Cost-effectiveness Analysis of Pooled Testing

Fan Zhong (Frank)

Key Codes

This project is aimed to conduct a cost-effectiveness analysis of pooled testing based on simulated data. The pooled testing protocol can be used in various fields, including usages in pandemics, quality checks in manufacturing areas, etc.

Data Simulation

The data simulation process uses packages `tidyverse`, `simstudy` and `Rlab`. We begin the simulation with entry of values required. The `SimRep` stands for number of simulations, and this is highly dependent on the random-access memory of your computer or laptop. It may take tens of hours or even days to run all the simulation, so be careful when setting this value and it is recommended to set this value small at first. The OR values (or odds ratio) represent the odds that an outcome will occur given a particular exposure, compared to the odds of the outcome occurring in the absence of that exposure. For example, the `SexOR` describes the risk of males being infected compared to females. For reference, the values below were used in this simulation (correct to 2 decimal places for non-exact values).

```
SimRep          <- XXX # No. Simulation
familyno        <- XXX # No. Fammily within each simulation
intercpt        <- XXX # Logit regression intercept, determining
                       expected infection rate
AgeGrpOR        <- XXX # OR value for age group (over or below
                       70 years)
SexOR           <- XXX # OR value for male
ComorbidityOR   <- XXX # OR value for chronic comorbidity
VaccinationOR   <- XXX # OR value for vaccination
oddsratio       <- XXX # OR value for secondary infection risk
                       ratio
```

Then we create a data frame with 3 variables: the simulation number `SimRep`, family number `familyno` and row number `row_number`, for later storage of simulated data.

```
rowtt <- SimRep*familyno
df    <- data.frame(SimRepNo = rep(1:SimRep, each = familyno),
                    FamilyNo = c(1:familyno))
df    <- df %>% mutate(No = row_number())
```

Firstly, we simulate the family sizes and input the variables `FamilyNo` and `FamilySize` into the data frame. The family sizes were assumed to be following a non-zero Poisson distribution with $\lambda = 2.4$.

```
gen.family <- defData(varname = "FamilySize", dist = "noZeroPoisson",
                      formula = 2.4, id = "No")
set.seed(4321)
dtfamily   <- genData(rowtt, gen.family)
m1         <- merge(df, dtfamily, all.x = TRUE, by = "No")
```

The `FamMemNo` variable will then be added into the data frame, so that each subject in the simulation can be located by the combination of `FamilyNo` and `FamMemNo`. At the same time, we rearrange the order of `FamMemNo` and perform data cleaning.

```
FamilySize_vector <- m1$FamilySize
repeat_times      <- ifelse(m1$FamilySize %in% FamilySize_vector,
                            FamilySize_vector, 0)
m2                <- m1[rep(FamMemNo <- seq_len(nrow(m1)),
                            repeat_times), ]
m2                <- m2 %>% mutate(No2 = row_number())
m2                <- m2 %>% select(No2, No, SimRepNo, FamilyNo,
                                   FamilySize)
m2                <- m2 %>%
                     arrange  (SimRepNo, FamilyNo) %>%
                     group_by (SimRepNo, FamilyNo,) %>%
                     mutate   (FamMemNo = rank(FamilySize,
                               ties.method = 'first'))
```

As we can see, the data frame now has 791,800 rows, each corresponding to a distinct subject, 300 sets of simulation and 1,000 families within each simulation.

```
tail(m2)
```

```
## # A tibble: 6 × 6
## # Groups:   SimRepNo, FamilyNo [3]
##      No2     No SimRepNo FamilyNo FamilySize FamMemNo
##    <int>  <int>    <int>    <int>      <dbl>    <int>
## 1 791795 299998      300      998          1        1
## 2 791796 299999      300      999          2        1
## 3 791797 299999      300      999          2        2
## 4 791798 300000      300     1000          3        1
## 5 791799 300000      300     1000          3        2
## 6 791800 300000      300     1000          3        3
```

Secondly, we simulate the data for age group `AgeGrp`, sex `Sex`, chronic comorbidity `Comorbidity` and vaccination status `Vaccination`. All the four variables were assumed to follow binomial distribution (age group used for age simulation).

```
set.seed(1)
m2$AgeGrp      <- rbinom(nrow(m2), 1, XXX)
m2$Sex         <- rbinom(nrow(m2), 1, XXX)
m2$Comorbidity <- rbinom(nrow(m2), 1, XXX)
m2$Vaccination <- rbinom(nrow(m2), 1, XXX)
```

We assume that the infection statuses are affected by the age group `AgeGrp`, sex `Sex`, chronic comorbidity `Comorbidity` and vaccination status `Vaccination` of the individuals. In addition, considering the characteristics of infectious diseases, we simulated the secondary infection within families. The logistic regression (a statistical model that models the log-odds of an event as a linear combination of one or more independent variables, https://en.wikipedia.org/wiki/Logistic_regression) was used for simulating the infection statuses within each family. For simulation of the first infection status within families, four OR values (for age group, sex, chronic comorbidity and vaccination status) were used.

```
m2$logit <- intercpt + AgeGrpOR*m2$AgeGrp + SexOR*m2$Sex +
            ComorbidityOR*m2$Comorbidity +
            VaccinationOR*m2$Vaccination
m2$p     <- exp(m2$logit)/(1+exp(m2$logit))
m2$runi  <- runif(nrow(m2))
m2       <- m2 %>% mutate(infection = if_else(runi < p, 1, 0))

m2  <- subset(m2, select = -c(No, No2, logit, p, runi))
m21 <- m2[m2$FamMemNo == 1, ]
m21 <- rename(m21, Infection_RO = infection)
m21 <- subset(m21, select = -c(AgeGrp, Sex, Comorbidity, Vaccination))
m21 <- subset(m21, select = -c(FamMemNo))
```

Then we use the same method to simulate the secondary infections within families, with an additional β for the secondary infection risk ratio.

```
mm1       <- m2 %>% left_join(m21, by = c("SimRepNo", "FamilyNo"))
mm1$logit <- intercpt + AgeGrpOR*mm1$AgeGrp + SexOR*mm1$Sex +
             ComorbidityOR*mm1$Comorbidity +
             VaccinationOR*mm1$Vaccination + oddsratio*mm1$Infection_RO
mm1$p     <- exp(mm1$logit)/(1+exp(mm1$logit))
mm1$runi  <- runif(nrow(mm1))
mm1       <- mm1 %>% mutate(infection = if_else(runi < p, 1, 0))

mm1       <- mm1 %>% mutate(infection = if_else(FamMemNo == 1,
             Infection_RO, infection))
mm1       <- subset(mm1, select = -c(logit, p, runi))
```

To ensure the simulation stays in line with our presumed infection rate, we have to conduct a double check on the infection rate within the simulated data, especially for `Infection_R0` (`Infection_R0` must be smaller than `Infection`).

```
summary(mm1[["infection"]])
summary(mm1[["Infection_RO"]])
```

The following codes are for data cleaning of the simulation process.

```
mm1     <- subset(mm1, select = -c(FamilySize.y, Infection_RO))
dsfinal <- rename(mm1, FamilySize = FamilySize.x)
rm(list=ls()[!ls() %in% "dsfinal"])
```

Assigning Tube ID

For the assignment of tube ID, we need the following packages:

```
library(proto)
library(gsubfn)
library(RSQLite)
library(sqldf)
```

The assumed three assigning rules are as follows:

Rule 1 (for the first family):

If $S_1 \leq m$, then all members will be pooled in the first tube ($t_1 = 1$ in this case). Otherwise ($S_1 > m$), the first m members will be in the first tube and the rest will be in the second tube (i.e. $t_1 = 2$).

If $S_n > m$, then the first m members in the nth family will be pooled in one single tube, and the rest will continue pooling with the $(n+1)^{th}$ family.

Rule 2 (for the second family):

If $S_1 + S_2 \leq m$ or $S_1 > m$, $S_1 + S_2 \leq 2m$, then all members in the second family will be pooled in tube t_1 together with the first family (i.e. $t_1 = t_2 = 1$).

If $S_1 \leq m$, $S_1 + S_2 > m$ or $S_1 > m$, $S_1 + S_2 > 2m$, then the second family start with a new tube (following Rule 1) (i.e. If $S_2 \leq m$, then all members will be in tube $t_1 + 1[= t_2]$; Otherwise, the first m members will be in tube $t_1 + 1$ and the rest in tube $t_1 + 2[= t_2]$).

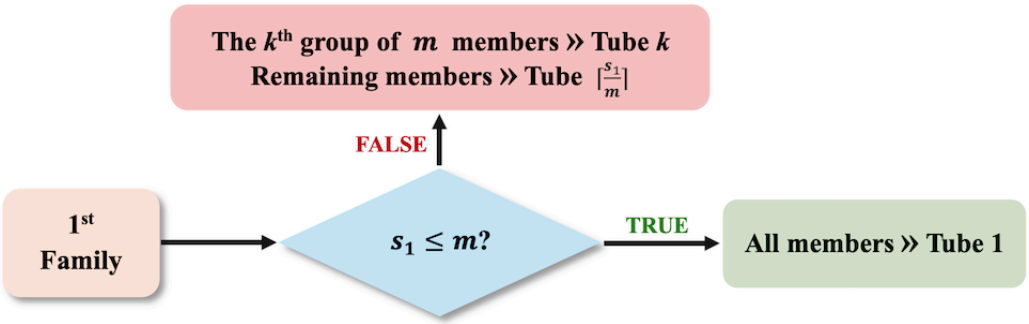
Rule 3 (for the n^{th} family, $n \geq 3$):

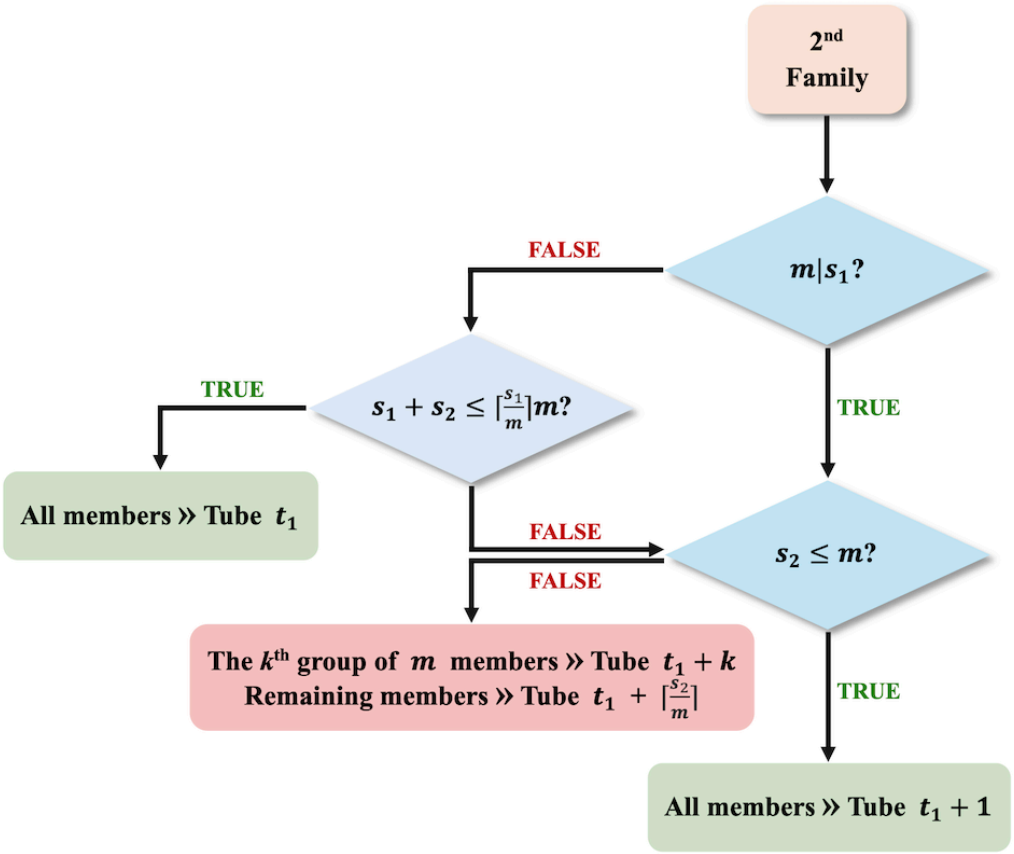
The n^{th} family only look for possible vacancies in the $(n-2)^{th}$ and the $(n-1)^{th}$ families, and vacancies in the $(n-2)^{th}$ family will be considered before those in the $(n-1)^{th}$ family.

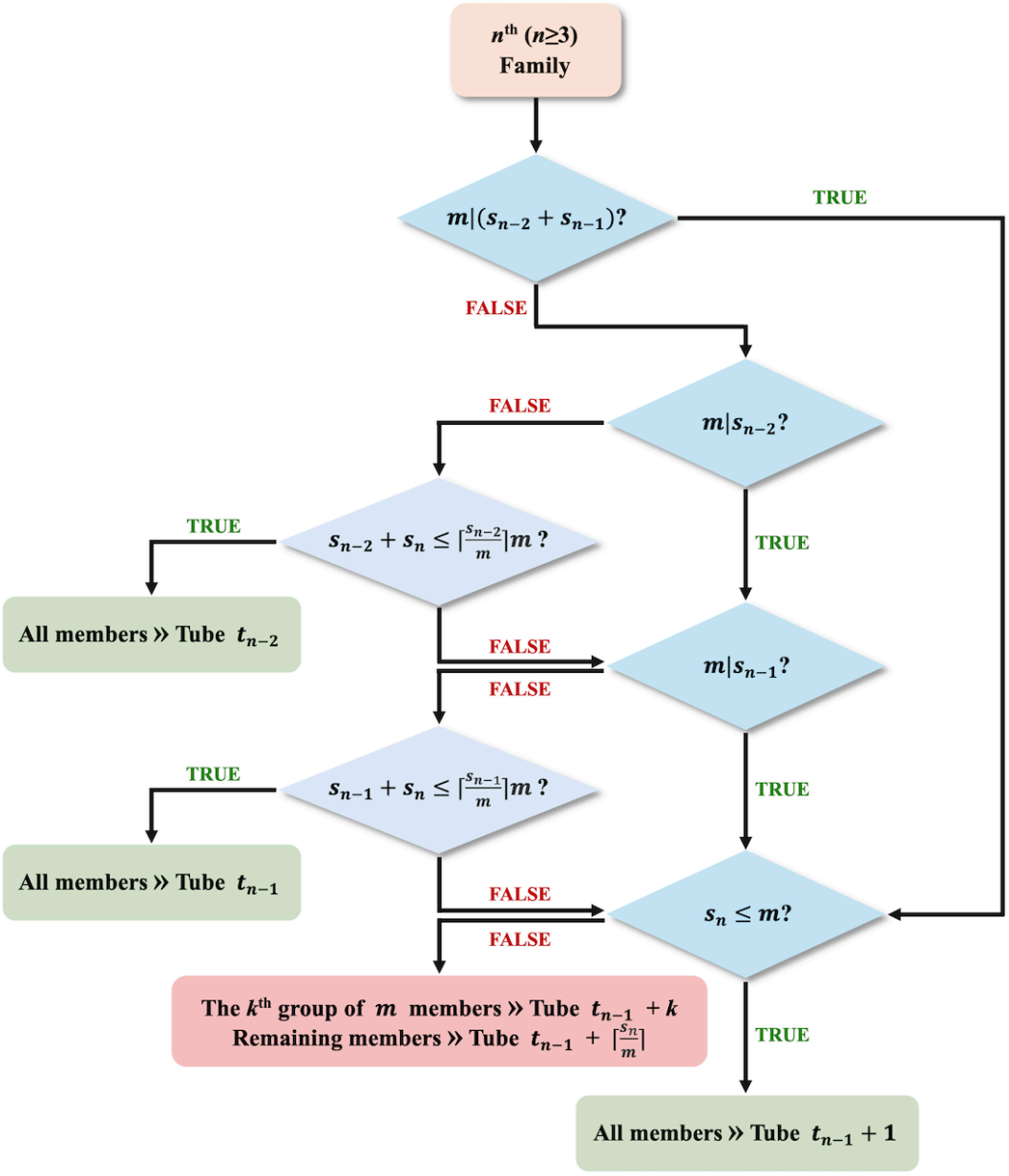
If neither of the $(n-2)^{th}$ and the $(n-1)^{th}$ families provides enough vacancies for the n^{th} family, then the n^{th} family will start with a new tube (following Rule 1).

Where: m denotes the pooling size (i.e. the tubes have a maximum storage capacity of m swabs of each single tube). S_n denotes the family size of the n^{th} family. t_n denotes the number of tubes used after the n^{th} family is pooled.

The rules are also demonstrated in the flow charts as follows:


1st Family
$s_1 \leq m$?
TRUE
All members » Tube 1
FALSE
The k^{th} group of m members » Tube k
Remaining members » Tube $\lceil \frac{s_1}{m} \rceil$


2nd Family
$m|s_1$?
FALSE
TRUE
$s_1 + s_2 \leq \lceil \frac{s_1}{m} \rceil m$?
TRUE
All members » Tube t_1
FALSE
FALSE
$s_2 \leq m$?
TRUE
The k^{th} group of m members » Tube $t_1 + k$
Remaining members » Tube $t_1 + \lceil \frac{s_2}{m} \rceil$
All members » Tube $t_1 + 1$



The function `AssignTube` is set to assign tubes to each resident, strictly obeying the rules mentioned above.

```
AssignTube <- function(dataset, capacity){

  dataset   = dataset %>% arrange(FamilyNo, FamMemNo)
  familymax = max(dataset$FamilyNo)
  tubeid01  = sqldf('select FamilyNo,count(*) as memsum from dataset
                     group by FamilyNo')
  tubeid02  = dataset %>% mutate(tubeid = if_else(FamilyNo==1,
                                 ceiling(FamMemNo/capacity), NA))
```

```
tubeid0   = tubeid01 %>% left_join(tubeid02, by = 'FamilyNo')

tubeid11  = tubeid0 %>% filter(tubeid==max(tubeid0$tubeid
                               [tubeid0$FamilyNo==1]))
tubeid12  = sqldf('select FamilyNo, tubeid, count(tubeid) as tuberes
                   from tubeid11')
tubeid1   = tubeid12 %>% mutate(tuberes = capacity-tuberes)
tubeid0   = tubeid0 %>% mutate(capacity = capacity)
tubeid0   = sqldf('select a.FamilyNo, a.FamMemNo, a.memsum,
                   case when a.FamilyNo = 2 and a.memsum<=b.tuberes
                   then b.tubeid
                   when a.FamilyNo = 2
                   then b.tubeid + ceil(a.FamMemNo/capacity)
                   else a.tubeid end as tubeid
                   from tubeid0 a
                   left join tubeid1 b on a.FamilyNo = 2
                   order by a.FamilyNo, a.FamMemNo;')

for(i in 3:familymax){
  tubeid11 = tubeid0 %>% filter(tubeid==max(tubeid0$tubeid
                                            [tubeid0$FamilyNo==i-2]))
  tubeid12 = sqldf('select FamilyNo, tubeid, count(tubeid) as tuberes
                    from tubeid11')
  tubeid1  = tubeid12 %>% mutate(tuberes = capacity-tuberes)

  tubeid21 = tubeid0 %>% filter(tubeid==max(tubeid0$tubeid
                                            [tubeid0$FamilyNo==i-1]))
  tubeid22 = sqldf('select FamilyNo, tubeid, count(tubeid) as tuberes
                    from tubeid21')
  tubeid2  = tubeid22 %>% mutate(tuberes = capacity-tuberes)

  tubeidl  = max(tubeid0$tubeid[!is.na(tubeid0$tubeid)])

  tubeid0  = tubeid0 %>% mutate(i = i, capacity = capacity,
                                tubeidl = tubeidl)

  tubeid0  = sqldf('select a.FamilyNo, a.FamMemNo, a.memsum,
                    case when a.FamilyNo = i and a.memsum<=b.tuberes
                    then b.tubeid
                    when a.FamilyNo = i and a.memsum<=c.tuberes
                    then c.tubeid
                    when a.FamilyNo = i
                    then tubeidl + ceil(a.FamMemNo/capacity)
                    else a.tubeid end as tubeid
                    from tubeid0 a
                    left join tubeid1 b on a.FamilyNo = i
                    left join tubeid2 c on a.FamilyNo = i
                    order by a.FamilyNo,a.FamMemNo;')
}
```

```
  output = tubeid0 %>% select(FamilyNo, FamMemNo, tubeid)
  return(output)
}
```

Then we assign the tube IDs to the residents, be aware that this process may take a long time as the number of sets of simulations `SimRemax` gets larger.

```
Add_Tubeid   <- data.frame(SimRepNo = numeric(0),
                           FamilyNo = numeric(0),
                           FamMemNo = numeric(0),
                           tubeid = numeric(0))
SimRemax     <- max(dsfinal$SimRepNo, na.rm = FALSE)

for(ii in 1:SimRemax){
  family     <- subset(dsfinal, SimRepNo==ii)
  tubeid0    <- AssignTube(dataset = family, capacity = 5)
  Add_Tubeid <- rbind(Add_Tubeid, tubeid0 %>% mutate(SimRepNo = ii))
  gc()
}
```

Calculating number of tests needed

From the tube-assignment results, we calculate the number of tests, for further cost-effectiveness analysis.

The number of first-time testings are given by:

```
notube        <- sqldf('select SimRepNo, max(tubeid) as FirstTubeid
                        from Add_Tubeid group by SimRepNo')
tubecompare   <- sqldf('select A.SimRepNo, A.NoTubeid, B.FirstTubeid
                        from notube0 A
                        full join notube B on A.SimRepNo = B.SimRepNo')
```

And similarly for the second-time testings:

```
Add_Tubeid_IF  <- sqldf('select a.SimRepNo, a.FamilyNo, a.FamMemNo,
                         a.tubeid, b.infection from Add_Tubeid a
                         left join dsfinal b
                         on a.SimRepNo = b.SimRepNo
                         and a.FamilyNo = b.FamilyNo
                         and a.FamMemNo = b.FamMemNo
                         order by a.SimRepNo, a.FamilyNo, a.FamMemNo;')
 secondtube    <- sqldf('select SimRepNo, infection, Tubeid,
                         max(Tubeid) AS Secondtube from Add_Tubeid_IF
                         group by SimRepNo, Tubeid having
                         sum(infection)>0
                         order by SimRepNo, Tubeid;')
 tubecompare2  <- sqldf('select SimRepNo, count(SimRepNo) AS SecTubeid
                         from secondtube group by SimRepNo;')
```

Finally, we merge the two number of testings required.

```
Forcost       <- sqldf('select a.SimRepNo, a.NoTubeid, a.FirstTubeid,
                        b.SecTubeid from tubecompare a
                        full join tubecompare2 b
                        on a.SimRepNo = b.SimRepNo;')
Forcost$SecTubeid[Forcost$FirstTubeid>0 & is.na(Forcost$SecTubeid)]<-0
```

Calculating cost of testings

For the output of our cost-effectiveness analysis, we calculate the cost for each testing protocol under various empirical infection rates.

The bootstrap method is used to calculated the mean and confidence interval of the costs.

```
library(FertBoot)

First_Test_Cost       <- XXX # Cost of first time testing
Sec_Test_Cost         <- XXX # Cost of second time testing
PCR_service_Cost      <- XXX # Cost of testing service fee

df_DsXXX              <- Forcost
df_DsXXX              <- (df_DsXXX %>% mutate(
                          Cost = First_Test_Cost*FirstTubeid +
                                 Sec_Test_Cost*SecTubeid +
                                 PCR_service_Cost))
df_DsXXXcostperhead   <- (df_DsXXX %>% mutate(
                          Costperhead = Cost/NoTubeid))

vec_DsXXXabsolutecost <- df_DsXXXcostperhead$Costperhead
mean(vec_DsXXXabsolutecost)
boot.CI(vec_DsXXXabsolutecost, alpha = 0.05, CI.type = "per")
```