```
## Code for data simulation and analysis

## Load Required Packages

library(Epi)
library(gam)
library(multipleNCC)
library(peperr)
library(plyr)
library(stats)
library(survival)
library(lattice)
library(etm)
library(geepack)




SimCRData <- function(n, covn, alpha01, alpha02, cov01, cov02, uppercens) {

  ## Simulates competing risks data for 2 competing events.
  ##
  ## Args:
  ##  n : size of cohort
  ##  covn : number of exposed individuals
  ##  alpha01 : event 1 cause-specific hazard for unexposed
  ##  alpha02 : event 2 cause-specific hazard for unexposed
  ##  cov01 : event 1 hazard ratio exposed
  ##  cov02 : event 2 hazard ratio exposed
  ##  uppercens : upper limit of uniform distribution for censoring
  ##
  ## Returns:
  ##   data frame of competing risks data


  # creates list object
  data <- list()

  # entry time 0 for all individuals (no left-truncation)
  entry  <- rep(0,n)

  # event times for non-exposed individuals using exponential
  # distribution and all-cause hazard (sum of unexposed cause-specific
  # hazards)
  exit1 <- rexp((n-covn), alpha01 +alpha02)

  # assigns event type to each event time using multinomial distribution
  cause1 <- rmultinom (n - covn, size = 1, prob = c(alpha01, alpha02))

  # consolidates event types into one vector
  cause1[cause1!=0] <- row(cause1)[cause1!=0]
  cause1<- t(cause1[cause1!=0])

  # covariate vector for unexposed
  cov1 <- rep(0,(n-covn))

  # event time for exposed individuals using exponential distribution
  # and all-cause hazard (sum of exposed cause-specific hazards)
  exit2 <- rexp((covn), (cov01 * alpha01) + (cov02 * alpha02))

  # assign event-type to each event time using multinomial distribution
  cause2 <- rmultinom (covn, size = 1, prob = c((cov01 * alpha01),
                                                (cov02 * alpha02)))

  # consolidate event types into one vector
  cause2[cause2!=0] <- row(cause2)[cause2!=0]
  cause2<- t(cause2[cause2!=0])
```

```
  # covariate vector for exposed
  cov2 <- rep(1,(covn))

  # create individual id
  adm_id <- seq_len(n)

  # covariate vector for full cohort
  cov <-c(cov1,cov2)

  # event type for full cohort
  cause  <-c(cause1,cause2)

  # event time for full cohort
  exit <- c(exit1,exit2)

  # create censoring times
  potential_cens <- runif(n ,0, uppercens)

  # create stop times: minimum of event time and censoring time
  stop <- pmin(exit, potential_cens)

  # create indicator of event vs. censoring
  fail <- c(exit <= potential_cens) * cause

  # consolidate into data frame
  data <- data.frame(adm_id, cov, fail, entry, stop, potential_cens)

  # return data frame
  data

}


CreateSamplestat <- function(nestinfo, data, samplestat, id, set) {

  ## Adjusts samplestat vector for use in multipleNCC package.
  ## Sampled controls are given "1."
  ##
  ##
  ## Args:
  ##   nestinfo:    data frame of information for reduced cohort after
  ##                IDS sampling
  ##   joininfo:    full cohort data frame before sampling
  ##   samplestat:  vector of sampling and status (case or control) info
  ##                before adjustment
  ##   id:          id for joining data frames
  ##   set:         time-matched case-control set
  ##
  ## Returns:
  ##   data frame for full cohort with adjusted samplestat vector


  ## marks patient as sampled
  nestinfo[, samplestat] <- replace (nestinfo[, samplestat],
                       nestinfo[[samplestat]] == 0, 1)


  # gives columns consistent names for join function
  ncc1 <- data.frame( nestinfo[[id]],  nestinfo[[samplestat]],  nestinfo[[set]])
  colnames(ncc1) <- c(id, samplestat, set)

  # removes duplicate event times
  ncc1 <- unique(ncc1[!duplicated(ncc1[-3]),] )

  # merges new samplestat vector to original data frame
  ncc1 <- join(ncc1, data, by = id, type = "full")

  # returns data frame
  ncc1
```

```
}

## Simulate competing risk data
set.seed(1)
Test_Data <- SimCRData(10000, 5000, .02, 0.5, 2, 0.5, 1000000)

## Column for Case-Control Set
Test_Data$Set <- c(rep(0,10000))

## Add noise to "stop" times to break ties
set.seed(1)
Test_Data$stop <- jitter(Test_Data$stop,
                         .000001)

## Creates samplestat vector
## Assign '2' for event 1 individuals
## (these will be given inverse probability weight = 1 in analysis)
temp <- replace(Test_Data$fail, Test_Data$fail== 2, 0)
temp <- replace(temp, temp == 1, 2)
Test_Data$samplestat1 <- temp
rm(temp)


## FULL COHORT ANALYSIS

## Event 1
Test_Data_ncc_IDS_Full1 <- coxph(Surv(stop, fail== 1) ~ cov,
                                 data = Test_Data, ties ='breslow')


## Event 2
Test_Data_ncc_IDS_Full2 <- coxph(Surv(stop, fail== 2) ~ cov,
                                data = Test_Data, ties ='breslow')

## Event 1 Risk Ratio

# Censor competing events
Test_Data$Event_1 <-ifelse(Test_Data[, "fail"]==1, 1, 0)

# Log Binomial Model
Test_Data_Sublite_NI_Full <- glm(Event_1 ~ cov,
                                 family = binomial(link = "log"), data= Test_Data)


## SAMPLING

## Performs Incidence Density Sampling. 1 control per 1 event
set.seed(1)
Test_ncc <- ccwc(entry = 0,
                 exit = stop,
                 fail = fail == 1,
                 origin=0,
                 controls = 1,
                 data = Test_Data,
                 include = list(adm_id, stop, cov, samplestat1))


# Adjusts samplestat vector. Sampled controls are given value '1'
Test_Data <- CreateSamplestat(Test_ncc, Test_Data, "samplestat1", "adm_id", "Set")


## TRADITIONAL ANALYSIS NCC DESIGN (CONDITIONAL LOGISTIC REGRESSION: Event 1)

Test_Data_ncc_IDS_Clog <- clogit(Fail ~ cov + strata(Set),  data = Test_ncc)


## INVERSE PROBABILITY WEIGHTING (IPW) ANALYSIS: Event 1
```

```
## Step 1
## Calculate Inclusion Probabilities

# Kaplan-Meier Weights
Test_Data$KM_weights1 <- KMprob(Test_Data[["stop"]],
                                Test_Data["samplestat1"], m = 1)

# Logistic Regression Weights
Test_Data$GLM_weights1 <-GLMprob(Test_Data[["stop"]],
                                 Test_Data[["samplestat1"]])

# remove non-cases, non-sampled indviduals
Test_Data_subcohort <- subset(Test_Data, Test_Data[,"samplestat1"] != 0)


## Step 2
## Weighted Cox Model

# Kaplan-Meier inclusion weights
Test_Data_ncc_IDS_KM1 <- coxph(Surv(stop, fail == 1)
                               ~ cov + cluster(Set),
                               data = Test_Data_subcohort, weights = (1/(KM_weights1)),
                               ties ='breslow')

# Logistic Regression Weights
Test_Data_ncc_IDS_GLM1 <- coxph(Surv(stop, fail == 1)
                              ~ cov + + cluster(Set),
                              data = Test_Data_subcohort, weights = (1/(GLM_weights1)),
                              ties ='breslow')


## INVERSE PROBABILITY WEIGHTING (IPW) ANALYSIS: Event 2


## Step 1
## Calculate Inclusion Probabilities

## Same as above

## Step 2
## Weighted Cox Model

# Kaplan-Meier inclusion weights
Test_Data_ncc_IDS_KM2 <- coxph(Surv(stop, fail == 2)
                              ~ cov + cluster(Set),
                              data = Test_Data_subcohort, weights = (1/(KM_weights1)),
                              ties ='breslow')


# Logistic Regression Weights
Test_Data_ncc_IDS_GLM2 <- coxph(Surv(stop, fail == 2)
                              ~ cov + cluster(Set),
                              data = Test_Data_subcohort, weights = (1/(GLM_weights1)),
                              ties ='breslow')


## INVERSE PROBABILITY WEIGHTING (IPW) RISK RATIO ANALYSIS: Event 1

## Step 1
## Calculate Inclusion Probabilities

## Same as above


## Step 2
## Weighted Log Binomial Model

# Kaplan-Meier Inclusion Weights
Test_Data_Sublite_NI_Subcohort_KM <- geeglm(Event_1 ~ cov,
```

```
                                          family = binomial(link="log"),
                                         data= Test_Data_subcohort,
                                         weights = (1/(KM_weights1) ), id= Set)

# Logistic Regression Weights
Test_Data_Sublite_NI_Subcohort_GLM <- geeglm(Event_1 ~ cov,
                                         family = binomial(link = "log"),
                                         data= Test_Data_subcohort,
                                         weights = (1/(GLM_weights1) ), id= Set)
```