Simulations under the null hypothesis: type-1 error rates

2023-03-04

Contents

1 Introduction

- This file documents the commands in the `simnull.R` script that simulates data under the null hypothesis for the type-1 error rates.
- Simulations are done on the Compute Canada cluster.
  - `simnull.sh` and `simnull.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 of checked pedigrees from which to sample from for a study (produced by `checkpeds.R` and named `pedpool.txt`) and the .RData file of chromosome 8 sequences from which to sample founder sequences (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 200 batches of 10 studies, run as an "array job" on the cluster. In an "array job", the same script is run multiple times, each time with a unique "job ID".
  - The SLURM script to run the array job is `simnull.sh`. This runs the R script `simnull.R` generated by this `simnull.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

- Each study contains three pedigrees and we run simulations in batches of 10 studies.

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

- 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 of the carrier probability.

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

2.1.4 Create output files

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

```
oldop <- options(scipen=999) #prevent scientific notation in output
```

- **Note:** The following detailed description of the output files can be skipped on first reading. However, you'll need to return to this description when working through `simnullSummary.Rmd` which reads in and summarizes the simulation output.

- For each simulated study and each cRV within a study, we will record the p-values of the 5 testing methods.
- Here we create comma-delimited files to hold the p-values for each batch of 10 studies, and write column names to each file.
- The p-value files include the following columns of information about each study: its replicate number, the IDs of the three study pedigrees, and the IDs of the cRVs observed in the study.
- For n_p carrier probabilities, there are up to $3 \times (2 \times n_p + 3)$ p-values to 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 global transmission approaches, 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 the third cRV is `gobalR_5_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} cRVs 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`.

```
pfile <- paste0(outfileDir,"/pvalnullres",dID,".csv") # file for type 1 error rate 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
```

2.2 Simulation study

- Our goal is to simulate a batch of `N` studies and, for each study, calculate the p-values 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:
  - Sample `studysize` (i.e. 3) pedigrees to make a study.
  - Sample cRVs and founder sequence data for the study.
  - Get look-up tables of p-values for each possible global configuration of the study.
  - Use the look-up tables to find the p-value for the observed global configuration of each sampled cRV.
  - Write the results to an output file.

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 sequence data

- Christina's `sim_RVstudy()` function is primarily for simulating sequence data under the alternative hypothesis of cRVs for the disease; i.e., it simulates cRVs and does *conditional* gene drop of sequences through the pedigrees in a study. However, `sim_RVstudy()` can also simulate under the null hypothesis of no cRVs; i.e., it can do *unconditional* gene drop of sequences.
  - The trick for simulating under the null hypothesis is to remove the disease-allele status variables in the pedigree data structure (variables `DA1` and `DA2`) before passing it to `sim_RVstudy()`. This trick is describe briefly in the **Details** section of the `sim_RVstudy()` documentation, just below the description of the input variable `ped_files`.
  - With the disease-allele status variables removed, `sim_RVstudy()` (i) samples founder sequences from Nirodha's American Admixed population, and (ii) unconditionally gene drops these sequences through the pedigree. `sim_RVstudy()` then combines the resulting sequence data into one object to be returned to the user.
- With unconditional gene drop, all variants are simulated under the null hypothesis of no cRVs for the disease. As a result, any of the RVs that are observed in the affected individuals could be used to estimate the type-1 error rate. Christina chose not to use just any RV, however, because she wanted the frequencies of the variants used to estimate the type-1 error rate to match the frequencies of the variants used to estimate power. She therefore repeated the gene-drop simulation until at least one of the variants from the pool of candidate cRVs was observed in the affected individuals (see the paragraph just below the numbered list on page 50 of her thesis). Christina's approach for the simulations under the null hypothesis ensures that its variant frequencies are calibrated to the variant frequencies under the alternative hypothesis.
- Recall from `simalt.Rmd` that the sequences returned by `sim_RVstudy()` require filtering to remove RVs not observed in the sequenced affected individuals (because `sim_RVstudy()` also keeps RVs in pedigree ancestors that connect the sequenced affected individuals).

2.2.1.3 Get look-up tables of p-values

- 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 of the observed configuration

- For each sampled cSNV, we observe its global configuration and look up the p-values for this global configuration in the p-value lookup table.

2.2.1.5 Write the results to files

- Finally, we write the p-values to an output file.

2.2.2 Running a batch of simulated studies with `simnullreps()`

- Coding note: The simulation study is encapsulated in its own function, `simnullreps()` that calls worker functions to handle some of the more mundane details. Later in the Rmarkdown file, I call `simnullreps()` after it and all of its worker functions have been defined. See Section 2.2.2 of `simalt.Rmd` for the rationale behind this approach.
  - I label my worker functions `*worker*` and label Christina's functions from `SimRVSequences` `*Christina's*`.

```
simnullreps <- function(N,studysize,pfile,infileDir){
for(simrep in 1:N) {
  cat("simrep",simrep,"\n")
  #-------------------------------------------------------------
  #  Sample the IDs of 3 peds from 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*
  # Remove the DA1 and DA2 columns of the pedigree data structure so that
  # sim_RVstudy() will do unconditional gene drop.
  s_peds$DA1 <- s_peds$DA2 <- NULL
  #-------------------------------------------------------------
  # Generate sequence data with sim_RVstudy() until we get a data set with one
  # of the cRVs in our list of candidate cRVs is observed in the affecteds.
  # We then filter the sequence data further.
  foundcRV <- FALSE; genedropcounter<- 0
  while(!foundcRV) {
    genedropcounter <- genedropcounter+1
    s_seqs <- sim_RVstudy(s_peds,chr8) # *Christina's*. Seqs for all study members
    # see if any cRVs are observed in the affecteds
    if(sum(s_seqs$ped_haplos[s_seqs$haplo_map$affected,s_seqs$SNV_map$is_CRV]) > 0){
      foundcRV <- TRUE
    }
  }
  # While debugging I reported how many gene drops we had to do:
  # cat("Did",genedropcounter,"genedrops to get a cRV in affecteds \n")
  # It was anywhere from 20 to over 1000!
  a_seqs <- filter2aff(s_seqs) # *worker*
  numRVs <- nrow(a_seqs$SNV_map)
  #-------------------------------------------------------------
  # Call cd_new() to get lookup tables of statistics and p-values for each
  # possible global configurations of affected individuals in the study.
  # Even with the repeated gene drops in the above while() loop, this is
  # still the most computationally-intensive part of the simulation.
  lookupTabs = cd_new(peds = s_peds, subtypes = c("HL", "NHL"),
                      carrier_probs = carrier_probs)
  # For each cRV observed in the affecteds, find its global configuration and then
  # the corresponding p-values from the lookup table.
  cRVs <- a_seqs$SNV_map$marker[a_seqs$SNV_map$is_CRV]
  pvals <- get_pvals(cRVs,lookupTabs,a_seqs) # *worker*
  # Write the pval results to a file. Also write info
  # on the rep number, pedigree IDs and observed cRVs in each pedigree
  cRVs <- cRVs[1:3] # force vector of length 3, with NAs if length of cRVs < 3
  repinfo <- c(simrep,studypedIDs,cRVs)
  pvalinfo <- c(pvals[1,],pvals[2,],pvals[3,])
  cat(paste0(paste0(c(repinfo,pvalinfo),collapse=","),"\n"),
      file=pfile,append=TRUE)
} # end for loop over studies
  # Note that simnullreps() can return NUL, because its output is written to a file.
  return(NULL)
} # end simnullreps()
```

2.2.3 Utility functions for `simnullreps()`

- The worker functions called in `simnullreps()` are a subset of those called in `simalt.Rmd`: `read_studypeds()`, `filter2aff()`, `find_binID`, `config2binID()`, and `get_pvals()`. We have to define them again here so that the R script generated by the `.Rmd` file works but they are all exactly the same as for `simalt.Rmd`. (Note however, that the function `get_ranks()` that ranks the five statistics for each study is not required for simulations under the null hypothesis and so is not included.)

```
# read_studypeds() takes a vector if 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 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 pedigreees.
      s_peds <- pp
    } else{ # Add new ped to the data frame of previous ones.
      s_peds <- rbind(s_peds,pp) # Return the pedigrees.
    }
  }
  return(s_peds)
}
# filter2aff() takes a "famStudy" object returned by sim_RVstudy() and
# reduces its sequence data to variants that are seen in the
# affected individuals. It also finds the genotypes for each
# variant/individual.
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 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$SNV_map[cc>0,]
  # Lastly, pair sequences from individuals into multilocus genotypes
  odd_inds <- seq(from=1,to=nrow(ped_haplos)-1,by=2)
  even_inds <- seq(from=2,to=nrow(ped_haplos),by=2)
  ped_genos <- ped_haplos[odd_inds,] + ped_haplos[even_inds,]
  return(list(ped_haplos=ped_haplos, haplo_map=haplo_map,
              SNV_map=SNV_map, ped_genos=ped_genos))
}
# find_binID takes the name of an RV and the reduced sequence data on
# affecteds output by filter2aff() and returns the base-10 representation
# of the configuration (what Christina calls the "binID").
find_binID <- function(RV,a_seqs) {
  config_vec <- a_seqs$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_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)
}
```

2.2.4 Call `simnullreps()`

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

```
simnullreps(N,studysize,pfile,infileDir)
```

2.3 Generate the R script for the cluster.

- Execute the following R command:

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

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