```
#* Script name: Functions_cSTM_time_indep.R
#-------------------------------------------------------------------------------
#
####                         Decision Model
####
#-------------------------------------------------------------------------------
#
#' Decision Model
#'
#' \code{decision_model} implements the decision model used.
#'
#' @param l_params_all List with all parameters of decision model
#' @param verbose Logical variable to indicate print out of messages
#' @return The transition probability array and the cohort trace matrix.
#' @export
decision_model <- function(l_params_all, verbose = FALSE) {
  with(as.list(l_params_all), {
    ########################### Process model inputs
###########################
    ### Process model inputs
    ## Number of cycles
    n_cycles <- (n_age_max - n_age_init)/cycle_length # time horizon,
number of cycles
    ## Cycle-specific transition probabilities to the Dead state
    # compute mortality rates
    r_S1D <- r_HD * hr_S1 # annual mortality rate in the Sick state
    r_S2D <- r_HD * hr_S2 # annual mortality rate in the Sicker state
    # transform rates to probabilities
    p_HS1  <- rate_to_prob(r = r_HS1, t = cycle_length) # constant annual
probability of becoming Sick when Healthy conditional on surviving
    p_S1H  <- rate_to_prob(r = r_S1H, t = cycle_length) # constant annual
probability of becoming Healthy when Sick conditional on surviving
    p_S1S2 <- rate_to_prob(r = r_S1S2, t = cycle_length)# constant annual
probability of becoming Sicker when Sick conditional on surviving
    p_HD   <- rate_to_prob(r = r_HD, t = cycle_length)  # annual mortality
risk in the Healthy state
    p_S1D  <- rate_to_prob(r = r_S1D, t = cycle_length) # annual mortality
risk in the Sick state
    p_S2D  <- rate_to_prob(r = r_S2D, t = cycle_length) # annual mortality
risk in the Sicker state

    ## Annual transition probability of becoming Sicker when Sick for
treatment B
    # apply hazard ratio to rate to obtain transition rate of becoming
Sicker when Sick for treatment B
    r_S1S2_trtB <- r_S1S2 * hr_S1S2_trtB
    # transform rate to probability
    # probability to become Sicker when Sick
    # under treatment B conditional on surviving
    p_S1S2_trtB <- rate_to_prob(r = r_S1S2_trtB, t = cycle_length)

    #################### Construct state-transition models
####################
    ## Initial state vector
    # All starting healthy
    v_m_init <- c(H = 1, S1 = 0, S2 = 0, D = 0) # initial state vector
    # Number of health states
    n_states    <- length(v_m_init)
```

```r
    # Health state names
    v_names_states <- names(v_m_init)

    ## Initialize cohort trace for SoC
    m_M <- matrix(NA,
                  nrow = (n_cycles + 1), ncol = n_states,
                  dimnames = list(0:n_cycles, v_names_states))
    # Store the initial state vector in the first row of the cohort trace
    m_M[1, ] <- v_m_init
    ## Initialize cohort trace for strategies A, B, and AB
    # Structure and initial states are the same as for SoC
    m_M_strA  <- m_M # Strategy A
    m_M_strB  <- m_M # Strategy B
    m_M_strAB <- m_M # Strategy AB

    ## Initialize transition probability matrix for strategy SoC
    # all transitions to a non-death state are assumed to be conditional
on survival
    m_P <- matrix(0,
                  nrow = n_states, ncol = n_states,
                  dimnames = list(v_names_states,
                                  v_names_states)) # define row and column
names
    ## Fill in matrix
    # From H
    m_P["H", "H"]   <- (1 - p_HD) * (1 - p_HS1)
    m_P["H", "S1"]  <- (1 - p_HD) * p_HS1
    m_P["H", "D"]   <- p_HD
    # From S1
    m_P["S1", "H"]  <- (1 - p_S1D) * p_S1H
    m_P["S1", "S1"] <- (1 - p_S1D) * (1 - (p_S1H + p_S1S2))
    m_P["S1", "S2"] <- (1 - p_S1D) * p_S1S2
    m_P["S1", "D"]  <- p_S1D
    # From S2
    m_P["S2", "S2"] <- 1 - p_S2D
    m_P["S2", "D"]  <- p_S2D
    # From D
    m_P["D", "D"]   <- 1

    ## Initialize transition probability matrix for strategy A as a copy
of SoC's
    m_P_strA <- m_P

    ## Initialize transition probability matrix for strategy B
    m_P_strB <- m_P
    # Update only transition probabilities from S1 involving p_S1S2
    m_P_strB["S1", "S1"] <- (1 - p_S1D) * (1 - (p_S1H + p_S1S2_trtB))
    m_P_strB["S1", "S2"] <- (1 - p_S1D) * p_S1S2_trtB

    ## Initialize transition probability matrix for strategy AB as a copy
of B's
    m_P_strAB <- m_P_strB

    ### Check if transition probability matrices are valid
    ## Check that transition probabilities are [0, 1]
    check_transition_probability(m_P,      verbose = TRUE)
    check_transition_probability(m_P_strA, verbose = TRUE)
    check_transition_probability(m_P_strB, verbose = TRUE)
    check_transition_probability(m_P_strAB, verbose = TRUE)
    ## Check that all rows sum to 1
    check_sum_of_transition_array(m_P,      n_states = n_states, n_cycles
= n_cycles, verbose = TRUE)
    check_sum_of_transition_array(m_P_strA, n_states = n_states, n_cycles
= n_cycles, verbose = TRUE)
    check_sum_of_transition_array(m_P_strB, n_states = n_states, n_cycles
= n_cycles, verbose = TRUE)
    check_sum_of_transition_array(m_P_strAB, n_states = n_states, n_cycles
= n_cycles, verbose = TRUE)

    #### Run Markov model ####
    # Iterative solution of time-independent cSTM
    for(t in 1:n_cycles){
      # For SoC
      m_M[t + 1, ] <- m_M[t, ] %*% m_P
      # For strategy A
      m_M_strA[t + 1, ] <- m_M_strA[t, ] %*% m_P_strA
      # For strategy B
      m_M_strB[t + 1, ] <- m_M_strB[t, ] %*% m_P_strB
      # For strategy AB
      m_M_strAB[t + 1, ] <- m_M_strAB[t, ] %*% m_P_strAB
    }

    ## Strategy names
    v_names_str <- c("Standard of care",      # store the strategy names
                     "Strategy A",
                     "Strategy B",
                     "Strategy AB")
    n_str       <- length(v_names_str)        # number of strategies

    ## Store the cohort traces in a list
    l_m_M <- list(m_M,
                  m_M,
                  m_M_strB,
                  m_M_strB)
    names(l_m_M) <- v_names_str

    ###################### RETURN OUTPUT #####################
    return(l_m_M)
  }
  )
}

#------------------------------------------------------------------------------
#
####              Calculate cost-effectiveness outcomes
####
#------------------------------------------------------------------------------
#
#' Calculate cost-effectiveness outcomes
#'
#' \code{calculate_ce_out} calculates costs and effects for a given vector
of parameters using a simulation model.
#' @param l_params_all List with all parameters of decision model
#' @param n_wtp Willingness-to-pay threshold to compute net benefits
#' @return A data frame with discounted costs, effectiveness and NMB.
#' @export
```

```
calculate_ce_out <- function(l_params_all, n_wtp = 100000){ # User defined
  with(as.list(l_params_all), {

    #### Run Markov Model ####
    ## Cohort traces
    l_m_M <- decision_model(l_params_all = l_params_all)

    ## Strategy names
    v_names_str <- c("Standard of care",      # store the strategy names
                     "Strategy A",
                     "Strategy B",
                     "Strategy AB")
    n_str       <- length(v_names_str)        # number of strategies

    #### State Rewards ####
    ## Vector of state utilities under strategy SoC
    v_u_SoC    <- c(H  = u_H,
                    S1 = u_S1,
                    S2 = u_S2,
                    D  = u_D) * cycle_length
    ## Vector of state costs under strategy SoC
    v_c_SoC    <- c(H  = c_H,
                    S1 = c_S1,
                    S2 = c_S2,
                    D  = c_D) * cycle_length
    ## Vector of state utilities under strategy A
    v_u_strA   <- c(H  = u_H,
                    S1 = u_trtA,
                    S2 = u_S2,
                    D  = u_D) * cycle_length
    ## Vector of state costs under strategy A
    v_c_strA   <- c(H  = c_H,
                    S1 = c_S1 + c_trtA,
                    S2 = c_S2 + c_trtA,
                    D  = c_D) * cycle_length
    ## Vector of state utilities under strategy B
    v_u_strB   <- c(H  = u_H,
                    S1 = u_S1,
                    S2 = u_S2,
                    D  = u_D) * cycle_length
    ## Vector of state costs under strategy B
    v_c_strB   <- c(H  = c_H,
                    S1 = c_S1 + c_trtB,
                    S2 = c_S2 + c_trtB,
                    D  = c_D) * cycle_length
    ## Vector of state utilities under strategy AB
    v_u_strAB  <- c(H  = u_H,
                    S1 = u_trtA,
                    S2 = u_S2,
                    D  = u_D) * cycle_length
    ## Vector of state costs under strategy AB
    v_c_strAB  <- c(H  = c_H,
                    S1 = c_S1 + (c_trtA + c_trtB),
                    S2 = c_S2 + (c_trtA + c_trtB),
                    D  = c_D) * cycle_length

    ## Store the vectors of state utilities for each strategy in a list
    l_u   <- list(SQ = v_u_SoC,
                  A  = v_u_strA,
                  B  = v_u_strB,
                  AB = v_u_strAB)
    ## Store the vectors of state cost for each strategy in a list
    l_c   <- list(SQ = v_c_SoC,
                  A  = v_c_strA,
                  B  = v_c_strB,
                  AB = v_c_strAB)

    # assign strategy names to matching items in the lists
    names(l_u) <- names(l_c) <- v_names_str

    ## create empty vectors to store total utilities and costs
    v_tot_qaly <- v_tot_cost <- vector(mode = "numeric", length = n_str)
    names(v_tot_qaly) <- names(v_tot_cost) <- v_names_str

    ## Number of cycles
    n_cycles <- (n_age_max - n_age_init)/cycle_length # time horizon,
number of cycles

    ## Discount weight for costs and effects
    v_dwc  <- 1 / ((1 + d_e * cycle_length) ^ (0:n_cycles))
    v_dwe  <- 1 / ((1 + d_c * cycle_length) ^ (0:n_cycles))

    ## Within-cycle correction (WCC) using Simpson's 1/3 rule
    v_wcc <- darthtools::gen_wcc(n_cycles = n_cycles,
                                 method = "Simpson1/3") # vector of wcc

    #### Loop through each strategy and calculate total utilities and
costs ####
    for (i in 1:n_str) {
      v_u_str <- l_u[[i]]   # select the vector of state utilities for the
i-th strategy
      v_c_str <- l_c[[i]]   # select the vector of state costs for the i-
th strategy

      #### Expected QALYs and costs per cycle ####
      ### Vector of QALYs and Costs
      ## Apply state rewards ###
      v_qaly_str <- l_m_M[[i]] %*% v_u_str # sum the utilities of all
states for each cycle
      v_cost_str <- l_m_M[[i]] %*% v_c_str # sum the costs of all states
for each cycle

      #### Discounted total expected QALYs and Costs per strategy and
apply half-cycle correction if applicable ####
      ## QALYs
      v_tot_qaly[i] <- t(v_qaly_str) %*% (v_dwe * v_wcc)
      ## Costs
      v_tot_cost[i] <- t(v_cost_str) %*% (v_dwc * v_wcc)
    }

    ## Vector with discounted net monetary benefits (NMB)
    v_nmb <- v_tot_qaly * n_wtp - v_tot_cost

    ## data.frame with discounted costs, effectiveness and NMB
    df_ce <- data.frame(Strategy = v_names_str,
                        Cost     = v_tot_cost,
                        Effect   = v_tot_qaly,
                        NMB      = v_nmb)

    return(df_ce)
  }
  )
}

#------------------------------------------------------------------------------
#
####             Generate a PSA input parameter dataset
####
#------------------------------------------------------------------------------
#
#' Generate parameter sets for the probabilistic sensitivity analysis
(PSA)
#'
#' \code{generate_psa_params} generates a PSA dataset of the parameters of
the
#' cost-effectiveness analysis.
#' @param n_sim Number of parameter sets for the PSA dataset
#' @param seed Seed for the random number generation
#' @return A data.frame with a PSA dataset of he parameters of the
#' cost-effectiveness analysis
#' @export
generate_psa_params <- function(n_sim = 1000, seed = 071818){
  set.seed(seed) # set a seed to be able to reproduce the same results
  df_psa <- data.frame(
    # Transition probabilities (per cycle), hazard ratios
    r_HD    = rgamma(n_sim, shape = 20, rate = 10000), # constant rate of
dying when Healthy (all-cause mortality)
    r_HS1   = rgamma(n_sim, shape = 30, rate = 170 + 30), # constant rate
of becoming Sick when Healthy conditional on surviving
    r_S1H   = rgamma(n_sim, shape = 60, rate = 60 + 60),  # constant rate
of becoming Healthy when Sick conditional on surviving
    r_S1S2  = rgamma(n_sim, shape = 84, rate = 716 + 84), # constant rate
of becoming Sicker when Sick conditional on surviving
    hr_S1   = rlnorm(n_sim, meanlog = log(3),  sdlog = 0.01), # hazard
ratio of death in Sick vs Healthy
    hr_S2   = rlnorm(n_sim, meanlog = log(10), sdlog = 0.02), # hazard
ratio of death in Sicker vs Healthy

    # Effectiveness of treatment B
    hr_S1S2_trtB = rlnorm(n_sim, meanlog = log(0.6), sdlog = 0.02), #
hazard ratio of becoming Sicker when Sick under treatment B

    # State rewards
    # Costs
    c_H     = rgamma(n_sim, shape = 100, scale = 20),     # cost of
remaining one cycle in Healthy
    c_S1    = rgamma(n_sim, shape = 177.8, scale = 22.5), # cost of
remaining one cycle in Sick
    c_S2    = rgamma(n_sim, shape = 225, scale = 66.7),   # cost of
remaining one cycle in Sicker
    c_D     = 0,                                          # cost of being
dead (per cycle)
    c_trtA  = rgamma(n_sim, shape = 73.5, scale = 163.3), # cost of
treatment A
```

```
    c_trtB  = rgamma(n_sim, shape = 86.2, scale = 150.8), # cost of
treatment B

    # Utilities
    u_H     = rbeta(n_sim, shape1 = 200, shape2 = 3),     # utility when
Healthy
    u_S1    = rbeta(n_sim, shape1 = 130, shape2 = 45),    # utility when
Sick
    u_S2    = rbeta(n_sim, shape1 = 230, shape2 = 230),   # utility when
Sicker
    u_D     = 0,                                          # utility when
Dead
    u_trtA  = rbeta(n_sim, shape1 = 300, shape2 = 15)     # utility when
being treated with A
  )
  return(df_psa)
}

#' Update parameters
#'
#' \code{update_param_list} is used to update list of all parameters with
new
#' values for specific parameters.
#'
#' @param l_params_all List with all parameters of decision model
#' @param params_updated Parameters for which values need to be updated
#' @return
#' A list with all parameters updated.
#' @export
update_param_list <- function(l_params_all, params_updated){

  if (typeof(params_updated)!="list"){
    params_updated <- split(unname(params_updated),names(params_updated))
#converte the named vector to a list
  }
  l_params_all <- modifyList(l_params_all, params_updated) #update the
values
  return(l_params_all)
}
```