Code challenge: solutions

Nathan Green

nathan.green@imperial.ac.uk

These are some example model solutions to the code challenge, set as part of the R for trial & model-based cost-effectiveness analysis workshop taking place 9th July 2019 at University College London. The questions can be found here.

Load supporting packages.

```r
library(purrr)
library(knitr)
```

C1. A simple decision tree

Define the separate monthly transition probabilities.

```r
delta0 <- 0.001182
deltac <- 0.025
deltaa <- 0.08
betac  <- 0.0027
betaa  <- 0.0083
```

and the utilities of being in each state.

```r
sutils <- c(dead = 0, CA = 0.3, cancer = 0.6, AIDS = 0.5, well = 1)
```

Note that the order of states is `dead`, `cancer & AIDS`, `cancer`, `AIDS`, `well`.

Define decision trees in terms of the structure, values and probabilities. We'll use the `tribble` function just because it allows us to specify a matrix by rows rather than columns.

```r
library(tibble)

tree_probs <- list()

# unique state outcomes each branch
tree_probs$well <-
  tribble(~rowname,  ~dead, ~ndead,  ~recurc, ~ncancer, ~CA,  ~cancer, ~AIDS, ~well,
          "well0",   delta0,1-delta0,NA,      NA,       NA,   NA,      NA,    NA,
          "ndead",   NA,    NA,      betac,   1-betac,  NA,   NA,      NA,    NA,
          "recurc",  NA,    NA,      NA,      NA,       betaa,1-betaa, NA,    NA,
          "ncancer", NA,    NA,      NA,      NA,       NA,   NA,      betaa, 1-betaa) %>%
  column_to_rownames()

tree_probs$cancer <-
  tribble(~rowname, ~dead, ~ndead,  ~diec, ~survc,  ~CA,  ~cancer,
          "cancer0",delta0,1-delta0,NA,    NA,      NA,   NA,
          "ndead",  NA,    NA,      deltac,1-deltac,NA,   NA,
          "survc",  NA,    NA,      NA,    NA,      betaa,1-betaa) %>%
  column_to_rownames()

tree_probs$AIDS <-
  tribble(~rowname,~dead, ~ndead,  ~diea, ~surva,  ~CA,  ~AIDS,
          "AIDS0", delta0,1-delta0,NA,    NA,      NA,   NA,
          "ndead", NA,    NA,      deltaa,1-deltaa,NA,   NA,
          "surva", NA,    NA,       NA,    NA,       betac,1-betac) %>%
  column_to_rownames()

tree_probs$CA <-
  tribble(~rowname,~dead, ~ndead,  ~diec, ~survc,  ~diea, ~CA,
          "CA0",   delta0,1-delta0,NA,    NA,      NA,    NA,
          "ndead", NA,    NA,      deltac,1-deltac,NA,    NA,
          "survc", NA,    NA,      NA,    NA,      deltaa,1-deltaa) %>%
  column_to_rownames()

tree_probs$dead <- 1

tree_probs
```

```
## $well
##              dead    ndead recurc ncancer     CA cancer   AIDS   well
## well0   0.001182 0.998818     NA      NA     NA     NA     NA     NA
## ndead         NA       NA 0.0027  0.9973     NA     NA     NA     NA
## recurc        NA       NA     NA      NA 0.0083 0.9917     NA     NA
## ncancer       NA       NA     NA      NA     NA     NA 0.0083 0.9917
##
## $cancer
##              dead    ndead  diec survc     CA cancer
## cancer0 0.001182 0.998818    NA    NA     NA     NA
## ndead         NA       NA 0.025 0.975     NA     NA
## survc         NA       NA    NA    NA 0.0083 0.9917
##
## $AIDS
##            dead    ndead diea surva     CA   AIDS
## AIDS0 0.001182 0.998818   NA    NA     NA     NA
## ndead       NA       NA 0.08  0.92     NA     NA
## surva       NA       NA   NA    NA 0.0027 0.9973
##
## $CA
##            dead    ndead  diec survc diea   CA
## CA0   0.001182 0.998818    NA    NA   NA   NA
## ndead       NA       NA 0.025 0.975   NA   NA
## survc       NA       NA    NA    NA 0.08 0.92
##
## $dead
## [1] 1
```

```
tree_vals <- list()

tree_vals$well <-
  tribble(~rowname, ~dead, ~ndead, ~cancer, ~ncancer, ~CA, ~CnotA, ~AIDS, ~well,
          "well0", 0,0,0,0,0,0,0,0,
          "ndead", 0,0,0,0,0,0,0,0,
          "cancer", 0,0,0,0,0.3,0.6,0,0,
          "ncancer", 0,0,0,0,0,0,0.5,1) %>%
  column_to_rownames()

tree_vals$cancer <-
  tribble(~rowname, ~dead, ~ndead, ~cancer, ~ncancer, ~CA, ~CnotA, ~AIDS, ~well,
          "well", 0,0,0,0,0,0,0,0,
          "ndead", 0,0,0,0,0,0,0,0,
          "cancer", 0,0,0,0,0.3,0.6,0,0,
          "ncancer", 0,0,0,0,0,0,0.5,1) %>%
  column_to_rownames()

tree_vals$AIDS <-
  tribble(~rowname, ~dead, ~ndead, ~cancer, ~ncancer, ~CA, ~CnotA, ~AIDS, ~well,
          "well", 0,0,0,0,0,0,0,0,
          "ndead", 0,0,0,0,0,0,0,0,
          "cancer", 0,0,0,0,0.3,0.6,0,0,
          "ncancer", 0,0,0,0,0,0,0.5,1) %>%
  column_to_rownames()

tree_vals$CA <-
  tribble(~rowname, ~dead, ~ndead, ~cancer, ~ncancer, ~CA, ~CnotA, ~AIDS, ~well,
          "well", 0,0,0,0,0,0,0,0,
          "ndead", 0,0,0,0,0,0,0,0,
          "cancer", 0,0,0,0,0.3,0.6,0,0,
          "ncancer", 0,0,0,0,0,0,0.5,1) %>%
  column_to_rownames()
```

C2. Extend C1 for multiple cycles

Assuming a binomial tree we can forward simulate for a synthetic cohort. This is a brute force approach and is potentially time-consuming.

```
cohort <- list()
n_cohort <- 1000
death_states <- c("diea", "diec", "dead")

for (i in seq_len(n_cohort)) {

  traj_s <- NULL
  traj_u <- NULL
  state_name <- "well"

  while (!state_name %in% death_states) {

    p <- tree_probs[[state_name]]
    binp <- p[state_name, !is.na(p[state_name, ])] #partial match

    while (nrow(binp) > 0) {

      state_name <-
        if (runif(1) < binp[1]) names(binp)[1] else names(binp)[2]

      binp <- p[state_name == rownames(p), !is.na(p[state_name, ])]
    }

    traj_s <- c(traj_s, state_name)
    traj_u <- c(traj_u, sutils[state_name])
  }
  cohort[[i]] <- traj_u
}
```

An example trajectory

```
cohort[[1]]
```

```
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well   well   well   well   well
##    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0    1.0
##   well   well   well   well   well   well cancer cancer cancer cancer
##    1.0    1.0    1.0    1.0    1.0    1.0    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer cancer cancer cancer cancer cancer
##    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6    0.6
## cancer cancer cancer cancer cancer     CA     CA     CA     CA     CA
##    0.6    0.6    0.6    0.6    0.6    0.3    0.3    0.3    0.3    0.3
##     CA     CA     CA     CA   <NA>
##    0.3    0.3    0.3    0.3     NA
```

The mean summary statistics should be close to the true expected value. However, it appears to be pretty noisy and even for fairly large values (in terms of run time) it can be off by one or two.

```
mean(map(cohort, sum, na.rm = TRUE) %>% unlist())
```

```
## [1] 85.5559
```

C3. Markov-cycle tree

Given the following transition matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ (1-\delta_0)\delta_c + (1-\delta_0)(1-\delta_c)\delta_a & (1-\delta_0)(1-\delta_c(1-\delta_a) & 0 & 0 & 0 \\ \delta_0 + (1-\delta_0)\delta_c & (1-\delta_0)(1-\delta_c)\beta_a & (1-\delta_0)(1-\delta_c)(1-\beta_a) & 0 & 0 \\ \delta_0 + (1-\delta_0)\delta_a & (1-\delta_0)\beta_c(1-\delta_a) & 0 & (1-\delta_0)(1-\beta_c)(1-\delta_a) & 0 \\ \delta_0 & (1-\delta_0)\beta_c\beta_a & (1-\delta_0)\beta_c(1-\beta_a) & (1-\delta_0)(1-\beta_c)\beta_a & (1-\delta_0)(1-\beta_c)(1-\beta_a) \end{pmatrix}$$

Then define the transition matrix object

```r
p <- list()

p$dead <- c(1,0,0,0,0)

p$CA <-
  c(delta0 + (1-delta0)*deltac + (1-delta0)*(1-deltac)*deltaa, (1-delta0)*(1-deltac)*(1-deltaa),0,0,0)

p$cancer <-
  c(delta0 + (1-delta0)*deltac, (1-delta0)*(1-deltac)*betaa, (1-delta0)*(1-deltac)*(1-betaa),0,0)

p$AIDS <-
  c(delta0 + (1-delta0)*deltaa, (1-delta0)*betac*(1-deltaa), 0, (1-delta0)*(1-betac)*(1-deltaa), 0)

p$well <-
  c(delta0, (1-delta0)*betac*betaa, (1-delta0)*betac*(1-betaa), (1-delta0)*(1-betac)*betaa,
    (1-delta0)*(1-betac)*(1-betaa))

trans <- do.call(rbind, p)
```

Combine the tree data all together into a single list using a function.

```r
create_tree <- function(trans, utils) {

  if (!all(rowSums(trans) == 1)) stop("probabilities don't sum to one")
  if (nrow(trans) != ncol(trans)) stop("not square matrix")
  if (nrow(trans) != length(utils)) stop("utils length doesnt match transition matrix dimensions")

  colnames(trans) <- rownames(trans)
  names(utils) <- rownames(trans)

  list(trans = trans,
       utils = utils)
}

my_tree <- create_tree(trans, sutils)
```

Check the input data.

```r
str(my_tree)
```

```
## List of 2
##  $ trans: num [1:5, 1:5] 1 0.10406 0.02615 0.08109 0.00118 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:5] "dead" "CA" "cancer" "AIDS" ...
##   .. ..$ : chr [1:5] "dead" "CA" "cancer" "AIDS" ...
##  $ utils: Named num [1:5] 0 0.3 0.6 0.5 1
##   ..- attr(*, "names")= chr [1:5] "dead" "CA" "cancer" "AIDS" ...
```

```r
kable(my_tree$trans, digits = 3)
```

	dead	CA	cancer	AIDS	well
dead	1.000	0.000	0.000	0.000	0.000
CA	0.104	0.896	0.000	0.000	0.000
cancer	0.026	0.008	0.966	0.000	0.000
AIDS	0.081	0.002	0.000	0.916	0.000
well	0.001	0.000	0.003	0.008	0.988

Now we're ready to do the forward cycle. Basically, using the same approach as above for the separate probabilities, we simulate individuals and then take an average.

```r
cohort <- list()
n_cohort <- 1000
p <- my_tree$trans

for (i in seq_len(n_cohort)) {

  traj_s <- NULL
  traj_u <- NULL
  state_name <- "well"

  while (state_name != "dead") {

    res <- rmultinom(n = 1, size = 1, prob = p[state_name, ])
    state_name <- rownames(res)[res[,1] == 1]
    traj_s <- c(traj_s, state_name)
    traj_u <- c(traj_u, sutils[state_name])
  }

  cohort[[i]] <- traj_u
}
```

Here's an example trajectory.

```r
cohort[[1]]
```

```
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well well well well well well well well well well well
##  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
## well well well well well AIDS AIDS AIDS AIDS dead
##  1.0  1.0  1.0  1.0  1.0  0.5  0.5  0.5  0.5  0.0
```

The expected value is

```
mean(map(cohort, sum, na.rm = TRUE) %>% unlist())
```

```
## [1] 88.3194
```

C4. Regular Markov model

After initialising values for the calculation, for each cycle the probability of being in each of the states is calculated using the transition matrix and the previous cycle state occupancy probabilities. Similarly, the utilities associated with being in each state are calculated for each cycle.

```
run_model <- function(tree,
                      probs,
                      n_cycles = 1000) {

  if (!is.matrix(probs))
    probs <- matrix(probs, nrow = 1)

  qalys <- NULL
  costs <- NULL

  for (i in seq_len(n_cycles)) {

    probs <- rbind(probs, probs[i, ] %*% tree$trans)
    qalys <- rbind(qalys, probs[i, ]*tree$utils)
  }

  list(probs = probs,
       qalys = qalys)
}
```

By summing over all cycles we obtain the total utilities for each state. The total sum is the expected QALYs value for an individual starting in state `well` until `dead`.

```
init_pop <- c(0,0,0,0,1)
res <- run_model(my_tree, init_pop)
colSums(res$qalys)
```

```
##       dead         CA     cancer       AIDS       well
##  0.0000000  0.2134372  3.8587613  4.0724888 82.3270612
```

The total expected QALYs are therefore

```
sum(res$qalys)
```

```
## [1] 90.47175
```

C5. Roll back Markov-cycle tree

Let's write a recursive function to do the value iteration. Because this is more general than a simple binary tree we need to sum over the number of to-nodes at each step. Also, we need to limit the number of recursions since this function would run until we get a stack overflow error. In the original paper, Hazen (1992) gives a table for 1, 2, 3, 10, 100, 1000 cylces to show convergence. Here we inlude a `limit` argument which exits the function call after a certain tree depth is reached.

```
value_iteration <- function(n,        # starting node number
                            R,        # `reward` (utility/quality factor)
                            p,        # transition probabilty matrix
                            cycle,    # exits recursion after limit tree depth
                            limit = 100) {

  ## sub NAs so returns for absorbing state
  p[p == 0] <- NA
  p[p == 1] <- NA

  to_node <- which(!is.na(p[n, ]))

  if (length(to_node) == 0 || cycle == limit) {
    return(R[n])
  }
  else {
    Vsum <- 0

    for (i in seq_along(to_node)) {
      Vsum <- Vsum + p[n, to_node[i]]*value_iteration(to_node[i], R, p,
                                                      cycle = cycle + 1, limit = limit)
    }

    return(R[n] + Vsum)
  }
}
```

If we run this for the cycles in Table 2 in Hazen (1992), omitting the 1000 cycle because it takes too long to run, then we get the following same values

```
map_dbl(c(1,2,3,10,100),
        function(x) value_iteration(n = 5,
                                    R = my_tree$utils, p = my_tree$trans,
                                    cycle = 1, limit = x))
```

```
## [1]  1.000000  1.993599  2.980485  9.680872 63.018813
```

The problem with this representation is that it grows exponenitally with the number of cycles. One way to speed things up is to do some of the calculation up-front so that we only do it once. We can achieve this by nesting a second function inside of the initial as follows.

```
value_iteration2 <- function(n,        # starting node number
                             R,        # `reward` (utility/quality factor)
                             p,        # transition probabilty matrix
                             cycle,    # exits recursion after limit tree depth
                             limit = 100) {

  ## sub NAs so returns for absorbing state
  p[p == 0] <- NA
  p[p == 1] <- NA

  to_node <- map(seq(nrow(p)), function(i){ which(!is.na(p[i,])) } )

  v_iter <- function(n, cycle) {

    if (length(to_node) == 0 || cycle == limit) {
      return(R[n])
    }
    else {
      Vsum <- 0
      for (i in seq_along(to_node[[n]])) {
        Vsum <- Vsum + p[n, to_node[[n]][i]]*v_iter(to_node[[n]][i],cycle = cycle + 1)
      }
      return(R[n] + Vsum)
    }
  }

  return(v_iter(n, cycle))
}
```

Although we still have the original main problem this does appreciably improve run time.

```
microbenchmark::microbenchmark(
  value_iteration(n = 5,
                  R = my_tree$utils,
                  p = my_tree$trans,
                  cycle = 1, limit = 200),
  value_iteration2(n = 5,
                   R = my_tree$utils,
                   p = my_tree$trans,
                   cycle = 1, limit = 200), times = 1)
```

```
## Unit: seconds
##                                                                                       expr
##   value_iteration(n = 5, R = my_tree$utils, p = my_tree$trans,      cycle = 1, limit = 200)
##  value_iteration2(n = 5, R = my_tree$utils, p = my_tree$trans,      cycle = 1, limit = 200)
##       min       lq     mean   median       uq      max neval
##  47.00826 47.00826 47.00826 47.00826 47.00826 47.00826     1
##  18.71777 18.71777 18.71777 18.71777 18.71777 18.71777     1
```