

# Supplementary Material 2 : Simulate ascertained pedigrees

Nirodha Epasinghege Dona, Jinko Graham

2021-12-14

## Contents

<b>1</b>	<b>Get to know the <code>sim_Rvped()</code> function</b>	<b>2</b>
<b>2</b>	<b>Simulate pedigrees on the Compute Canada cluster</b>	<b>4</b>
<b>3</b>	<b>Examine the simulated pedigrees</b>	<b>7</b>
	<b>Reference</b>	<b>11</b>

- In this document, we discuss the second major phase of our work flow, shown in the blue box of the flow chart below.

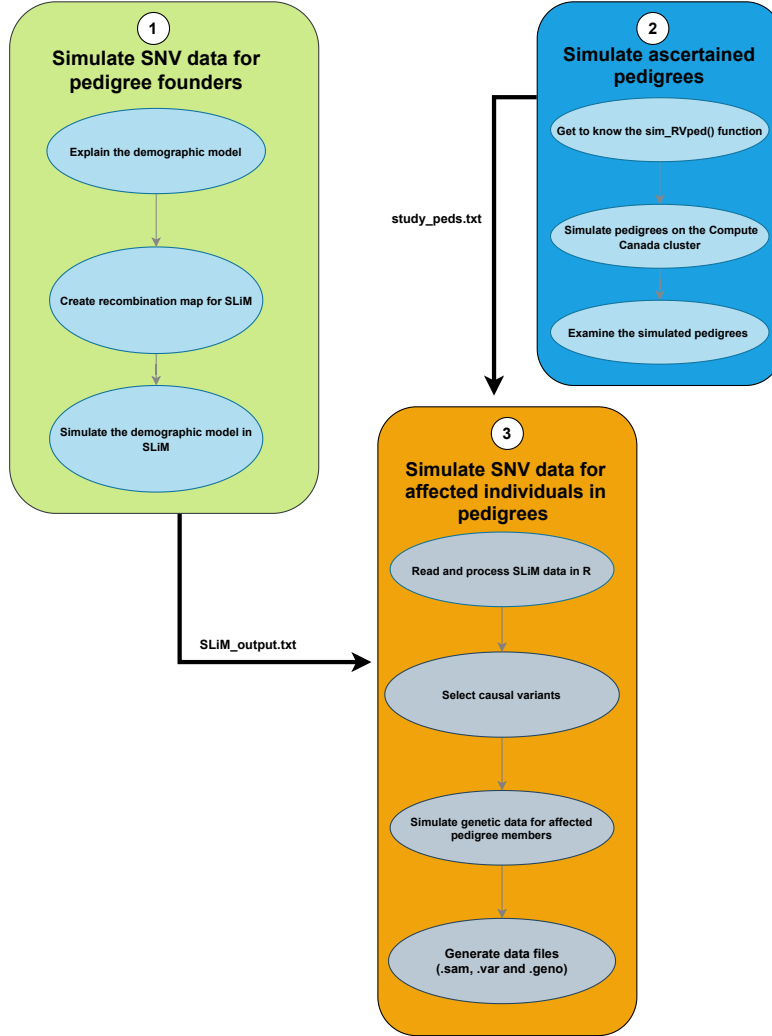


Figure 1: The flow chart which represents the entire work flow of simulating the sequence data for the pedigrees.

## 1 Get to know the `sim_Rvped()` function

- The R package `SimRVPedigree` (Nieuwoudt et al. 2018) simulates pedigrees ascertained to contain a minimum number of disease-affected relatives.
- 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.

- We define a **genetic pedigree** to be a pedigree in which the seed founder carries a cRV. In contrast, a **sporadic pedigree** is a pedigree in which the seed founder does not carry a cRV.
- The package constructs a pedigree by growing it from a single starting individual or “seed founder” and obtaining the seed founder’s descendants.
- The package allows us to introduce a cRV into the seed founder with probability equal to either one or the carrier probability of the cRV in the population. Once the cRV is introduced into a pedigree, it is transmitted from parent to offspring according to Mendel’s law.
- Specifically, the package models the age-specific life events of the seed founder and his/her descendants such as birth, disease onset and death based on cRV status.
- 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.
- A cRV is introduced into the seed founder with probability equal to that of carrying any of the possible cRVs in the population. This simulates pedigrees ascertained from the **general population**.
- Ascertained pedigrees may be either genetic or sporadic, and may contain both genetic and sporadic cases.
- The second phase of our simulation study is generating a sample of 150 ascertained pedigrees, realistically.
- For this task, we use the core function in the **SimRVPedigree** package: the **sim\_Rvped()** function.
- 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 function generates the waiting times of their next three life events: disease onset, reproduction and death.
- The function then selects the event with the minimum waiting time as their next life event. It adds this waiting time to the current age of the individual and record the corresponding life event.
- The function repeats these steps until the individual dies or the study reaches its stop year.
- The complete details of using this function can be found in Nieuwoudt and Graham (2018).
- We use the **sim\_Rvped()** function to simulate 150 pedigrees from the population of general pedigrees.
- The following is the set up of the function arguments.
- The required arguments, explained below, are: **hazard\_rates**, **GRR**, **num\_affected**, **ascertain\_span**, **FamID**, and **founder\_byears**. Other arguments of specific interest in the upcoming simulation are also explained below including: **stop\_year**, **carrier\_prob**, **RV\_founder**, **recall\_probs** and **first\_diagnosis**.
- **hazard\_rates**- We use the **AgeSpecific\_Hazards** dataset in the **SimRVPedigree** package. The first column of the **AgeSpecific\_Hazards** data-frame represents the age-specific hazard rates for the disease in the general population. The second column represents the age-specific hazard rates for death in the unaffected population. The third column represents 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 the cRV relative to those who carry no copies of the 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 of the birth year of the seed founder is set to the (1880, 1920) interval.
- **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 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 first year that the 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. The reason for this is the simulation time is comparatively high and it varies pedigree to pedigree.
- The next sub-section explains the Compute Canada cluster simulation.

## 2 Simulate pedigrees on the Compute Canada cluster

- In the cluster we submit the job as an array job. (We submit 150 jobs and each job represents a pedigree.)
- The following code chunk is used as the slurm batch file.

```
#!/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
```

- When we submit an array job the resources asked for correspond to one sub job (one array).
- Let's understand each line of the above batch script file.
- **#SBATCH-account=def-jgraham** - specifies the account name. In this example, the account name is "def-jgraham."
- **#SBATCH -array= 1:150**- specifies the array. We create an array starting from 1 to 150. (The number of pedigrees.)
- **#SBATCH-ntasks=1** - defines the number of processors. We request 1 processor to run each sub job (each one of our 150 sub jobs, will use 1 CPU )
- **#SBATCH-mem=4000M** - specifies memory that we require to run each sub job (each array). We request 4GB.

- `#SBATCH--time=23:59:00-` specifies the time limit for each sub job. If we allocate 24 hours or more to run a sub job, we must wait longer in the queue to start a sub job 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 jobs.
- We use R CMD BATCH command to submit our R script.
- The `SimRVPedigree.R` script is as follows:

```
# 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_byyears = 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()`, and inside this function we call the `sim_RVped()` function.
- The `analyseData` function has one argument, `dataID`, and this represents the array identity number created by the cluster job.
- The environment variable, `SLURM_ARRAY_TASK_ID`, identifies the individual identity number of each job in the array.

- We use `dID = Sys.getenv("SLURM_ARRAY_TASK_ID")` to get the array job identities and assign this as the `analyseData()` function argument (last two lines of the above R script).
- Also, the same variable, `dID`, is assigned as the seed value of each iteration. Since each job runs separately in a different CPU in the cluster, we can not assign the same seed value for each job. Therefore, we change the seed value for each job. The job number is set as the seed value.
- Among 150 sub jobs, 140 jobs are run within allocated time period. However, the unfinished jobs are run again. (The reason for this is some pedigrees take a longer time to ascertain.)
- We use the linux command `ls` to identify the finished jobs. This command returns all the files in our directory and among them we get the job IDs that are missing. Among all 150 jobs, IDs 14, 30, 48, 50, 63, 73, 83, 94, 102 and 129 are unfinished. We run these jobs again. The following code chunk represents how we identify the unfinished jobs.

```
[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	

- Also, for these unfinished jobs, we need to assign a new seed value. For that we set, `job number * 20` as the seed value for each unfinished job. (i.e. `seed = as.numeric(dID)*20`.) The reason of selecting 20 as the multiplier is to avoid repeating the seed values. As an example, if we multiply the job ID 14 by 10, we get 140 as the seed and we already use this seed for the family ID 140 in the previous run.
- We get the 150 simulated pedigrees separately (i.e. we have 1.txt, 2.txt,...,150.txt objects) and we read all of them into R and save as `study_peds.txt` file.
- We use a for loop to read all the 150 pedigrees and save them in a list. Then we combine all 150 list elements to a single data frame. The following code chunk explains this.

```
## Load all the simulated 150 pedigrees and save them in a single list

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

- We read the file, `study_peds.txt`, to R and convert the data frame to the class of `ped` using the `new.ped()` function in the `SimRVPedigree` R package.
- We print the session information at the end of this code chunk.

```
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)
```

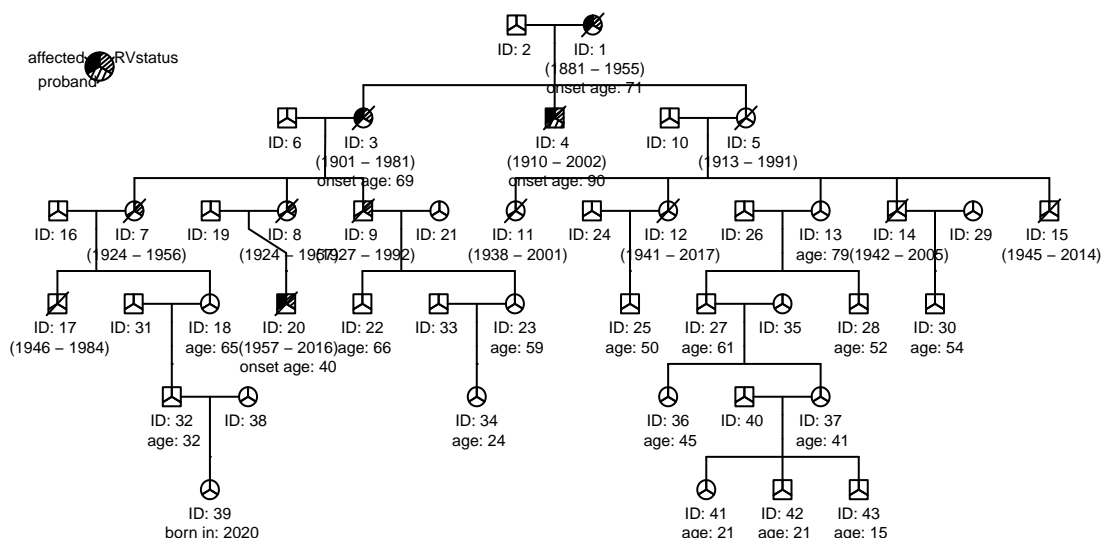
```
##   FamID ID sex dadID momID affected DA1 DA2 birthYr onsetYr deathYr available
## 1     1  1  1    NA    NA     TRUE  0  1   1881   1952   1955      TRUE
## 2     1  2  0    NA    NA    FALSE  0  0    NA     NA     NA      FALSE
## 3     1  3  1     2     1     TRUE  0  1   1901   1970   1981      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 `study_peds`, is a data-frame that represents the ascertained pedigrees.
- The rows of the data-frame represent the individuals in the pedigree and the columns are:
  1. `FamID`- the family identity number.
  2. `ID`- the individual identity number.
  3. `sex`- sex of the individual such that `sex` = 0 for males, and `sex` = 1 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, such that `affected` = `TRUE` if the individual has developed disease and `affected` = `FALSE` otherwise.
  7. `DA1`- the cRV status of the paternally inherited allele, such that `DA1` = 1 if the cRV is inherited and 0 otherwise.
  8. `DA2`- the cRV status of the maternally inherited allele such that `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, such that `proband` = `TRUE` if the individual is the proband and `FALSE` otherwise.
- Let's next draw a pedigree in the year 2020:

```
plot(study_peds[study_peds$FamID == 1, ], ref_year = 2020,
     cex = 0.5)
```





- The seed founder is the individual with ID 1 and he and all his descendants have age data 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. 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, 3, 4 and 20).
- The proband (ID 4) has shading in the lower portion of their symbol.
- Carrier individuals (IDs 1, 3, 4, 7, 8, 9 and 20) have shading in the upper-right portion of their symbol.
- We use `study_peds.text` file to simulate genetic sequence data for the affected individuals in these pedigrees.
- We will discuss about this in our third supplementary material.
- Finally we print the session information to get to know about the R version and the `SimRVPedigree` package information that we used in this analysis 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

```

## Reference

- 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>.