# Simulations under the alternative hypothesis: power and ranking

2023-05-01

## Contents

## 1 Introduction

- This file documents the commands in the `simalt.R` script that simulates data under the alternative hypothesis for the power/ranking part of the simulation study.
- Simulations are done on the Compute Canada cluster.
  - `simalt.sh` and `simalt.R` are the slurm and R scripts.
- These simulations require the pedigree data files (generated by `simrvped.R`, and named `ascertained_pedi.txt`, for `i=1,...,150`), the plain-text file of IDs for the pool checked pedigrees to sample study pedigrees from (produced by `checkpeds.R` and named `pedpool.txt`) and the .RData file of chromosome 8 sequences to sample founder sequences from (generated by `getseqscrvs.R`, and named `chr8.RData`).
  - All of the above data files are to be stored on Compute Canada in the `/project/def-jgraham/FJdata` directory. The following code chunk sets this directory as the path for the input and output files. Modify the `infileDir` and `outfileDir` variables below if you decide to read and/or write from/to different directories.

```
infileDir <- "/project/def-jgraham/FJdata"
outfileDir <- "/project/def-jgraham/FJdata"
```

- The simulation of 2000 replicate studies is done in batches of N=10, and run as an "array job" on the cluster, meaning that the same script will run 200 times, each with a unique "job ID".
  - The SLURM script to run the array job is `simalt.sh`. This runs the R script `simalt.R` that is generated by this .Rmd file.
- Each incarnation of the R script can access the SLURM job ID through a Unix environment variable called `SLURM_ARRAY_TASK_ID`, which is read into R as follows.

```
dID = as.numeric(Sys.getenv("SLURM_ARRAY_TASK_ID"))
if(is.na(dID)) {
  dID=1
  warning("No task ID, setting task ID to 1")
}
seed <- dID
set.seed(seed)
```

# 2 Workflow

## 2.1 Prepare for simulations

### 2.1.1 Load R packages and functions

```
library(SimRVSequences)
library(RVMethods)
library(Matrix)
# Load the cd_new() function from its R source file
source("cd_new.R")
```

### 2.1.2 Read in pedigrees and sequences to sample from

- Read in
  - the IDs (between 1 and 150) of the 55 checked pedigrees (output by the `checkpeds.R` script) to sample studies from
  - the population of chromosome 8 sequences (output by the `getseqscrvs.R` script) that we will sample pedigree founder sequences from

```
pedpool <- scan(paste0(infileDir,"/pedpool/pedpool.txt"))
load(paste0(infileDir,"/chr8.RData"))
```

### 2.1.3 Set parameters for the simulation study

- There are three pedigrees per study and we run simulations in batches of 10 studies.

```
studysize <- 3 # number  of pedigrees in a study
N <- 10   # number of studies per batch of simulations
```

- We also create the vector of carrier probabilities to consider for global likelihood ratio statistics and global configuration probabilities under the null hypothesis (used in p-values for the global approaches).
  - The true value of the carrier probability used to simulate the data is 0.00032. We will consider this value plus misspecified values of 1/10, 1/2, 2 times and 10 times the true value.

```
true_carrier_prob <- 0.00032
carrier_probs <- true_carrier_prob*c(1/10,1/2,1,2,10)
```

### 2.1.4 Create output files to which to write the results.

- We want to avoid scientific notation in our output because R reads it as character values which get coerced to numeric as `NA`s. We use `cat()` to write results to output files. Large values of `cat()`'s printing option 'scipen" (e.g., 999) prevent scientific notation.

```
options(scipen=999)
```

- **Note:** The following detailed description of the output files can be skipped on first reading. However, you will need to come back to this description when you work through the `simaltSummary.Rmd` script that reads in and summarizes the simulation output.

- In the next code chunk we write the header (column names) to the comma-delimited output files that will hold p-values and rankings. The columns of these files are described below.
- For each simulated study and each cRV within a study, we record the p-values and ranks of the 5 methods.
- Here we create comma-delimited files to hold the p-values and ranks, and write column names to each file.
- In both the p-value and ranking output files we include the following columns of information about the study: its replicate number, the IDs of the three study pedigrees, and the IDs of the cRVs sampled in each pedigree. (IDs can be duplicated if the same cRV is sampled in more than one family.)
- For $n_p$ carrier probabilities, up to $3 \times (2 \times n_p + 3)$ p-values are recorded. Specifically, for each of up to three cRVs, we have $n_p$ p-values for the global LR, $n_p$ p-values for the transmission approach, and one p-value for each of the three local approaches (local LR, RVS and modified RVS).
  - For the $i^{th}$ carrier probability and $j^{th}$ cRV, the column names for the global likelihood ratio test are `globalLR`$i$`_`$j$. For example, the column name for the global LR using the first carrier probability and the first cRV is `globalLR1_1`, and the column name for the global LR using the fifth carrier probability and third cRV is `globalLR5_3`.
  - Analogously, for the $i^{th}$ carrier probability and $j^{th}$ cRV the column names for the global transmission test are `globaltrans`$i$`_`$j$.
  - For the local tests (that do not depend on the carrier probability) the column names for the $j^{th}$ cRV are `localLR_`$j$, `RVS_`$j$ and `modRVS_`$j$, respectively.
  - If fewer than three cRVs are in the study, empty slots for p-values are encoded as `NA`.
- For $n_p$ carrier probabilities, up to $3 \times (n_p + 4)$ rankings are recorded. Specifically, for each of up to three cRVs we have $n_p$ rankings for the global LR and one ranking each for the transmission, local LR, RVS and modified RVS approaches. We also record the number of chromosome 8 RVs that were observed in the study (the number of RVs the cRVs were ranked against).
  - The column to hold the number of RVs per study is `numRV`.
  - For the $i^{th}$ carrier probability and $j^{th}$ cRV, the column names for the rankings are `globalLR`$i$`_`$j$ for the global likelihood ratio statistic. The transmission statistic and local statistics do not depend on the carrier probability. For the transmission statistic and local statistics, the column names for the $j^{th}$ cRV are `globaltrans_`$j$, `localLR_`$j$, `RVS_`$j$ and `modRVS_`$j$, respectively.
  - If fewer than three cRVs are in the study, empty slots for rankings are encoded as `NA`

```r
pfile <- paste0(outfileDir,"/pvalres",dID,".csv") # file for power results
repcols <-c("rep","studyped_1","studyped_2","studyped_3","cRV_1","cRV_2","cRV_3")
pvalcols <- c(paste0("globalLR",1:length(carrier_probs),"_1"),
               paste0("globaltrans",1:length(carrier_probs),"_1"),
               "localLR_1","RVS_1","modRVS_1",
               paste0("globalLR",1:length(carrier_probs),"_2"),
               paste0("globaltrans",1:length(carrier_probs),"_2"),
               "localLR_2","RVS_2","modRVS_2",
               paste0("globalLR",1:length(carrier_probs),"_3"),
               paste0("globaltrans",1:length(carrier_probs),"_3"),
               "localLR_3","RVS_3","modRVS_3")
cat(paste0(c(repcols,pvalcols),collapse=","),"\n",file=pfile) # write header to file

rfile <- paste0(outfileDir,"/rankres",dID,".csv") # file for ranking results
rankcols <- c("numRV",paste0("globalLR",1:length(carrier_probs),"_1"),
               "globaltrans_1",
               "localLR_1","RVS_1","modRVS_1",
               paste0("globalLR",1:length(carrier_probs),"_2"),
               "globaltrans_2",
               "localLR_2","RVS_2","modRVS_2",
               paste0("globalLR",1:length(carrier_probs),"_3"),
               "globaltrans_3",
               "localLR_3","RVS_3","modRVS_3")
cat(paste0(c(repcols,rankcols),collapse=","),"\n",file=rfile) # write header to file
```

## 2.2 Simulation study

- Our goal is to simulate a batch of `N` studies and, for each study, calculate the p-values and ranks of the five methods.

### 2.2.1 Overview of simulation loop

- We repeat the following a small number `N` (e.g. 10) times for a compute node on the cluster:
  1. Sample `studysize` (i.e. 3) pedigrees to make a study.
  2. Sample cRVs and sequence data for the study.
  3. Get look-up tables of p-values and ranks for each possible global configuration of the study.
  4. Use the look-up tables to find the p-value and rank for the observed global configuration of each sampled cRV.
  5. Write the results to output files.

#### 2.2.1.1 Sample pedigrees

- Pedigrees are chosen by sampling without replacement from the pool of 55 eligible pedigrees.
- Each pedigree is read into R as a data-frame from its plain-text file.
- The three pedigree-specific data-frames are then combined into a single large data frame, with a family ID column to distinguish the pedigrees, for Christina's `sim_RVstudy()` function.

#### 2.2.1.2 Sample cRVs and sequence data

- Christina's `sim_RVstudy()` function simulates cRVs and sequence data in the study. `sim_RVstudy()` takes the combined study data-frame as input, but operates on the individual pedigrees separately.
- For each pedigree it (i) samples a cSNV from the pool of cSNVs, (ii) samples founder sequences from Nirodha's American Admixed population, and (iii) conditionally gene drops these sequences through the pedigree.
- It then combines the resulting sequence data into one object to be returned to the user.
- The returned sequences require further filtering to remove RVs because `sim_RVstudy()` keeps not only the RVs observed in the sequenced individuals (i.e. alive, affected and ascertained during the study period) but also those available in any pedigree ancestors that connect them. However, a study will only record RVs that are observed in the sequenced individuals, and so we need to filter RVs accordingly.

#### 2.2.1.3 Get look-up tables of p-values and ranks

- We use the function `cd_new()` to get the lookup tables of test statistics and p-values for the 5 methods. `cd_new()` is documented in `cd_new.Rmd`.

#### 2.2.1.4 Use the look-up tables to find p-values and ranks of the observed configuration

- For each sampled cSNV, we find its global configuration and:
  (i) Look up the p-values for this global configuration in the p-value lookup table.
  (ii) Look up the statistic values for this global configuration in the statistics lookup table and rank them against other global configurations in the observed sequence data.

#### 2.2.1.5 Write the results to files

- Finally, we write the p-values and the ranks to their respective output files.

### 2.2.2 Running a batch of simulated studies with `simaltreps()`

- As a coding note, I initially wrote the R code to implement the simulation study as one big code chunk in this Rmarkdown file, but found the code hard to follow. It's easier to follow if you encapsulate some of the more mundane details into worker functions, but then the Rmarkdown will only knit if all the worker functions are defined *before* they are called, in which case the function definitions lack context.
- To give proper context I wanted a top-down programming style in which the main function is defined first and the worker functions are defined afterwards. The only way I could think to do this was to encapsulate the simulation study into its own function, `simaltreps()`, that is defined first, assuming the worker functions exist and then defining the worker functions. Later in the Rmarkdown file, I call `simaltreps()` after it and all of its worker functions have been defined.
  - To distinguish my worker functions from Christina's functions in `SimRVSequences`, my worker functions are labelled `*worker*` in the comments below and Christina's `SimRVSequences` functions are labelled `*Christina's*`.

```
simaltreps <- function(N,studysize,pfile,rfile,infileDir){
for(simrep in 1:N) {
  cat("simrep",simrep,"\n")
  #------------------------------------------------------------
  #  Sample the IDs of 3 peds from our pool of 55 and then read the
  #  three pedigrees into a data frame
  studypedIDs <- sample(pedpool,size=studysize)
  s_peds <- read_studypeds(studypedIDs,infileDir) # *worker*
  #------------------------------------------------------------
  #  Generate sequence data with sim_RVstudy() and then filter RVs further.
  s_seqs <- sim_RVstudy(s_peds,chr8) # *Christina's*
  a_seqs <- filter2aff(s_seqs) # *worker*
  numRVs <- nrow(a_seqs$SNV_map)
  #------------------------------------------------------------
  #   Call cd_new() to get lookup tables of statistics and p-values for each
  #   possible global configuration of affected individuals in the study.
  #   This is the most computationally-intensive part of the simulation.
  lookupTabs = cd_new(peds = s_peds, subtypes = c("HL", "NHL"),
                     carrier_probs = carrier_probs)
  #   For each familial cRV, find its global configuration and then
  #   the corresponding p-values and ranks from the lookup tables.
  cRVs <- unique(a_seqs$haplo_map$FamCRV) # can be 3, 2 or 1 cRVs in the study
  pvals <- get_pvals(cRVs,lookupTabs,a_seqs) # *worker*
  ranks <- get_ranks(cRVs,lookupTabs,a_seqs) # *worker*
  #   Write the pval and rank results to their files. Also write info
  #   on the rep number, pedigree IDs and the cRVs from each pedigree
  pedcRVs <- get_pedcRVs(a_seqs) # *worker*
  repinfo <- c(simrep,studypedIDs,pedcRVs)
  pvalinfo <- c(pvals[1,],pvals[2,],pvals[3,])
  cat(paste0(paste0(c(repinfo,pvalinfo),collapse=","),"\n"),
      file=pfile,append=TRUE)
  rankinfo <- c(numRVs,ranks[1,],ranks[2,],ranks[3,])
  cat(paste0(paste0(c(repinfo,rankinfo),collapse=","),"\n"),
      file=rfile,append=TRUE)
} # End for-loop over studies.
# Note that simaltreps() can return NULL, because its output is written to files.
```

```
  return(NULL)
} # end simaltreps()
```

### 2.2.3 Utility functions for simaltreps()

```
# read_studypeds() takes a vector of pedigree IDs and reads the
# pedigrees from their plain-text files
read_studypeds <- function(studypedIDs,infileDir){
  for(i in 1:length(studypedIDs)){
    pedfile <- paste0(infileDir,"/ascertained_ped",studypedIDs[i],".txt")
     # Read in pedfile and set the resulting object to be of class "ped" and "data.frame".
    pp <- read.table(pedfile);class(pp) <- c("ped","data.frame")
    # Set FamID column to the ID of our sampled pedigree.
    pp[,"FamID"] <- studypedIDs[i]
    if(i==1) { # Initialize the data frame containing the pedigrees.
      s_peds <- pp
    } else{ # Add new pedigree to the data frame of previous ones.
      s_peds <- rbind(s_peds,pp)
    }
  }
  return(s_peds) #Return the pedigrees.
}
# filter2aff() takes a "famStudy" object returned by sim_RVstudy()
# and reduces its sequence data to rare variants that are seen in
# the affected individuals. It also finds the genotypes for each
# variant/individual. We'll need these genotypes later in the
# ranking application, when we find the RV configurations and their
# corresponding test stats for each variant in the sequence data.
filter2aff <- function(seqs){
  # seqs is a famStudy object, which is a list having elements
  # - ped_haplos (sparse matrix of sequence data),
  # - haplo_map (maps the sequences to individuals), and
  # - SNV_map (support info on each SNV).
  # First reduce to sequences in affected pedigree members with DNA
  ped_haplos <- seqs$ped_haplos[seqs$haplo_map$affected,]
  haplo_map <- seqs$haplo_map[seqs$haplo_map$affected,]
  # Next filter variants to those that appear in the affected individuals with DNA.
  # The variants appear as columns of ped_haplos, and rows of SNV_map
  cc <- colSums(ped_haplos)
  ped_haplos <- ped_haplos[,cc>0]
  SNV_map <- seqs$SNV_map[cc>0,]
  # Lastly, pair sequences from individuals into multilocus genotypes
  odd_inds <- seq(from=1,to=nrow(ped_haplos)-1,by=2)
  even_inds <- seq(from=2,to=nrow(ped_haplos),by=2)
  ped_genos <- ped_haplos[odd_inds,] + ped_haplos[even_inds,]
  return(list(ped_haplos=ped_haplos, haplo_map=haplo_map,
              SNV_map=SNV_map, ped_genos=ped_genos))
}
# find_binID takes the name of an RV and the genotypes of the affected
# individual returned by filter2aff() and returns the compact base-10
# representation of the configuration (what Christina calls the "binID").
find_binID <- function(RV,a_seqs) {
  config_vec <- a_seqs$ped_genos[,a_seqs$SNV_map$marker == RV] # vec of 0's and 1's
  return(config2binID(config_vec))
}
config2binID <- function(config_vec) {
  # Collapse the vector to a string of 0's and 1's and then use
  # strtoi to convert this base-2 number to an integer.
  return(base::strtoi(paste0(config_vec, collapse = ""), base = 2))
}
# get_seqstats takes the observed sequence data on affected individuals
# returned by filter2aff() and the lookup table of statistic values
# for all global configurations and returns statistic values for each
# configuration observed in the sequence data.
get_seqstats <- function(ped_genos,statvals){
  # - ped_genos is the genotypes of the affected individuals at the
  # variants that are observed in the affected individuals.
  # - statvals is the lookup table of statistic values
  binID <- statvals[,"binID"] # binIDs of rows of statvals
  allstats <- matrix(NA,nrow=ncol(ped_genos),ncol=ncol(statvals))
  colnames(allstats) <- colnames(statvals)
  for(i in 1:ncol(ped_genos)) {
    binID.obs <- config2binID(ped_genos[,i]) # *worker*
    if(binID.obs %in% binID) { # then this config is in the lookup table
      allstats[i,1] <- binID.obs
      allstats[i,] <- statvals[binID==binID.obs,]
    }
  } # End loop over the SNVs.
  foundconfig <- !is.na(allstats[,1])
  allstats <- allstats[foundconfig,]
  return(allstats)
}
# get_pedcRVs finds the cRVs of each pedigree in the study
get_pedcRVs <- function(a_seqs){
  # Start by tabulating unique combinations of the family
  # ID and cRV ID. Then extract the cRV IDs.
  FamIDcRV <- unique(a_seqs$haplo_map[,c("FamID","FamCRV")])
  return(FamIDcRV$FamCRV)
}
get_pvals <- function(cRVs,lookupTabs,a_seqs){
  # Create a matrix to hold the p-values, with rows for cRVs and columns
  # for the different methods. In this context, the global LR and global
  # transmission tests with different carrier probs are different "methods".
  # The number of methods is the number of columns of lookupTabs$pvals
  # minus 1 (since the first column of lookupTabs$pvals is binID).
  pvals <- matrix(NA,nrow=3,ncol=(ncol(lookupTabs$pvals)-1))
  for(i in 1:length(cRVs)) {
    # Find binID of the current cRV's configuration.
    binID <- find_binID(cRVs[i],a_seqs) # *worker*
    # Extract pvals for this configuration from the lookup table but
    # remember that the first column of the lookup table is the binID.
    pvals[i,] <- lookupTabs$pvals[lookupTabs$pvals[,"binID"]==binID,-1]
  }
  return(pvals)
}
get_ranks <- function(cRVs,lookupTabs,a_seqs){
  # Create a matrix to hold the ranks, with rows for cRVs and columns for
  # the different methods. In this context, the global LR approach with
  # different values of the carrier probability is taken to be different
  # "methods". The # of methods is the # of columns of lookupTabs$statvals
  # minus 1 (since the 1st column of lookupTabs$statvals is binID).
  ranks <- matrix(NA,nrow=3,ncol=(ncol(lookupTabs$statvals)-1))
  for(i in 1:length(cRVs)) {
    # Find base-10 representation, or "binID", of the configuration.
    binID <- find_binID(cRVs[i],a_seqs)
    # Calculate ranks over the values of the statistics corresponding to
    #the observed configurations in the sequence data on affected individuals.
    obs_stats <- get_seqstats(a_seqs$ped_genos,lookupTabs$statvals) # *worker*
    for(j in 1:ncol(ranks)){
      # Rank current cRV on jth statistic (in (j+1)st col of obs_stats)
      curr_stat <- unique(obs_stats[obs_stats[,"binID"]==binID,j+1])
      ranks[i,j]<-mean(obs_stats[,j+1] >= curr_stat)
    }
  }
  return(ranks)
}
```

### 2.2.4 Call `simaltreps()`

- With all the necessary functions defined, we can now call `simaltreps()` to simulate a batch of `N` studies.

```
simaltreps(N,studysize,pfile,rfile,infileDir)
```

## 2.3 Generate the R script for the cluster.

- Execute the following R command:

```
knitr:: purl(input="simalt.Rmd",output="simalt.R")
```

- Then delete the setup code chunk at the beginning of the script and the `purl()` command at the end of the script.