Supplementary Material 2: Simulate ascertained pedigrees

Nirodha Epasinghege Dona, Jinko Graham

2022-01-30

Contents

This document discusses the second major step in our work-flow to simulate exome-sequencing data in pedigrees ascertained to have four or more relatives affected with lymphoid cancer (the blue box labelled 2).

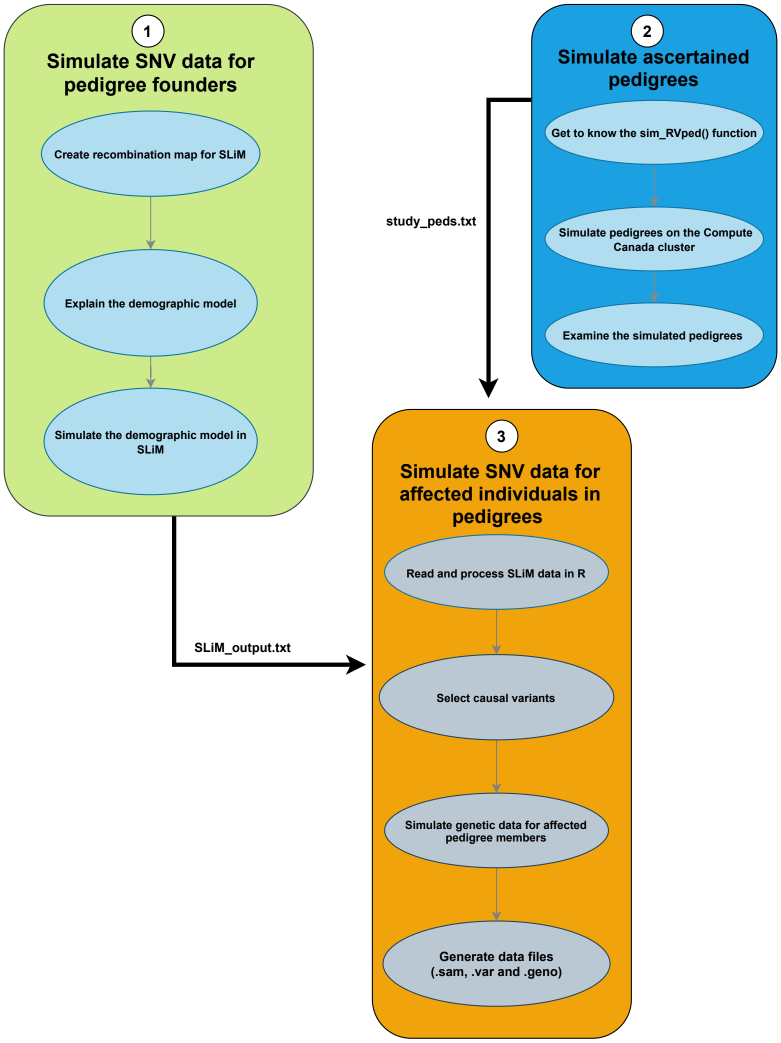

1
Simulate SNV data for pedigree founders
Create recombination map for SLiM
Explain the demographic model
Simulate the demographic model in SLiM
2
Simulate ascertained pedigrees
Get to know the sim_RVped() function
Simulate pedigrees on the Compute Canada cluster
Examine the simulated pedigrees
study_peds.txt
SLiM_output.txt
3
Simulate SNV data for affected individuals in pedigrees
Read and process SLiM data in R
Select causal variants
Simulate genetic data for affected pedigree members
Generate data files (.sam, .var and .geno)

Figure 1: Work-flow for simulating the exome-sequencing data for ascertained pedigrees.

1 Get to know the `sim_Rvped()` function

We use the SimRVPedigree (Nieuwoudt et al. 2018) R package to simulate 150 pedigrees ascertained to contain four or more relatives affected with lymphoid cancer. Affected pedigree members can have either sporadically occurring disease or genetic disease caused by a single rare variant that is segregating in the pedigree. We refer to the causal rare variants as cRVs. The package constructs a pedigree by growing it from a single starting individual or “seed founder” and obtaining the seed founder’s descendants. A cRV may be introduced into the seed founder with probability equal to either one or the carrier probability of a cRV in the population. When a cRV is introduced into the seed founder with probability equal to the carrier probability of a cRV in the population, the pedigree is ascertained from the **general population**. A *genetic* pedigree is defined as a pedigree in which the seed founder carries a cRV, whereas a *sporadic* pedigree is defined as a pedigree in which the seed founder does not carry a cRV. Ascertained pedigrees may be either genetic or sporadic, and may contain both genetic and sporadic cases. Once a cRV is introduced into a seed founder, it is transmitted from parent to offspring according to Mendel's law. The age-specific life events of the seed founder and his/her descendants such as birth, disease onset and death are modelled according to the cRV carrier status of the individual. The modeling requires specification of the age-specific incidence rates of disease, the age-specific hazard rates of death and the genetic relative-risk (GRR) of disease.

We generate a sample of 150 ascertained pedigrees from the **general population** using `simRV_ped()`, the core function of the `SimRVPedigree` package. The `sim_RVped()` function generates a **single** pedigree by simulating all life events of the seed founder and his/her descendants, as described by Nieuwoudt et al. (2018). Starting at the birth of an individual, the waiting times of the possible next life events of the individual – disease onset, reproduction and death – are generated. The event with the minimum waiting time is selected as the individual's next life event. The waiting time is added to the current age of the individual and the corresponding life event is recorded. These steps are repeated until the individual dies or the study reaches its stop year. Further details of this function can be found in Nieuwoudt and Graham (2018).

The required arguments of `simRV_ped()` are: `hazard_rates`, `GRR`, `num_affected`, `ascertain_span`, `FamID`, and `founder_byears`. Non-required arguments of specific interest to us are: `stop_year`, `carrier_prob`, `RV_founder`, `recall_probs` and `first_diagnosis`. A short description of these arguments follows.

- **`hazard_rates`-** We use the `AgeSpecific_Hazards` dataset in the `SimRVPedigree` package. The first column of the `AgeSpecific_Hazards` data-frame gives the age-specific hazard rates for the disease in the general population. The second column gives the age-specific hazard rates for death in the unaffected population. The third column gives the age-specific hazard rates for death in the affected population.

- **`GRR`-** the genetic relative-risk; i.e, the risk of disease for individuals who carry a copy of a cRV relative to those who carry no copies of a cRV. We use 50 as the GRR.

- **`num_affected`-** the minimum number of disease-affected members needed to ascertain the pedigree is set to 4.

- **`ascertain_span`-** the period of ascertainment of the pedigree; i.e., (start-year, end-year) is set to (2000 , 2010).

- **`FamID`-** the family identity number of the simulated pedigree. We assign a vector that contains values 1 to 150 since we need to generate 150 pedigrees.

- **`founder_byears`-** the period for the possible birth year of a seed founder is set to be (1880, 1920).

- **`stop_year`-** 2020 is set to the year in which we stop collecting data.

- **`carrier_prob`-** the probability that an individual in the general population carries a cRV is set to 0.001.

- **`RV_founder`-** is set to `FALSE`, i.e., the seed founder carries a cRV with probability equal to the carrier probability (0.001) of a cRV in the population.

- **`recall_probs`-** the proband's recall probabilities of relatives in the pedigree is set to (1, 1, 1, 1, 0.75, 0.5, 0.25, 0.125, 0). These probabilities imply that first to fourth-degree relatives of the proband (e.g. fourth degree = great aunt) are recalled with probability 1, all fifth-degree relatives (e.g. first cousin once removed) of the proband are recalled with probability 0.75, and so forth.

- **`first_diagnosis`-** the earliest year after which reliable diagnoses can be made regarding the disease-affection status is set as 1940.

We set the above values for the arguments in the `sim_RVped()` function and use the Compute Canada cluster for the simulation. We use an array job on the cluster, with a processor (CPU) to simulate each pedigree.

Each pedigree takes time to simulate and it varies from pedigree to pedigree.

2 Simulate pedigrees on the Compute Canada cluster

We use the following slurm batch file to submit an array job with a processor for each of the 150 pedigrees.

```
#!/bin/bash
#SBATCH --account=def-jgraham
#SBATCH --array= 1-150
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=4000M
#SBATCH --time=23:59:00

module load  nixpkgs/16.09 gcc/5.4.0  r/3.5.0

echo "This is job $SLURM_ARRAY_TASK_ID out of $SLURM_ARRAY_TASK_COUNT jobs."

R CMD BATCH --no-save SimRVpedigree.R
```

In the batch script above, the parameters are set as follows.

- `#SBATCH-account=def-jgraham` - specifies the project account on Compute Canada. In this example, the project account is "def-jgraham."
- `# SBATCH -array= 1:150`- specifies the array of tasks, one for each of the 150 pedigrees.
- `#SBATCH-ntasks=1` - defines the number of array tasks per processor. We request 1 processor to run each task (i.e. each one of our 150 pedigrees, will use 1 CPU).
- `#SBATCH-mem=40000M` - specifies memory that we require to run each task in the array. We request 4GB.
- `#SBATCH-time=23:59:00`- specifies the time limit for each task. If we allocate 24 hours or more to run a task, we must wait longer in the queue to start a task than if we allocate less than 24 hours.
- `module load nixpkgs/16.09 gcc/5.4.0 r/3.5.0` - loads the R version that we installed.
- `echo "This is job $ SLURM_ARRAY_TASK_ID out of $ SLURM_ARRAY_TASK_COUNT jobs."`- prints the job number out of 150 tasks.

We use the `R CMD BATCH` command to submit the `SimRVpedigree.R` script to the cluster. The contents of the script is given below.

```
# load the SimRVPedigree library
library(SimRVPedigree)

# Create hazard object from AgeSpecific_Hazards data
data(AgeSpecific_Hazards)
my_HR = hazard(AgeSpecific_Hazards)

# Get the Unix environmental variable for array job id.
# This id is created by the cluster for each job.
dID = Sys.getenv("SLURM_ARRAY_TASK_ID")

# Set a seed value to assure the reproducibility.
seed = as.numeric(dID)
set.seed(seed)
```

```
analyseData = function(dataID){

  # Read the R function that do the analysis
  out =  sim_RVped(hazard_rates = my_HR,
            GRR = 50, FamID = dataID,
            RVfounder = FALSE,
            founder_byears = c(1880, 1920),
            ascertain_span = c(2000, 2010),
            stop_year = 2020,
            recall_probs = c(1, 1, 1, 1, 0.75, 0.5, 0.25, 0.125, 0),
            carrier_prob = 0.001,
            num_affected = 4,
            first_diagnosis = 1940)[[2]]

  # Save the results separately for each dataset.
  write.table(out, file = paste0("/project/6007536/epasiedn/Array_jobs/",
                          dataID,".txt"))
}

# Run the function.
analyseData(dID)
```

In the above script, we create a function `analyseData()` which calls the `sim_RVped()` function. The `analyseData()` function has one argument,`dataID`, for the task identifier created by the cluster scheduler. The environment variable, `SLURM_ARRAY_TASK_ID`, identifies each task in the array. In the last two lines of the R script above, we use `dID = Sys.getenv("SLURM_ARRAY_TASK_ID")` to get the task identifier and assign it as the argument to the `analyseData()` function. `dID` is also assigned as the random seed for each pedigree. Since each pedigree is a task that is run separately on a different CPU in the cluster, we want to assign a different seed value each time.

Among the 150 tasks, 140 manage to run within the allocated time period. Some tasks take longer because some pedigrees take a longer time to ascertain. The unfinished tasks are run again, with a different random seed. We use the linux command `ls` to identify the finished jobs. This command returns all the files in our directory, so that we can see which task IDs are missing. Among all 150 tasks, IDs 14, 30, 48, 50, 63, 73, 83, 94, 102 and 129 are unfinished. The following code chunk shows how we identify the unfinished tasks.

```
[epasiedn@cedar1 Array_jobs_check]$ ls
100.txt  140.txt  45.txt  88.txt                  slurm-19422255_124.out  slurm-19422255_26.out  slurm-
101.txt  141.txt  46.txt  89.txt                  slurm-19422255_125.out  slurm-19422255_27.out  slurm-
103.txt  142.txt  47.txt  8.txt                   slurm-19422255_126.out  slurm-19422255_28.out  slurm-
104.txt  143.txt  49.txt  90.txt                  slurm-19422255_127.out  slurm-19422255_29.out  slurm-
105.txt  144.txt  4.txt   91.txt                  slurm-19422255_128.out  slurm-19422255_2.out   slurm-
106.txt  145.txt  51.txt  92.txt                  slurm-19422255_129.out  slurm-19422255_30.out  slurm-
107.txt  146.txt  52.txt  93.txt                  slurm-19422255_12.out   slurm-19422255_31.out  slurm-
108.txt  147.txt  53.txt  95.txt                  slurm-19422255_130.out  slurm-19422255_32.out  slurm-
109.txt  148.txt  54.txt  96.txt                  slurm-19422255_131.out  slurm-19422255_33.out  slurm-
10.txt   149.txt  55.txt  97.txt                  slurm-19422255_132.out  slurm-19422255_34.out  slurm-
110.txt  150.txt  56.txt  98.txt                  slurm-19422255_133.out  slurm-19422255_35.out  slurm-
111.txt  15.txt   57.txt  99.txt                  slurm-19422255_134.out  slurm-19422255_36.out  slurm-
112.txt  16.txt   58.txt  9.txt                   slurm-19422255_135.out  slurm-19422255_37.out  slurm-
113.txt  17.txt   59.txt  job_array.sh            slurm-19422255_136.out  slurm-19422255_38.out  slurm-
114.txt  18.txt   5.txt   SIMrvpedigree.R         slurm-19422255_137.out  slurm-19422255_39.out  slurm-
115.txt  19.txt   60.txt  SIMrvpedigree.Rout      slurm-19422255_138.out  slurm-19422255_3.out   slurm-
116.txt  1.txt    61.txt  slurm-19422255_100.out  slurm-19422255_139.out  slurm-19422255_40.out  slurm-
```

```
117.txt  20.txt   62.txt  slurm-19422255_101.out  slurm-19422255_13.out   slurm-19422255_41.out  slurm-
118.txt  21.txt   64.txt  slurm-19422255_102.out  slurm-19422255_140.out  slurm-19422255_42.out  slurm-
119.txt  22.txt   65.txt  slurm-19422255_103.out  slurm-19422255_141.out  slurm-19422255_43.out  slurm-
11.txt   23.txt   66.txt  slurm-19422255_104.out  slurm-19422255_142.out  slurm-19422255_44.out  slurm-
120.txt  24.txt   67.txt  slurm-19422255_105.out  slurm-19422255_143.out  slurm-19422255_45.out  slurm-
121.txt  25.txt   68.txt  slurm-19422255_106.out  slurm-19422255_144.out  slurm-19422255_46.out  slurm-
122.txt  26.txt   69.txt  slurm-19422255_107.out  slurm-19422255_145.out  slurm-19422255_47.out  slurm-
123.txt  27.txt   6.txt   slurm-19422255_108.out  slurm-19422255_146.out  slurm-19422255_48.out  slurm-
124.txt  28.txt   70.txt  slurm-19422255_109.out  slurm-19422255_147.out  slurm-19422255_49.out  slurm-
125.txt  29.txt   71.txt  slurm-19422255_10.out   slurm-19422255_148.out  slurm-19422255_4.out   slurm-
126.txt  2.txt    72.txt  slurm-19422255_110.out  slurm-19422255_149.out  slurm-19422255_50.out  slurm-
127.txt  31.txt   74.txt  slurm-19422255_111.out  slurm-19422255_14.out   slurm-19422255_51.out  slurm-
128.txt  32.txt   75.txt  slurm-19422255_112.out  slurm-19422255_150.out  slurm-19422255_52.out  slurm-
12.txt   34.txt   76.txt  slurm-19422255_113.out  slurm-19422255_15.out   slurm-19422255_53.out  slurm-
130.txt  35.txt   77.txt  slurm-19422255_114.out  slurm-19422255_16.out   slurm-19422255_54.out  slurm-
131.txt  36.txt   78.txt  slurm-19422255_115.out  slurm-19422255_17.out   slurm-19422255_55.out  slurm-
132.txt  37.txt   79.txt  slurm-19422255_116.out  slurm-19422255_18.out   slurm-19422255_56.out  slurm-
133.txt  38.txt   7.txt   slurm-19422255_117.out  slurm-19422255_19.out   slurm-19422255_57.out  slurm-
134.txt  39.txt   80.txt  slurm-19422255_118.out  slurm-19422255_1.out    slurm-19422255_58.out  slurm-
135.txt  3.txt    81.txt  slurm-19422255_119.out  slurm-19422255_20.out   slurm-19422255_59.out  slurm-
136.txt  40.txt   82.txt  slurm-19422255_11.out   slurm-19422255_21.out   slurm-19422255_5.out   slurm-
137.txt  41.txt   84.txt  slurm-19422255_120.out  slurm-19422255_22.out   slurm-19422255_60.out  slurm-
138.txt  42.txt   85.txt  slurm-19422255_121.out  slurm-19422255_23.out   slurm-19422255_61.out  slurm-
139.txt  43.txt   86.txt  slurm-19422255_122.out  slurm-19422255_24.out   slurm-19422255_62.out
13.txt   44.txt   87.txt  slurm-19422255_123.out  slurm-19422255_25.out   slurm-19422255_63.out
```

For these unfinished tasks, we need to assign a new seed value which we set to be `job number * 20`; i.e. `seed = as.numeric(dID)*20`. We select 20 as the multiplier to avoid repeating the seed values. For example, a multiplier of 10 doesn't work because if we multiply task ID 14 by 10, we get 140 as the seed, which has already been used for pedigree ID 140 in the previous run. In this way, we obtain the 150 simulated pedigrees in separate files (i.e. 1.txt, 2.txt,. . . ,150.txt), read them all into R and save them in a single file called `study_peds.txt`. We use a `for`-loop to read in the pedigrees and save them in a list. Then we combine all 150 list elements into a single data frame as shown in the next code chunk.

```r
## Load all 150 simulated pedigrees, save them in a single list, and write them to a text file.

study_peds <- list()

for(i in 1:150){

  study_peds[[i]] <- read.table(paste0(i,".txt"))

  }

study_peds <- do.call("rbind", study_peds)

write.table(study_peds, file = "study_peds.txt")
```

3 Examine the simulated pedigrees

In the next code chunk, we read `study_peds.txt` into R as a data frame and convert it to class `ped` using the `new.ped()` function of the `SimRVPedigree` R package.

```
library(SimRVPedigree)

# import study peds
study_peds <- read.table("study_peds.txt", header=TRUE, sep= " ")

# create an object of class ped, from a data.frame,
study_peds <- new.ped(study_peds)

head(study_peds)
```

```
##   FamID ID sex dadID momID affected DA1 DA2 birthYr onsetYr deathYr available
## 1     1  1   1    NA    NA     TRUE   0   1    1881    1952    1955      TRUE
## 2     1  2   0    NA    NA    FALSE   0   0      NA      NA      NA     FALSE
## 3     1  3   1     2     1     TRUE   0   1    1901    1970    1981      TRUE
## 5     1  4   0     2     1     TRUE   0   1    1910    2000    2002      TRUE
## 6     1  5   1     2     1    FALSE   0   0    1913      NA    1991      TRUE
## 8     1  7   1     6     3    FALSE   0   1    1924      NA    1956      TRUE
##   Gen proband
## 1   1   FALSE
## 2   1   FALSE
## 3   2   FALSE
## 5   2    TRUE
## 6   2   FALSE
## 8   3   FALSE
```

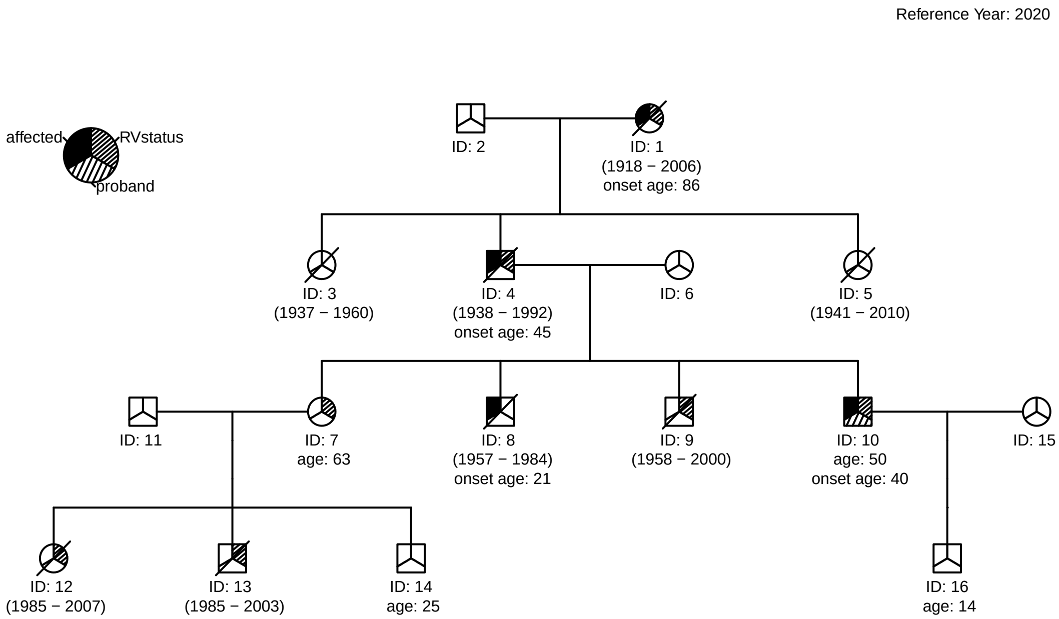
The rows of `study_peds` represent individuals and the columns are:

1. `FamID`- the identity number of the ascertained pedigree.
2. `ID`- the individual identity number.
3. `sex`- sex of the individual, with `sex` = 0 for males and `sex` = 1 for females.
4. `dadID`- individual identity number of the father.
5. `momID`- individual identity number of the mother.
6. `affected`- the disease status of the individual, with `affected = TRUE` if the individual has developed disease and `affected = FALSE` otherwise.
7. `DA1`- the cRV status of the paternally inherited allele, with `DA1 = 1` if the cRV is inherited and 0 otherwise.
8. `DA2`- the cRV status of the maternally inherited allele, with `DA2 = 1` if the cRV is inherited and 0 otherwise.
9. `birthYr`- the birth year of the individual.
10. `onsetYr`- the disease-onset year of the individual, when applicable, and NA otherwise.
11. `deathYr`- the death year of the individual, when applicable, and NA otherwise.
12. `RR`- the genetic relative-risk of disease for carriers of the cRV.
13. `available`- the availability of life-events information on the individual. Specifically, if an individual descends from the seed founder and is recalled by the proband then `available = TRUE`. If an individual descends from the seed founder and is not recalled by the proband then `available = FALSE`. Finally, if an individual is not descended from the seed founder (i.e. has married into the pedigree) `available = FALSE`.
14. `Gen`- the generation of the individual within the pedigree.

15. `proband`- the proband status, with `proband = TRUE` if the individual is the proband and `FALSE` otherwise.

Let's draw a pedigree in `study_peds`, in the year 2020. We will take the 39th pedigree out of the 150 generated:

```
plot(study_peds[study_peds$FamID == 39, ], ref_year = 2020,
     cex = 0.5)
```



The legend identifies affected individuals, the proband, and the cRV status of the individuals. Disease-affected individuals have solid shading in the upper-left third of their symbol (IDs 1, 4, 8 and 10). The proband (ID 10) has shading in the lower portion of their symbol. Carrier individuals (IDs 1, 4, 7, 9, 10, 12 and 13) have shading in the upper-right portion of their symbol. The seed founder is the individual with ID 1 and he and all his descendants have ages relative to the reference year of 2020. We create age labels at a selected reference year by providing the argument `ref_year` to the `plot()` function. The birth year and the death year of dead individuals are displayed in parentheses. The age of the individuals who are alive at the end of the reference year displays under their symbol. Any individual with disease onset before the end of the reference year has a disease-onset year given under their symbol.

For reference, session information giving the versions of R and packages used by the `SimRVPedigree` package is as follows.

```
# Get the session information
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22000)
##
## Matrix products: default
```

```
##
## locale:
## [1] LC_COLLATE=English_Canada.1252  LC_CTYPE=English_Canada.1252
## [3] LC_MONETARY=English_Canada.1252 LC_NUMERIC=C
## [5] LC_TIME=English_Canada.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] SimRVPedigree_0.4.4
##
## loaded via a namespace (and not attached):
##  [1] quadprog_1.5-8  lattice_0.20-44 digest_0.6.27   grid_4.1.1
##  [5] magrittr_2.0.1  evaluate_0.14   rlang_0.4.11    stringi_1.7.4
##  [9] kinship2_1.8.5  Matrix_1.2-12   rmarkdown_2.11  tools_4.1.1
## [13] stringr_1.4.0   xfun_0.25       yaml_2.2.1      fastmap_1.1.0
## [17] compiler_4.1.1  htmltools_0.5.2 knitr_1.34
```

In the third and final step of our workflow, to be discussed next, the file `study_peds.text` will be used to simulate the exome-sequencing data for the 150 ascertained pedigrees.

References

Nieuwoudt, Christina, and Jinko Graham. 2018. “SimRVPedigree: Simulate Pedigrees Ascertained for a Rare Disease. R package version 0.1.0. https://CRAN.R-project.org/package=SimRVPedigree.” https://doi.org/10.1101/234153%3E.Depends.

Nieuwoudt, Christina, Samantha J. Jones, Angela Brooks-Wilson, and Jinko Graham. 2018. “Simulating pedigrees ascertained for multiple disease-affected relatives.” *Source Code for Biology and Medicine*. https://doi.org/10.1186/s13029-018-0069-6.