# Appendix 6: R code listings for example

## MGF method

# Load necessary libraries

library(tidyverse)

library(flexsurv)

# [... Define parameters ...]

# Probability density and cumulative distribution functions

f\_X1\_treatment <- function(x) dweibull(x, scale = lambda\_treatment, shape = k1)

F\_X1\_treatment <- function(x) pweibull(x, scale = lambda\_treatment, shape = k1)

f\_X1\_control <- function(x) dweibull(x, scale = lambda\_control, shape = k1)

F\_X1\_control <- function(x) pweibull(x, scale = lambda\_control, shape = k1)

f\_X2 <- function(x) dgompertz(x, rate = b2, shape = a2)

F\_X2 <- function(x) pgompertz(x, rate = b2, shape = a2)

#' Convenience function for calculating discounted QALYs accrued between A and B

#'

#' @param mgf\_a\_0 M\_A^(0)(-r)

#' @param mgf\_a\_1 M\_A^(1)(-r)

#' @param mgf\_a\_2 M\_A^(2)(-r)

#' @param mgf\_b\_0 M\_B^(0)(-r)

#' @param mgf\_b\_1 M\_B^(1)(-r)

#' @param mgf\_b\_2 M\_B^(2)(-r)

MGF\_qaly <- function(mgf\_a\_0, mgf\_a\_1, mgf\_a\_2, mgf\_b\_0, mgf\_b\_1, mgf\_b\_2) {

((

(u0\*mgf\_a\_0 + u1\*mgf\_a\_1 + u2\*mgf\_a\_2)\*r^2 +

(u1\*mgf\_a\_0 + 2\*u2\*mgf\_a\_1)\*r +

(2\*u2\*mgf\_a\_0)

) - (

(u0\*mgf\_b\_0 + u1\*mgf\_b\_1 + u2\*mgf\_b\_2)\*r^2 +

(u1\*mgf\_b\_0 + 2\*u2\*mgf\_b\_1)\*r +

(2\*u2\*mgf\_b\_0)

))/r^3

}

# Calculate probability X1 < X2 given receive treatment

p\_treatment <- integrate(

function(x) { f\_X2(x)\*F\_X1\_treatment(x) },

lower = 0,

upper = Inf,

rel.tol = 1e-8

)$value

# Calculate probability X1 < X2 given receive control

p\_control <- integrate(

function(x) { f\_X2(x)\*F\_X1\_control(x) },

lower = 0,

upper = Inf,

rel.tol = 1e-8

)$value

# Calculate EMGFs for all variables with j = 0, 1, 2

GM\_X1\_treatment <- map\_dbl(

0:2,

~ integrate(

function(x) { x^(.)\*exp(-r\*x) \* f\_X1\_treatment(x) \* (1-F\_X2(x)) / p\_treatment },

lower = 0,

upper = Inf,

rel.tol = 1e-8

)$value

)

GM\_X1\_control <- map\_dbl(

0:2,

~ integrate(

function(x) { x^(.)\*exp(-r\*x) \* f\_X1\_control(x) \* (1-F\_X2(x)) / p\_control },

lower = 0,

upper = Inf,

rel.tol = 1e-8

)$value

)

GM\_X2\_treatment <- map\_dbl(

0:2,

~ integrate(

function(x) { x^(.)\*exp(-r\*x) \* f\_X2(x) \* (1-F\_X1\_treatment(x)) / (1-p\_treatment) },

lower = 0,

upper = Inf,

rel.tol = 1e-8

)$value

)

GM\_X2\_control <- map\_dbl(

0:2,

~ integrate(

function(x) { x^(.)\*exp(-r\*x) \* f\_X2(x) \* (1-F\_X1\_control(x)) / (1-p\_control) },

lower = 0,

upper = Inf,

rel.tol = 1e-8

)$value

)

GM\_X3 <- map\_dbl(

0:2,

~ integrate(

function(x) { x^(.)\*exp(-r\*x) \* dlnorm(x, meanlog = mu3, sdlog = sigma3) },

lower = 0,

upper = Inf,

rel.tol = 1e-8

)$value

)

# Combine EMGFs to calculate costs and QALYs

MGF <- data.frame(

arm = factor(c("Treatment", "Control")),

cost\_stable = c(

c\_treatment / r \* (1 - p\_treatment\*GM\_X1\_treatment[1] - (1-p\_treatment)\*GM\_X2\_treatment[1]),

c\_control / r \* (1 - p\_control\*GM\_X1\_control[1] - (1-p\_control)\*GM\_X2\_control[1])

),

cost\_progression = c(p\_treatment, p\_control) \* c\_progression \*

c(GM\_X1\_treatment[1], GM\_X1\_control[1]),

cost\_death = c\_death \* c(

p\_treatment \* GM\_X1\_treatment[1] \* GM\_X3[1] + (1-p\_treatment) \* GM\_X2\_treatment[1],

p\_control \* GM\_X1\_control[1] \* GM\_X3[1] + (1-p\_control) \* GM\_X2\_control[1]

),

cost\_progressive = c(p\_treatment, p\_control) \* c\_pd / r \*

c(GM\_X1\_treatment[1], GM\_X1\_control[1]) \* (1 - GM\_X3[1]),

QALY\_stable = v\_sd\*c(

p\_treatment\*MGF\_qaly(1, 0, 0, GM\_X1\_treatment[1], GM\_X1\_treatment[2], GM\_X1\_treatment[3]) +

(1-p\_treatment)\*MGF\_qaly(1, 0, 0, GM\_X2\_treatment[1], GM\_X2\_treatment[2], GM\_X2\_treatment[3]),

p\_control\*MGF\_qaly(1, 0, 0, GM\_X1\_control[1], GM\_X1\_control[2], GM\_X1\_control[3]) +

(1-p\_control)\*MGF\_qaly(1, 0, 0, GM\_X2\_control[1], GM\_X2\_control[2], GM\_X2\_control[3])

),

QALY\_progressive = v\_pd \* c(p\_treatment, p\_control) \* c(

MGF\_qaly(

GM\_X1\_treatment[1], GM\_X1\_treatment[2], GM\_X1\_treatment[3],

GM\_X1\_treatment[1]\*GM\_X3[1],

GM\_X1\_treatment[2]\*GM\_X3[1] + GM\_X1\_treatment[1]\*GM\_X3[2],

GM\_X1\_treatment[3]\*GM\_X3[1] + 2\*GM\_X1\_treatment[2]\*GM\_X3[2] + GM\_X1\_treatment[1]\*GM\_X3[3]

),

MGF\_qaly(

GM\_X1\_control[1], GM\_X1\_control[2], GM\_X1\_control[3],

GM\_X1\_control[1]\*GM\_X3[1],

GM\_X1\_control[2]\*GM\_X3[1] + GM\_X1\_control[1]\*GM\_X3[2],

GM\_X1\_control[3]\*GM\_X3[1] + 2\*GM\_X1\_control[2]\*GM\_X3[2] + GM\_X1\_control[1]\*GM\_X3[3]

)

)

) %>% transmute(

arm = arm,

cost = cost\_stable + cost\_progression + cost\_death + cost\_progressive,

QALY = QALY\_stable + QALY\_progressive,

NMB = QALY \* threshold - cost

)

## Discrete event simulation

# Load necessary libraries

library(dplyr)

library(flexsurv)

# [... Define parameters ...]

# Convenience function to calculate discounted QALYs

qaly <- function(a, b) {

(exp(-r\*a)\*((a^2\*u2+a\*u1+u0)\*r^2 + (2\*a\*u2+u1)\*r + 2\*u2)-exp(-r\*b)\*((b^2\*u2+b\*u1+u0)\*r^2 + (2\*b\*u2+u1)\*r + 2\*u2))/r^3

}

DES <- data.frame(

# Generate TTE random variables

iter = seq(1, n\_DES),

X1.treatment = rweibull(n = n\_DES, scale = lambda\_treatment, shape = k1),

X1.control = rweibull(n = n\_DES, scale = lambda\_control, shape = k1),

X2 = rgompertz(n = n\_DES, rate = b2, shape = a2),

X3 = rlnorm(n = n\_DES, meanlog = mu3, sdlog = sigma3)

) %>% mutate(

# Calculate path through model

progressed.treatment = (X1.treatment < X2),

progressed.control = (X1.control < X2),

# Calculate time in Stable state

LY\_stable.treatment = pmin(X1.treatment, X2),

LY\_stable.control = pmin(X1.control, X2),

# Calculate total time

LY.treatment = if\_else(progressed.treatment, X1.treatment + X3, X2),

LY.control = if\_else(progressed.control, X1.control + X3, X2),

# Calculate QALYs

QALY\_stable.treatment = v\_sd \* qaly(0, LY\_stable.treatment),

QALY\_stable.control = v\_sd \* qaly(0, LY\_stable.control),

QALY\_progressive.treatment = if\_else(progressed.treatment, v\_pd \* qaly(X1.treatment, X1.treatment+X3), 0),

QALY\_progressive.control = if\_else(progressed.control, v\_pd \* qaly(X1.control, X1.control+X3), 0),

QALY.treatment = QALY\_stable.treatment + QALY\_progressive.treatment,

QALY.control = QALY\_stable.control + QALY\_progressive.control,

# Calculate cost components

cost\_stable.treatment = c\_treatment / r \* (1 - exp(-r \* LY\_stable.treatment)),

cost\_stable.control = c\_control / r \* (1 - exp(-r \* LY\_stable.control)),

cost\_progression.treatment = if\_else(progressed.treatment, c\_progression\*exp(-r\*X1.treatment), 0),

cost\_progression.control = if\_else(progressed.control, c\_progression\*exp(-r\*X1.control), 0),

cost\_death.treatment = c\_death \* exp(-r\*LY.treatment),

cost\_death.control = c\_death \* exp(-r\*LY.control),

cost\_progressive.treatment = if\_else(progressed.treatment, c\_pd / r \* (exp(-r\*X1.treatment) - exp(-r\*(X1.treatment+X3))), 0),

cost\_progressive.control = if\_else(progressed.control, c\_pd / r \* (exp(-r\*X1.control) - exp(-r\*(X1.control+X3))), 0),

# Calculate total costs

cost.treatment = cost\_stable.treatment + cost\_progression.treatment + cost\_death.treatment + cost\_progressive.treatment,

cost.control = cost\_stable.control + cost\_progression.control + cost\_death.control + cost\_progressive.control

)

## Markov cohort simulation

# Load necessary libraries

library(heemod)

library(tidyverse)

# Function to prepare Markov model for given cycle length and tunnel

# state limit

prep\_MM <- function(.cycle\_length, .state\_time\_limit) {

lst(

cycle\_length = .cycle\_length,

state\_time\_limit = .state\_time\_limit,

par\_mod = define\_parameters(

# Model parameters

dr\_annual = 0.035,

cycle\_length = .cycle\_length,

dr = rescale\_discount\_rate(dr\_annual, 1, cycle\_length),

model\_years = (model\_time - 1) \* cycle\_length,

u0 = 0.95,

u1\_abs = 0.002,

u1 = -u1\_abs,

u2\_abs = 0.0005,

u2 = -u2\_abs,

v\_sd = 0.9,

v\_pd = 0.6,

c\_treatment = 480,

c\_control = 200,

c\_progression = 3000,

c\_death = 5000,

c\_pd = 1000,

lambda\_control = 1.5,

hr\_treatment = 0.56,

k1 = 2,

lambda\_treatment = lambda\_control \* hr\_treatment ^ (-1 / k1),

a2 = 0.4,

b2 = 0.1,

mu3 = 0,

sigma3 = 1,

# Cumulative hazard of progression

CumHaz\_progress\_control\_now = ((markov\_cycle - 1) \* cycle\_length / lambda\_control) ^

k1,

CumHaz\_progress\_control\_next = (markov\_cycle \* cycle\_length / lambda\_control) ^

k1,

CumHaz\_progress\_control\_incr = CumHaz\_progress\_control\_next - CumHaz\_progress\_control\_now,

CumHaz\_progress\_treatment\_now = ((markov\_cycle - 1) \* cycle\_length / lambda\_treatment) ^

k1,

CumHaz\_progress\_treatment\_next = (markov\_cycle \* cycle\_length / lambda\_treatment) ^

k1,

CumHaz\_progress\_treatment\_incr = CumHaz\_progress\_treatment\_next - CumHaz\_progress\_treatment\_now,

# Cumulative hazard of death

CumHaz\_death\_stable\_now = b2 / a2 \* (exp(a2 \* (markov\_cycle - 1) \* cycle\_length) - 1),

CumHaz\_death\_stable\_next = b2 / a2 \* (exp(a2 \* markov\_cycle \* cycle\_length) - 1),

CumHaz\_death\_stable\_incr = CumHaz\_death\_stable\_next - CumHaz\_death\_stable\_now,

CumHaz\_sum\_control = CumHaz\_progress\_control\_incr + CumHaz\_death\_stable\_incr,

CumHaz\_sum\_treatment = CumHaz\_progress\_treatment\_incr + CumHaz\_death\_stable\_incr,

# Transition probabilities

p\_remain\_stable\_control = exp(-CumHaz\_sum\_control),

p\_remain\_stable\_treatment = exp(-CumHaz\_sum\_treatment),

p\_progress\_control = if\_else(

CumHaz\_sum\_control > 0,

CumHaz\_progress\_control\_incr / CumHaz\_sum\_control \*

(1 - exp(-CumHaz\_sum\_control)),

0

),

p\_progress\_treatment = if\_else(

CumHaz\_sum\_treatment > 0,

CumHaz\_progress\_treatment\_incr / CumHaz\_sum\_treatment \*

(1 - exp(-CumHaz\_sum\_treatment)),

0

),

p\_death\_stable\_control = if\_else(

CumHaz\_sum\_control > 0,

CumHaz\_death\_stable\_incr / CumHaz\_sum\_control \*

(1 - exp(-CumHaz\_sum\_control)),

0

),

p\_death\_stable\_treatment = if\_else(

CumHaz\_sum\_treatment > 0,

CumHaz\_death\_stable\_incr / CumHaz\_sum\_treatment \*

(1 - exp(-CumHaz\_sum\_treatment)),

0

),

p\_death\_progressive = define\_survival(

distribution = "lnorm",

meanlog = mu3,

sdlog = sigma3

) %>% compute\_surv(time = state\_time, cycle\_length = cycle\_length)

),

# Transition matrix in control arm

mat\_control = define\_transition(

state\_names = c("stable", "progressive", "death"),

p\_remain\_stable\_control, p\_progress\_control, p\_death\_stable\_control,

0, C, p\_death\_progressive,

0, 0, 1

),

# Transition matrix in treatment arm

mat\_treatment = define\_transition(

state\_names = c("stable", "progressive", "death"),

p\_remain\_stable\_treatment, p\_progress\_treatment, p\_death\_stable\_treatment,

0, C, p\_death\_progressive,

0, 0, 1

),

# Stable state payoffs

state\_stable = define\_state(

cost\_undisc = cycle\_length \* dispatch\_strategy(control = c\_control,

treatment = c\_treatment),

QALY\_undisc = cycle\_length \* v\_sd \*

(u0 + u1 \* model\_years + u2 \* model\_years ^ 2),

cost = discount(cost\_undisc, r = dr),

QALY = discount(QALY\_undisc, r = dr)

),

# Progressive state payoffs

state\_progressive = define\_state(

cost\_undisc = cycle\_length \* c\_pd + if\_else(state\_time == 1, c\_progression, 0),

QALY\_undisc = cycle\_length \* v\_pd \*

(u0 + u1 \* model\_years + u2 \* model\_years ^ 2),

cost = discount(cost\_undisc, r = dr),

QALY = discount(QALY\_undisc, r = dr)

),

# Death state payoffs

state\_death = define\_state(

cost\_undisc = if\_else(state\_time == 1, c\_death, 0),

QALY\_undisc = 0,

cost = discount(cost\_undisc, r = dr),

QALY = discount(QALY\_undisc, r = dr)

),

# Description of control arm

strat\_control = define\_strategy(

transition = mat\_control,

stable = state\_stable,

progressive = state\_progressive,

death = state\_death

),

# Description of treatment arm

strat\_treatment = define\_strategy(

transition = mat\_treatment,

stable = state\_stable,

progressive = state\_progressive,

death = state\_death

),

# Model characteristics

time\_horizon = 20,

cycles = ceiling(time\_horizon / .cycle\_length)

)

}

# Function to run a Markov model which has been prepared

do\_MM <- function(prep) {

res\_mod <- run\_model(

parameters = prep$par\_mod,

control = prep$strat\_control,

treatment = prep$strat\_treatment,

init = c(1, 0, 0),

cycles = prep$cycles,

cost = cost,

effect = QALY,

state\_time\_limit = c(

progressive = min(prep$state\_time\_limit, prep$cycles),

death = 1

),

method = "life-table"

)

res\_mod\_summary <- summary(res\_mod, threshold = 20000)

return(

res\_mod\_summary$res\_values %>%

select(arm = .strategy\_names, cost, QALY) %>%

mutate(NMB = QALY\*20000 - cost))

}

prep <- prep\_MM(1/12, 24)

results <- do\_MM(prep)

## Markov microsimulation

# Load necessary libraries

library(tidyverse)

library(flexsurv)

# [... Define parameters ...]

# Multinomial sampling [Krijkamp et al. 2018]

samplev <- function(probs, m) {

d <- dim(probs)

n <- d[1]

k <- d[2]

lev <- dimnames(probs)[[2]]

if (!length(lev))

lev <- 1:k

ran <- matrix(lev[1], ncol = m, nrow = n)

U <- t(probs)

for (i in 2:k) {

U[i, ] <- U[i, ] + U[i - 1, ]

}

if (any((U[k, ] - 1) > 1e-05))

stop("error in multinom: probabilities do not sum to 1")

for (j in 1:m) {

un <- rep(runif(n), rep(k, n))

ran[, j] <- lev[1 + colSums(un > U)]

}

ran

}

# Function to run the microsimulation [adapted from Krijkamp et al. 2018]

# Modifications:

# - Remove function arguments and rely on bindings being provided in

# environment (certain variables/functions were already treated this

# way)

# - Call Probs with 'dur' to allow sojourn time-dependent transition

# probabilities

# - Call Costs with 'dur' to allow sojourn time-dependent costs

# - Call Effs with 't' to allow wall time-dependent utility

# - Calculate 'dur' as time in current state (not time in any disease

# state)

MicroSim <- function(TR.out = TRUE, TS.out = TRUE, Trt = FALSE, seed = 1) {

# TR.out: should the output include a Microsimulation trace? (default is TRUE)

# TS.out: should the output include a matrix of transitions between states? (default is TRUE)

# Trt: are the n.i individuals receiving treatment? (scalar with a Boolean value, default is FALSE)

# seed: starting seed number for random number generator (default is 1)

# Calculate the cost and QALY discount rates

v.dwc <- 1 / (1 + d.c) ^ (0:n.t)

v.dwe <- 1 / (1 + d.e) ^ (0:n.t)

# Create the matrix capturing the state name/costs/health outcomes for all individuals at each time point

m.M <- m.C <- m.E <- matrix(

nrow = n.i,

ncol = n.t + 1,

dimnames = list(paste("ind", 1:n.i, sep = " "),

paste("cycle", 0:n.t, sep = " "))

)

# Initial health state

m.M[, 1] <- v.M\_1

# Set the seed for every individual for the random number generator

set.seed(seed)

# create the dur variable that stores the number of cycles the individual has occupied the current state

# all individuals spend one cycle in the starting state

dur <- rep(1, n.i)

# estimate costs and QALYs per individual for the initial health state

m.C[, 1] <- Costs(m.M[, 1], dur, Trt)

m.E[, 1] <- Effs (m.M[, 1], 0, Trt)

for (t in 1:n.t) {

# calculate the transition probabilities at cycle t

m.p <- Probs(m.M[, t], dur, Trt)

# sample the next health state and store that state in matrix m.M

m.M[, t + 1] <- samplev(prob = m.p, m = 1)

# Increment dur if stayed in the same state, or set to 1 if moved state

remained <- m.M[, t + 1] == m.M[, t]

dur[remained] <- dur[remained] + 1

dur[!remained] <- 1

# estimate costs and QALYs per individual during cycle t + 1 conditional on treatment

m.C[, t + 1] <- Costs(m.M[, t + 1], dur, Trt)

m.E[, t + 1] <- Effs(m.M[, t + 1], t, Trt)

# display the progress of the simulation

cat('\r', paste(round(t / n.t \* 100), "% done", sep = " "))

}

# total (discounted) cost and QALYs per individual

tc <- m.C %\*% v.dwc

te <- m.E %\*% v.dwe

# average (discounted) cost and QALYs

tc\_hat <- mean(tc)

te\_hat <- mean(te)

if (TS.out == TRUE) {

# create a matrix of transitions across states

TS <- paste(m.M, cbind(m.M[, -1], NA), sep = "->")

TS <- matrix(TS, nrow = n.i)

# name the rows and columns

rownames(TS) <- paste("Ind", 1:n.i, sep = " ")

colnames(TS) <- paste("Cycle", 0:n.t, sep = " ")

} else {

TS <- NULL

}

if (TR.out == TRUE) {

TR <- t(apply(m.M, 2, function(x)

table(factor(

x, levels = v.n, ordered = TRUE

))))

# create a distribution trace

TR <- TR / n.i

# name the rows and columns

rownames(TR) <- paste("Cycle", 0:n.t, sep = " ")

colnames(TR) <- v.n

} else {

TR <- NULL

}

results <- list(

m.M = m.M,

m.C = m.C,

m.E = m.E,

tc = tc,

te = te,

tc\_hat = tc\_hat,

te\_hat = te\_hat,

TS = TS,

TR = TR

)

return(results) # return the results

}

# Function to calculate transition probabilities for all individuals

Probs <- function(M\_it, dur, Trt) {

# M\_it: health state occupied by individual i at cycle t (character variable)

# dur: the duration spent in the current state

# Trt: whether patient is receiving Treatment (as opposed to Control)

# create vector of state transition probabilities

m.p.it <- matrix(NA, n.s, n.i)

# assign names to the vector

rownames(m.p.it) <- v.n

# Calculate delta cumulative hazards

# Stable to Progressive

H.SP <-

pweibull(

q = (dur[M\_it == "Stable"]-1)\*cl,

scale = Trt\*lambda\_treatment + (1-Trt)\*lambda\_control,

shape = k1,

lower.tail = FALSE,

log.p = TRUE) -

pweibull(

q = dur[M\_it == "Stable"]\*cl,

scale = Trt\*lambda\_treatment + (1-Trt)\*lambda\_control,

shape = k1,

lower.tail = FALSE,

log.p = TRUE)

# Stable to Dead

H.SD <-

pgompertz(

q = (dur[M\_it == "Stable"]-1)\*cl,

shape = a2,

rate = b2,

lower.tail = FALSE,

log.p = TRUE) -

pgompertz(

q = dur[M\_it == "Stable"]\*cl,

shape = a2,

rate = b2,

lower.tail = FALSE,

log.p = TRUE)

# Progressive to Dead

H.PD <-

plnorm(

q = (dur[M\_it == "Progressive"]-1)\*cl,

meanlog = mu3,

sdlog = sigma3,

lower.tail = FALSE,

log.p = TRUE) -

plnorm(

q = dur[M\_it == "Progressive"]\*cl,

meanlog = mu3,

sdlog = sigma3,

lower.tail = FALSE,

log.p = TRUE)

# Probabilities leaving Stable

H.S <- H.SP + H.SD

p.SS <- exp(-H.S)

p.SP <- H.SP/H.S \* (1 - exp(-H.S))

p.SD <- H.SD/H.S \* (1 - exp(-H.S))

# Probabilities leaving progressive

p.PP <- exp(-H.PD)

p.PD <- 1 - exp(-H.PD)

# update the v.p with the appropriate probabilities

m.p.it[, M\_it == "Stable"] <- rbind(p.SS, p.SP, p.SD)

m.p.it[, M\_it == "Progressive"] <- rbind(0, p.PP, p.PD)

m.p.it[, M\_it == "Dead"] <- c(0, 0, 1)

ifelse(colSums(m.p.it) == 1,

return(t(m.p.it)),

print("Probabilities do not sum to 1")) # return the transition probabilities or produce an error

}

# Function to calculate costs across all individuals

Costs <- function(M\_it, dur, Trt = FALSE) {

# M\_it: health state occupied by individual i at cycle t (character variable)

# dur: the duration spent in the current state

# Trt: is the individual being treated? (default is FALSE)

c.it <- rep(0, length(M\_it))

if (any(M\_it == "Stable"))

c.it[M\_it == "Stable"] <- cl \* (Trt\*c\_treatment + (1-Trt)\*c\_control)

if (any(M\_it == "Progressive"))

c.it[M\_it == "Progressive"] <- c\_progression \* (dur[M\_it == "Progressive"] == 1) + cl \* c\_pd

if (any(M\_it == "Dead"))

c.it[M\_it == "Dead"] <- c\_death \* (dur[M\_it == "Dead"] == 1)

return(c.it)

}

# Function to calculate QALYs across all individuals

Effs <- function(M\_it, t, Trt = FALSE) {

# M\_it: health state occupied by individual i at cycle t (character variable)

# t: the current cycle

# Trt: is the individual treated? (default is FALSE)

u.baseline <- u0 + u1 \* ((t + 0.5) \* cl) + u2 \* ((t + 0.5) \* cl) \*\* 2

u.it <- rep(0, length(M\_it))

u.it[M\_it == "Stable"] <- u.baseline \* v\_sd

u.it[M\_it == "Progressive"] <- u.baseline \* v\_pd

u.it[M\_it == "Dead"] <- 0

# calculate the QALYs during cycle t

QALYs <- u.it \* cl

return(QALYs)

}

# Function factory

# Returns a function with all necessary parameters in a suitable

# environment

prep\_MarkovMicrosim <- function(params, n\_microsim, cycle\_length) {

e <- new.env(parent = globalenv())

e$n.i <- n\_microsim

e$n.t <- floor(20 / cycle\_length)

e$cl <- cycle\_length

e$v.n <- c("Stable", "Progressive", "Dead")

e$n.s <- length(e$v.n)

e$v.M\_1 <- rep("Stable", e$n.i)

e$d.c <- e$d.e <- (1 + params$dr) \*\* cycle\_length - 1

e$v.Trt <- c("Control", "Treatment")

# Copy over parameters

e <- list2env(params, e)

l\_MicroSim <- MicroSim

e$Probs <- Probs

e$Costs <- Costs

e$Effs <- Effs

environment(l\_MicroSim) <- e

environment(e$Probs) <- e

environment(e$Costs) <- e

environment(e$Effs) <- e

return(l\_MicroSim)

}

# Perform the microsimulation given a function from the function

# factory and combine results for control and treatment arms

do\_MarkovMicrosim <- function(microsim) {

control <- microsim(TR.out = FALSE, TS.out = FALSE, Trt = FALSE, seed = 1)

treatment <- microsim(TR.out = FALSE, TS.out = FALSE, Trt = TRUE, seed = 1)

return(

bind\_rows(

tibble(arm = "Control", costs = as.vector(control$tc), QALYs = as.vector(control$te)),

tibble(arm = "Treatment", costs = as.vector(treatment$tc), QALYs = as.vector(treatment$te))

)

)

}

results <- do\_MarkovMicrosim(prep\_MarkovMicrosim(meanParams, 100000, 1/12))