Appendix B: R code for DES example model

#NOTES

# Main script for running model and outputting results.

# Text to the right of the "#" sign are comments and are not processed.

# Comments are an essential part of model code and should be used to describe what the code is doing.

# This promotes transparency and helps avoid and identify errors.

#################################################################
#Start of script#################################################################

#remove previous contents

rm(list=ls())
# Set the working directory

# NOTE: Use the / rather than \ symbol

# NOTE: Put the working directory in "" symbols (to make it a character vector)

setwd("G:/ DSU PLS TSD R model")

# load in DES functions in the file DSU_DesFunctions.R

# NOTE: Must be in the same directory

source("DSU_DesFunctions.R")
# Global variables - placed in base environment

util.init <- 0.7 # initial utility
drc <- 0.035 # discount rate for costs
drq <- 0.035 # discount rate for QALYs

#npats <- 50000 # number of patients
npats <- 500 # use this if running through line by line

# utility multipliers for hip fractures and vert fractures
umult.hfrac <- 0.75
umult.vfrac <- 0.90

cost.hfrac <- 7000
cost.vfrac <- 3000

cost.int <- 500 # annual cost of intervention
mortprobhip <- 0.05 # probability of dying from a hip fracture

# Debug mode flag: this will print out the event list and accrued costs and qalys
# as the simulation is being run
dbg <- T

# Flag for whether individual level patient details should be stored and output
ind <- T

if (ind==T){
  PatData <- vector("list", length=npats) # empty list with 50000 elements
}
# Run the simulation
```r
set.seed(1)  # or any integer
RunSim()    # run simulation
```

# to fix simulation results - use set.seed. Useful for debugging as removes random variation

# examine results
```r
tot.costs.int # total (discounted) costs for intervention arm
tot.qalys.int # total (discounted) qalys for intervention arm
```
tot.costs.noint # total (discounted) costs for no intervention arm

tot.qalys.noint # total (discounted) qalys for no intervention arm

tot.dcosts # Total difference in costs between intervention and no intervention pairs

tot.dqalys # Total difference in qalys between intervention and no intervention pairs

# means

tot.costs.int/npats

tot.qalys.int/npats

tot.costs.noint/npats

tot.qalys.noint/npats

tot.dcosts/npats

tot.dqalys/npats

# With ind mode: lapply function to count the number of events

# events
# Number of deaths not relating to hip fractures in intervention arm

```
sum(sapply(PatData, function(x) x$int$deathother))
```

# Number of deaths from hip fractures in intervention arm

```
sum(sapply(PatData, function(x) x$int$deathhip))
```

# Total number of hip fractures in intervention arm

```
sum(sapply(PatData, function(x) x$int$nhip))
```

# Total number of vert fractures in intervention arm

```
sum(sapply(PatData, function(x) x$int$nvert))
```

# For non intervention arm equivalents of above, change 'int' to 'noint'

```
sum(sapply(PatData, function(x) x$noint$deathother))
```

```
sum(sapply(PatData, function(x) x$noint$deathhip))
```

```
sum(sapply(PatData, function(x) x$noint$nhip))
```
sum(sapply(PatData, function(x) x$noint$nvert))

# To access individual patient information for a single patient

# Say the 37th patient...

PatData[[37]]$noint # No intervention copy of the patient

PatData[[37]]$int # Intervention copy of the same patient

# To access the event list for a particular patient

PatData[[37]]$int$evtlist
PatData[[37]]$noint$evtlist

##################################################################################
#output results to excel###########################################################
##################################################################################
# Create object for exporting to Excel

```r
ExcelData <- data.frame(
  ptnum=1:npats,
  itthip=rep(NA, npats),
  ittvert1=rep(NA, npats),
  ittvert2=rep(NA, npats),
  ittdeath=rep(NA, npats),
  ihipcount=rep(NA, npats),
  ivertcount=rep(NA, npats),
  idthhip=rep(NA,npats),
  idthall=rep(NA, npats),
  icosts=rep(NA, npats),
  iqalys=rep(NA, npats),
  ntthip=rep(NA, npats),
  nttvert1=rep(NA, npats),
  nttvert2=rep(NA, npats),
  nttdeath=rep(NA, npats),
```
nhipcount=rep(NA, npats),
nvertcount=rep(NA, npats),
ndthhip=rep(NA, npats),
ndthall=rep(NA, npats),
ncosts=rep(NA, npats),
nqalys=rep(NA, npats)
}

for (i in 1:npats){

itthip <- PatData[[i]]$int$evtlist$evttime[PatData[[i]]$int$evtlist$evtname=="hfrac"]
ntthip <- PatData[[i]]$noint$evtlist$evttime[PatData[[i]]$noint$evtlist$evtname=="hfrac"]

ittvert <- PatData[[i]]$int$evtlist$evttime[PatData[[i]]$int$evtlist$evtname%in%c("vfrac1","vfrac2")]
nntvert <- PatData[[i]]$noint$evtlist$evttime[PatData[[i]]$noint$evtlist$evtname%in%c("vfrac1","vfrac2")]
ittdeath <- PatData[[i]]$int$evtlist$evttime[PatData[[i]]$int$evtlist$evtname=="death"]
nttdeath <- PatData[[i]]$noint$evtlist$evttime[PatData[[i]]$noint$evtlist$evtname=="death"]

ExcelData[i,"itthip"] <- itthip
ExcelData[i,"ntthip"] <- ntthip
ExcelData[i,c("ittvert1", "ittvert2")]<- ittvert
ExcelData[i,c("nttvert1", "nttvert2")]<- nttvert
ExcelData[i,"ittdeath"] <- ittdeath
ExcelData[i,"nttdeath"] <- nttdeath

ExcelData[i,"ihipcount"] <- PatData[[i]]$int$nhip
ExcelData[i, "ivertcount"] <- PatData[[i]]$int$nvert
ExcelData[i,"nhipcount"] <- PatData[[i]]$noint$nhip
ExcelData[i,"nvertcount"] <- PatData[[i]]$noint$nvert
ExcelData[i,"idthhip"] <- PatData[[i]]$int$deathhip
ExcelData[i,"idthall"] <- PatData[[i]]$int$deathother
ExcelData[i,"ndthhip"] <- PatData[[i]]$noint$deathhip
ExcelData[i,"ndthall"] <- PatData[[i]]$noint$deathother
ExcelData[i, "icosts"] <- PatData[[i]]$int$thscosts
ExcelData[i, "iqalys"] <- PatData[[i]]$int$thsqalys

ExcelData[i, "ncosts"] <- PatData[[i]]$noint$thscosts
ExcelData[i, "nqalys"] <- PatData[[i]]$noint$thsqalys

}

write.csv(ExcelData, file="DSU_PatData.csv")

Appendix A2: Functions called by the main script

# Functions called by DSU_DesScript.R

# Text to the right of the "#" sign are comments and are not processed.
# Comments are an essential part of model code and should be used to describe what the code is doing.
# This promotes transparency and helps avoid and identify errors.

# Initialise the event list for the no intervention group
# up to two vfracs could occur

InitEventList.noint <- function(){

  vfracs <- rweibull(n=2, shape=2, scale=8)  # This code simulates a vector of 2 independent Weibull variates, indicating sojourn times to vertebral fracture
  hfrac  <- rweibull(n=1, shape=4, scale=10)  # Similarly, this code simulates a time to hip fracture
  death  <- rnorm(n=1, mean=12, sd=3)       # As above, but now time to death using normal distribution
  output <- data.frame(

    evtname=c(  # This code assigns a vector of names stored as strings
      "hfrac",  # This allows the user to refer to the corresponding event times by name
      "vfrac1",  # It is good practice to assign names to elements of the model
      "vfrac2",
      "death"
    ),

    evttime=c(  # This code assigns the event times to a vector, which can be referred to later in the model
    )
    )

}
Note that using square brackets \([\) selects values from a vector, in this case the "vfracs" vector:

```r
vfracs[1],  # vfracs[1] + vfracs[2],
max(0, rnorm(n=1, mean=12, sd=3))  # the normal distribution used above has non-zero probability of returning a negative value
```

Using the max function prevents this from occurring.

# Sort event list by evttime
output <- output[order(output$evttime),]

if (ind==T){
  this.PatData$noint$evtlist <<- output
}
return(output)
InitEventList.int : input is output from InitEventList.noint

InitEventList.int <- function(input){
  output <- input

  # find the hfrac event and double the time to it
  thisrow <- which(output$evtname=="hfrac") # note reference to vector of names
  output$evttime[thisrow] <- 2 * output$evttime[thisrow] # Reassign to output vector

  # find the sojourn times for vfrac events and doubles the time to the first one.
  thisrow <- which(output$evtname=="vfrac1")
  vf1 <- output$evttime[thisrow]
  thisrow <- which(output$evtname=="vfrac2")
  vf2 <- output$evttime[thisrow] - vf1 # This returns the sojourn time rather than the raw event time

  vf1 <- 2 * vf1 # Time to first vfrac is doubled
  vf2 <- vf1 + vf2 # Time to second vfrac FROM time to first vfrac is NOT doubled, it is simply added to the updated time to first vert frac
output$evttme[which(output$evtnme=="vfrac1")]<-vf1 # Assign these fracture times to output vector
output$evttme[which(output$evtnme=="vfrac2")]<-vf2

#sort event list by evttme
output<-output[order(output$evttme),]

if (ind==T){
  this.PatData$int$evtlist <<- output  # This code stores individual patient output if this option has been chosen (see DesScript code)
}

  # Useful for debugging and validation, but increases memory requirements and may slow computation

return(output)

}

# AddOngoing: Calculate additional qalys and costs accrued from previous to current event

# Inputs:
# Icldrq : local version of discount rate for qalys
# Icldrc : local version of discount rate for costs
# lclprvtime: time of previous event

# lclcurtime: time of current event

# lclvalq : fixed value of qalys

# lclvalc : fixed value of costs

InstantDiscount <- function(rate){  
  # For compound continuous discounting - use INSTANTANEOUS rate.
  log(1+rate)
}

AddOngoing <- function(lcldrq=0.035, lcldrc=0.035, lclprvtime, lclcurtime, lclvalq, lclvalc){

  Instantdrq <- InstantDiscount(lcldrq)
  Instantdrc <- InstantDiscount(lcldrc)

  # calculate additional qalys
  addqalys <- ((lclvalq)/(0 - Instantdrq)) * (exp(lclcurtime * (0 - Instantdrq)) - exp(lclprvtime * (0 - Instantdrq)))

  # calculate additional costs
addcosts <- ((lclvalc)/(0 - Instantdrc)) * (exp(lclcurtime * (0 - Instantdrc)) - exp(lclprvtime * (0 - Instantdrc)))

# combine additional costs and additional qalys in a list
output <- list(addqalys=addqalys, addcosts=addcosts)

return(output)
}

#############################################################

# Add Instantaneous costs and qalys

# Inputs:
# lcldrq : local version of discount rate for qalys
# lcldrc : local version of discount rate for costs
# lclcurtime: time of current event
# lclvalq : fixed value of qalys
# lclvalc : fixed value of costs
AddInstant <- function(lcldrq=0.035, lcldrc=0.035, lclcurtime, lclvalq, lclvalc){

  addinstqalys <- lclvalq * ((1+lcldrq)^(-lclcurtime))    # Note use of DISCRETE TIME discounting for instantaneous costs and benefits
  addinstcosts <- lclvalc * ((1+lcldrc)^(-lclcurtime))

  # combine additional costs and additional qalys in a list
  output <- list(addinstqalys=addinstqalys, addinstcosts=addinstcosts)

  return(output)
}

GetNxtEvt <- function(intervention=F){  # This function identifies which event is to be processed next for each patient, depending on intervention

  if (intervention==F){    # It loops through events until death has occurred, either due to hip fracture or other causes
    # Do the following if no intervention
    # if there is at least one event still to process
    if (dim(evtlist.noint)[1] > 0){
      nextevt <- evtlist.noint$evname[1]
      nextevtttime <- evtlist.noint$evtime[1]
# remove next event as it has now been processed

# Debugging line: If debugging is enabled (see DesScript.R) then this will print out the next event.
# This helps detect errors in model logic (e.g. fracture occurs after death).
if (dbg==T){
   print(evtlist.noint)
}
evtlist.noint <<- evtlist.noint[-1,]
output <- list(evt=nextevt, evttime=nextevttime)
} else {
   output <- NULL
}
}
} else {
   # do the following if intervention
if (dim(evtlist.int)[1]> 0){
   nextevt <- evtlist.int$evtname[1]
   nextevtime <- evtlist.int$evttime[1]
   # remove next event as it has now been processed
# Debugging line:
if (dbg==T){
    print(evtlist.int)
}
evtlist.int <<- evtlist.int[-1,]
output <- list(evt=nextevt, evtt ime=nextevtt ime)
} else {
    output <- NULL
}

return(output)

# ReactEvt : react to the next event
# thisevt : a two element list containing the output from GetNextEvt
# $evt : event
# $evtt ime : event time
# intervention: boolean flag : TRUE or FALSE

ReactEvt <- function(thisevt, intervention){  # This function processes the next event (as identified in the GetNextEvt function)

  evt <- thisevt$evt  # Identify event type
  prevtime <- curtime  # Identify time of previous event
  curtime <<- thisevt$evttime  # Identify time of next event

  if (intervention==F){
    # No intervention logic
    if (evt=="death"){
      if(ind==T){
        this.PatData$noint$deathother <<- 1  # This indicates that the patient is now dead - no further events
      }
    }
    # create variable to additional ongoing costs and qalys
    additionals <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=0)
    instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=0)
thsqalys <- thsqalys + additionals$addqalys + instadditionals$addinstqalys
thscosts <- thscosts + additionals$addcosts + instadditionals$addinstcosts

curtime <<- Inf # Set current time to Infinity so patient level loop stops

} else if (evt %in% c("vfrac1","vfrac2")){
  if (ind==T){
    this.PatData$noint$nvert <<- this.PatData$noint$nvert + 1 # This records the number of vertebral fractures experienced AFTER event occurs
  }

  if (prvvert==F){
    # ================
    # [Logic if no previous vert fracture]
    # ================

    additionals <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=0)
    instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=cost.vfrac)
thsqalys <- thsqalys + additionals$addqalys + instadditionals$addinstqalys
ths costs <- thscosts + additionals$addcosts + instadditionals$addinstcosts

# utility multiplier at previous
utilmlt <- utilmlt * umult.vfrac  # Utility is set to previous utility (utilmlt) multiplied by the multiplier for a vertebral fracture
prvvert <- T
}
else {

#====================================
# [Logic if a previous vert fracture]
#====================================

additionals <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=0)
instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=cost.vfrac)

ths qalys <- thsqalys + additionals$addqalys + instadditionals$addinstqalys
ths costs <- thscosts + additionals$addcosts + instadditionals$addinstcosts
}
else if (evt="hfrac"){
if (ind==T){
    this.PatData$noint$nhip <<- this.PatData$noint$nhip + 1  # Records that patient had a hip fracture
}

# ==-------------------------------------------
# Hip fracture logic
# ==-------------------------------------------

additionals <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=0)
instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=cost.hfrac)

thsqalys <<- thsqalys + additionals$addqalys + instadditionals$addinstqalys
thscosts <<- thscosts + additionals$addcosts + instadditionals$addinstcosts

utilmlt <<- utilmlt * umult.hfrac  # as with vertebral - multiply previous utility by hip fracture modifier

# Death occurs with 5% probability

patdies <- runif(1) < mortprobhip  # This code generates a uniform variate and compares it with the probability of hip fracture related mortality

    # If the variate is less than the probability (0.05 in this example), "patdies" is set to TRUE
if (patdies) {
    if (ind==T) {
        this.PatData$noint$deathhip <<- this.PatData$noint$deathhip + 1  # This indicates that the patient died of a hip fracture
    }
    curtime <<- Inf
} else {
    # if this is reached, then something has gone wrong
    stop("Event type not recognised")  # Use debugging to identify why this occurred and fix error as appropriate
}

else { #
    #==============================================================
    # Intervention logic - follows same format as non-intervention logic
    #==============================================================
    if (evt=="death"){
        if (ind==T){
            this.PatData$int$deathother <<- this.PatData$int$deathother + 1
        }
    }
}
# create variable to additional ongoing costs and qalys

```
additionals <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=cost.int)

instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=0)
```

```
thsqalys <- thsqalys + additionals$addqalys + instadditionals$addinstqalys

thscosts <- thscosts + additionals$addcosts + instadditionals$addinstcosts
```

```
curtime <- Inf # Set current time to Infinity so patient level loop stops
```

# create variable to additional ongoing costs and qalys

} else if (evt %in% c("vfrac1","vfrac2")){

  if (ind==T){
    this.PatData$int$nvert <- this.PatData$int$nvert + 1
  }

  if (prvvert==F){
    if (prvvert==F){
# [Logic if no previous vert fracture]

additionals <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=cost.int)

instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=cost.vfrac)

thsqalys <- thsqalys + additionals$addqalys + instadditionals$addinstqalys
ths costs <- thscosts + additionals$addcosts + instadditionals$addinstcosts

# utility multiplier at previous
utilmlt <- utilmlt * umult.vfrac
prvvert <- T
}

else {

# [Logic if a previous vert fracture]

additionals <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=cost.int)
instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=cost.vfrac)

thsqalys <- thsqalys + additions$addqalys + instadditionals$addinstqalys
thscosts <- thscosts + additions$addcosts + instadditionals$addinstcosts

} else if (evt=="hfrac"){
  if (ind==T){
    this.PatData$int$nhip <- this.PatData$int$nhip + 1
  }
  additions <- AddOngoing(lclprvtime=prevtime, lclcurtime=curtime, lclvalq=utilmlt, lclvalc=cost.int)
  instadditionals <- AddInstant(lclcurtime=curtime, lclvalq=0, lclvalc=cost.hfrac)

  thsqalys <- thsqalys + additions$addqalys + instadditionals$addinstqalys
  thscosts <- thscosts + additions$addcosts + instadditionals$addinstcosts
  utilmlt <- utilmlt * umult.hfrac

  # Death occurs with 5% probability
  patdies <- runif(1) < mortprobhip
if (patdies) {
    if (ind==T) {
        this.PatData$int$deathhip <<- this.PatData$int$deathhip + 1
    }
    curtime <<- Inf
}

} else {
    # if this is reached, then something has gone wrong
    stop("Event type not recognised")
}

# set to return nothing (NULL object) so earlier operations are not returned by accident
return(NULL)
# Enclose simulation within a function

RunSim <- function(){

# initialise variable of total costs and QALYs (note use of superassignment operator (<<-))

tot.qalys.noint <<- 0 # total QALYs accrued by all patients - no intervention

tot.costs.noint <<- 0 # total costs accrued by all patients - no intervention


tot.qalys.int <<- 0 # total QALYs accrued by all patients - intervention

tot.costs.int <<- 0 # total costs accrued by all patients - intervention


tot.dqalys <<- 0 # difference in QALYs between intervention and comparator

tot.dcosts <<- 0 # difference in costs between intervention and comparator

# Outer loop, repeat for each patient

for (i in 1:npats){

if (ind==T){

  this.PatData <<- list(
int=list(
    nvert=0,
    nhip=0,
    deathhip=0,
    deathother=0
),

noint=list(
    nvert=0,
    nhip=0,
    deathhip=0,
    deathother=0
)

# Debugging line
if (dbg==T){
    cat(paste("#####\n[", i, "]\n#####\n"))  # The cat function prints text to console
print("No Intervention Patient")
}

# Generate event list - no intervention

evtlist.noint <<- InitEventList.noint()

# Generate event list - intervention

evtlist.int <<- InitEventList.int(evtlist.noint)

# For the no intervention patient:
# current time, set as global variable to 0

curtime <<- 0

utilmlt <<- 0.7 # initially, the patient has a utility of 0.7

prvvert <<- F # no previous vfract

# QALYs and costs for this patient

thsqalys <<- 0

thscosts <<- 0
while(curtime < Inf){
    # Get next event, process, repeat
    Evt <- GetNxtEvt(intervention=F)

    if (is.null(Evt)==F){
        ReactEvt(Evt, intervention=F)
    } else {curtime <<- Inf}

    if(dbg==T){
        print(paste("No Intervention, Qalys:", round(thsqalys, 2), "; cost:", round(thscosts,0) ))
    }

    if (ind==T){
        this.PatData$noint$thsqalys <<- thsqalys
        this.PatData$noint$thscosts <<- thscosts
    }
}
}

tot.qalys.noint <<- tot.qalys.noint + thsqalys
tot.costs.noint <- tot.costs.noint + thscosts

# subtracting costs and qalys as from no intervention arm, and want to know int - noint
tot.dqalys <- tot.dqalys - thsqalys
tot.dcosts <- tot.dcosts - thscosts

# for the intervention patient
if (dbg==T){ print("Intervention Patient")}

# reset curtime
curtime <- 0
utilmlt <<- 0.7 # initially, the patient has a utility of 0.7
prvvert <<- F # no previous vfrac

# QALYs and costs for this patient
thsqalys <<- 0
thscosts <<- 0

# if the event list has been emptied
emptylist <<- F

while(curtime < Inf){
Evt <- GetNxtEvt(intervention=T)

if (is.null(Evt)==F){
    ReactEvt(Evt, intervention=T)
} else {curtime <- Inf}

if(dbg==T){
    print(paste("Intervention, Qalys: ", round(thsqalys, 2), "; cost: ", round(thscosts,0) ))
}

if (ind==T){
    this.PatData$int$thsqalys <- thsqalys
    this.PatData$int$thscosts <- thscosts
}

if (ind==T){
    PatData[[i]] <- this.PatData
tot.qalys.int <<- tot.qalys.int + thsqalys
tot.costs.int <<- tot.costs.int + thscosts

# adding costs and qalys as from intervention arm, and no int has already been subtracted
tot.dqalys <<- tot.dqalys + thsqalys
tot.dcosts <<- tot.dcosts + thscosts