ch.from0, ch.from1,
+                        prob.from0, prob.from1, cens, max.int)
+     } else {
+         almostThere <- do.call(rbind, ll)
+
+         if (cens$yes) {
+             ## time to deal with independent censoring
```

```
+             cens.times <- runif(n, cens$param[1], cens$param[2])
+             almostThere <- almostThere[order(almostThere$id, almostThere$exit), ]
+             cens.times <- cens.times[almostThere$id]
+             indic1 <- almostThere$exit < cens.times
+             indic2 <- almostThere$entry < cens.times
+             almostThere[indic2 != indic1, "exit"] <-
+                 cens.times[indic2 != indic1]
+             almostThere[indic2 != indic1, "to"] <- "cens"
+             finito <- almostThere[indic2, ]
+         } else {
+             finito <- almostThere
+         }
+
+         return(finito)
+     }
+ }
```

Generate a data set

```
> set.seed(4428)
> dat.chap8.exo2 <- sim.chap8.exo2(list(), 200, 0.4,
+                                  ch.from.0, ch.from.1,
+                                  prob.from.0, prob.from.1,
+                                  cens = list(yes = FALSE))
```

Computation of the cumulative transition hazards using `mvna`
Matrix of logical defining the possible transitions

```
> tra.idm <- matrix(FALSE, nrow = 3, ncol = 3)
> tra.idm[1, 2:3] <- TRUE
> tra.idm[2, c(1, 3)] <- TRUE
```

```
mvna()
```

```
> mvna.chap8 <- mvna(dat.chap8.exo2, c("0", "1", "2"), tra.idm, NULL)
```

Plot of the Nelson-Aalen estimates

```
> xyplot(mvna.chap8, layout = c(2, 2), lwd = 2,
+        strip = strip.custom(bg = "white",
+        factor.levels = c(expression(0 %->% 1),
+        expression(0 %->% 2),
+        expression(1 %->% 0),
+        expression(1 %->% 2)),
+        par.strip.text = list(cex = 1.1,
+        font = 2)))
```

True quantities

```
> xx <- seq(0, 4, 0.001)
```

0 -> 1

```
> trellis.focus("panel", 1, 1, highlight = FALSE)
> llines(xx, (1/9) * xx^3 + (1/5) * xx, col = "red", lwd = 2)
```

0 -> 2

```
> trellis.focus("panel", 2, 1, highlight = FALSE)
> llines(xx, 3 * log(xx + 1), col = "red", lwd = 2)
```

1 -> 0

```
> trellis.focus("panel", 1, 2, highlight = FALSE)
> llines(xx, xx, col = "red", lwd = 2)
```

1 -> 2

```
> trellis.focus("panel", 2, 2, highlight = FALSE)
> llines(xx, (4/3) * (xx)^(3/2), col = "red", lwd = 2)
```