Supplementary Material 3 : Simulate SNV data for affected individuals in pedigrees

Nirodha Epasinghege Dona, Jinko Graham

2021-12-21

Contents

- In this R-markdown document, we discuss the 3rd box (the orange box) of the following flow diagram.

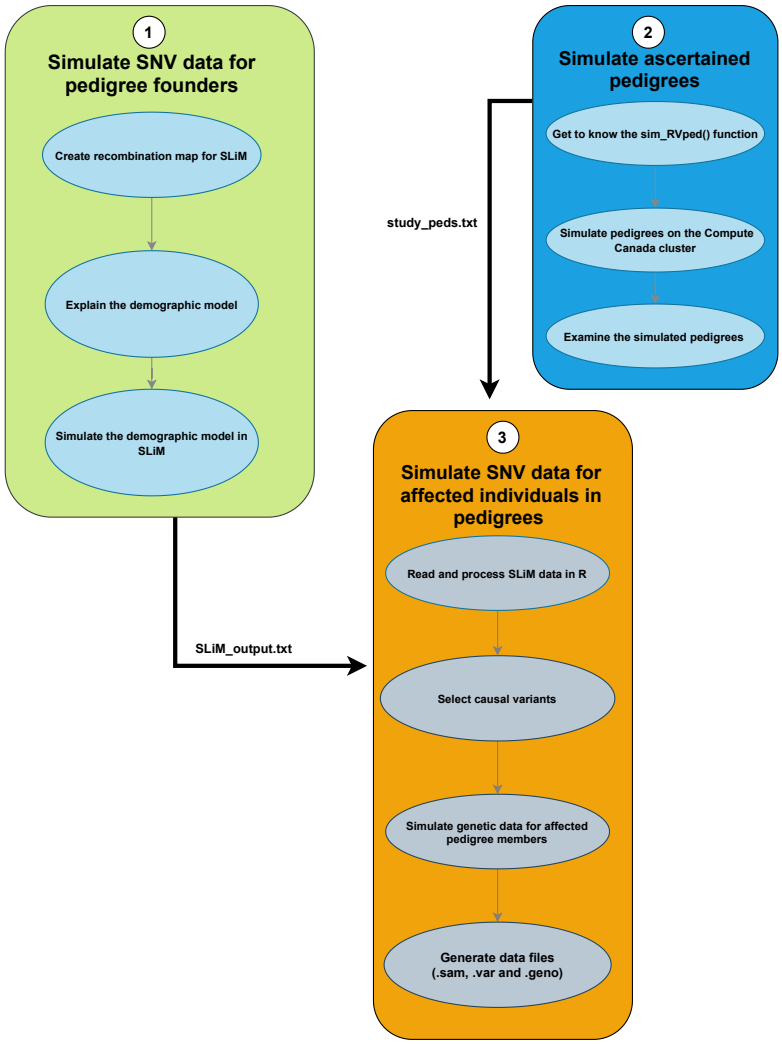

Figure 1: Work-flow for simulating the exome-sequencing data for ascertained pedigrees.

1 Read and process SLiM data in R

- Our final goal of the study is to simulate genetic sequences data for disease-affected family members. For this, we use `SimRVSequences` (Nieuwoudt, Brooks-Wilson, and Graham 2020) R package and the package uses the gene drop method to simulate genetic sequences. For that, it is required to supply sparse matrices of haplotypes of the individuals.
- However, due to the large size of data we can not read all of them into a single sparse matrix. Therefore we break them chromosome-by-chromosome.

- This is the central problem that we address in this section. Let's discuss how we achieve this goal step by step.
- Note: Make sure to load the `Matrix` package before running this R markdown document because it is required throughout.
- First, we read `SLiM_output.txt` (the SLiM output) to R.

```
# Read the text file to R.
# Note: Change the path for the file as necessary.
exData <- readLines("D:/SFU_Vault/SLiM_Output/SLiM_output.txt")
```

- The SLiM simulation output is saved as a text file of size approximately 6 GB.
- The file takes approximately 1 minute to load to R on a Windows OS with an i7-8550U @ 1.8GHz,16GB of RAM.
- This SLiM output is discussed in Supplementary Material 1-A: Simulate SNV data for pedigree founders.
- Note: We use the internal functions (non-exported) in the `SimRVSequences` R package (Nieuwoudt, Brooks-Wilson, and Graham 2020). We print the session information to get to know about the R version and `SimRVSequences` R package details at the end of the document.
- After we read the SLiM output into R, first we extract single-nucleotide variants (SNVs) and then select rare SNVs based on their population derived (mutated) allele frequencies. In the following subsection we explain how we achieve this task.

1.1 Extract and select the rare variants.

- First, we get the line numbers of the mutation and genome header sections in the SLiM output. The following code chunk explains how we get the line number.

```
# Find heading location (i.e. file line number) for mutations.
MutHead <- which(exData == "Mutations:")

# Find heading location (i.e. file line number) for genomes.
GenHead <- which(exData == "Genomes:")
```

- We create a data frame to store all the SNV data as follows.

```
# Extract mutation data from SLiM's Mutation output.
# Only retaining the tempID, position, selection coefficients and count
# of each mutation.
MutOut <- do.call(rbind, strsplit(exData[(MutHead + 1):(GenHead - 1)],
                                  split = " ", fixed = TRUE))

MutData <- data.frame(tempID = as.numeric(MutOut[, 1]),
                      type = MutOut[, 3],
                      position = as.numeric(MutOut[, 4]),
                      selCoef = as.numeric(MutOut[, 5]),
                      count = as.numeric(MutOut[, 9]),
                      stringsAsFactors = TRUE)
head(MutData)
```

```
##   tempID type position selCoef count
## 1   7229   m2 51287555       0 60626
## 2  13218   m2 39812003       0  9536
## 3   5202   m2 36490340       0 64099
## 4  25103   m2 41991968       0  3732
## 5  14264   m2 54793604       0 46668
```

```
## 6   4333   m2 30267920       0 79908
```

- The data frame, `MutData`, contains all the SNVs generated from the SLiM model. The rows of this data frame represent the SNVs and the columns represent the characteristics that we interest about the SNVs. They are:

1. `tempID`- specifies the SLiM-generated identifier number which helps to identify the SNV.
2. `type`- represents the type of the SNV."m_1" and "m_2" catalog the non-synonymous and synonymous SNVs, respectively.
3. `position`- indicates the base-pair position of the SNV on the chromosome.
4. `selCoef`- represents the selection coefficient of the SNV.
5. `count`- specifies the number of copies of the SNV in the population.

- Next we change the default behavior of the starting positions in SLiM. The starting position is always zero in SLiM but R starts its indexing at position one. To accommodate R indexing we add one to the `tempID` and `position` columns in the `MutData` data frame.

```
# Add 1 to temp ID so that we can easily associate mutations to columns.
# By default SLiM's first tempID is 0, not 1.
MutData$tempID <- MutData$tempID + 1

# First position in slim is 0, not 1
MutData$position <- MutData$position + 1
```

- Then we calculate the population derived allele frequencies of the SNVs. We divide the number of copies of the SNV in the population (the `count` column in the `MutData`) by the total number of sequences in the population.

```
# Get the population count of sequences.
popCount <- as.numeric(unlist(strsplit(exData[1], split = " ",
                                       fixed = TRUE))[4])
# Calculate the population derived allele frequency.
# Divide the allele count by the population size.
MutData$afreq <- MutData$count/(popCount)

# Order Mutation data set by tempID, so that (later) we can order
# the mutations on each haplotype by their genomic position.
MutData <- MutData[order(MutData$tempID), ]
```

- After calculating the population derived allele frequencies, we keep the SNVs which are rare in the population. For this, we set a threshold value for the population minor allele frequency (maf).
- To select only the rare SNVs, we assign a colID to each based on this threshold value. The SNVs with derived allele frequency in between maf and 1 - maf are assigned to column ID 0; i.e. they are thrown away. The SNVs with sufficiently small minor allele frequency are assigned increasing colIDs.

```
# Create a threshold value
maf <- 0.01
keep_SNVs <- (MutData$afreq <= maf | MutData$afreq >= (1 - maf))

# Variants with derived allele in between minor allele frequency
# and 1 - minor allele frequency are assigned column ID 0, i.e. thrown away.
# Variants with sufficiently small minor allele frequency
# are assigned increasing colIDs.

MutData$colID <- cumsum(keep_SNVs)*(keep_SNVs)
```

- We keep SNVs with colIDs greater than 0 as the rare variants. We create a new data frame, `RareMutData`, to represent only the rare SNVs in the population.

```
# Using the identified colID, create data frame of rare mutations only.
RareMutData <- MutData[MutData$colID > 0, ]
```

- Our next aim is to identify the chromosome on which each SNV resides. This will be needed when we loop through chromosomes to separate data chromosome-by-chromosome.
- We use the recombination map from the SLiM simulation to identify the exon positions in chromosomes. To get this recombination map, we use the `create_slimMap()` function (Nieuwoudt, Brooks-Wilson, and Graham 2020) in the `SimRVSequences` R package.
- Then we use the `reMap_mutations()` internal function (Nieuwoudt, Brooks-Wilson, and Graham 2020) in the `SimRVSequences` package to identify the chromosome of each SNV.

```
# Create recombination map for exon-only data using
# the hg_exons dataset.(From SimRVSequences.)
recomb_map <- SimRVSequences:::create_slimMap(exon_df = hg_exons)

# Use reMap_mutations function to identify the chromosome
# number on which each SNV resides.
RareMutData <- SimRVSequences:::reMap_mutations(mutationDF = RareMutData,
                                    recomb_map)
```

- When we call the internal function, `reMap_mutation`, it adds a new column,`chr`, to `RareMutData`.
- Then our next aim is to extract genomes from the SLiM output and this is covered in next subsection.

1.2 Extract genomes with rare SNVs.

- First we get the line number of the genome header section in the SLiM output.

```
# Find heading location (i.e. file line number) for genomes.
GenHead <- which(exData == "Genomes:")
```

- In the genomes section of the SLiM output, each line represents a haplotype and the columns represent a SLiM-generated identifier number to identify the SNVs.
- We use the `extract_tempIDs()` internal function in the `SimRVSequences` package to select the rare variants from haplotypes.

```
# Determine future row and column position of each mutation
# listed in genomes.

RareGenomes <- lapply(1:(popCount), function(x){
    SimRVSequences:::extract_tempIDs(mutString = exData[GenHead + x],
                    rarePos = MutData$colID)
  })
```

- Now we have both rare SNVs (`RareMutdata`) and the haplotypes (`RareGenomes`) with these rare SNVs in the population from which the pedigree founders are drawn.
- Then our final goal in this section is to separate these data chromosome-by-chromosome. The next subsection explains this step.

1.3 Process data chromosome-by-chromosome.

- This simulation is computer intensive. It takes approximately 16 hours to run on a Windows OS with an i7-8550U @ 1.8GHz,16GB because of the large number of mutations and the genomes in our SLiM

output. Therefore, we do not run the code chunk below. To knit this document we would recommend to load the `Chromwide.Rdata` file which can be found in Zenodo repository.

- To read data chromosome-by-chromosome, we loop over the number of chromosomes. The looping variable, k, varies from 1 to 22. We use the `foreach()` function to parallelize the R code.
- First, we create the haplotypes matrix for the corresponding chromosome using the `sparseMatrix()` function in the `Matrix` package. The haplotypes matrix is a sparse matrix of class dgCMatrix.
- In the haplotypes matrix, columns represent the rare SNVs that lie on the corresponding chromosome and rows represent the haplotypes in the population.
- we get the SNV ids that lie on the kth chromosome from the `RareMutData` R object and match them to the column IDs of the haplotypes in the `RareGenome` R object.
- The number of rows in the haplotypes matrix for the kth chromosome is the length of the `RareGenomes` R object which is 107,752 haplotypes in the population.
- We order both the haplotypes matrix and the SNV data frame according to the base-pair position of the SNV. Also, we name the columns of the SNV data frame accordingly.
- The following code chunk explains these steps.

```
#------------------#
# Get the results chromosome-by-chromosome #
#------------------#

# Load required libraries to parallel the code
library(foreach)
library(doParallel)

# Get unique chromosome IDs.
chrID <- unique(RareMutData$chrom)

# Create empty lists to save the results
chrby_haplotype  <- list()
chrby_SNVs <- list()
output <- list()

# Since we have a large number of SNVs in each chromosome,
# we parallelize the function to speed up the simulation time.
# Make the clusters.
cl <- makeCluster(detectCores() - 1)
# Register the clusters.
registerDoParallel(cl)

# Create a foreach loop.
out <- foreach(k= 1:length(chrID),
               .packages = c("Matrix", "tidyverse", "data.table",
                             "SimRVSequences"),
               .multicombine = TRUE)%dopar%{
    #-----------#
    # Genotypes #
    #-----------#
    # Get the column positions of the sparse matrix for the kth chromosome.
    # We use the jpos output that contains the
    # data from the genome section of the SLiM output.
    jpos_chr <- lapply(RareGenomes,function(x){
```

```
                    x[RareMutData[RareMutData$colID
                                  %in% x, ]$chrom == chrID[k]]})

    # Get the rows of the sparse matrix for the kth chromosome.
    ipos_chr <- lapply(1:length(jpos_chr), function(x){
                      rep(x, length(jpos_chr[[x]]))})

    # Create sparse matrix containing SNVs (columns)
    # for each genome (row).
    GenoData <- sparseMatrix(i = unlist(ipos_chr),
                             j = unlist(jpos_chr),
                             x = rep(1, length(unlist(jpos_chr))))

    GenoData <- GenoData[, -which(colSums(GenoData) == 0)]

    #-----------#
    # SNVs #
    #-----------#

    # Identify the SNV matrix for each chromosome.
    Mute_uni <- unlist(jpos_chr)
    RareSNVs <- RareMutData[RareMutData$colID %in% Mute_uni, ]

    # Order by genomic position of rare SNV.
    GenoData <- GenoData[, order(RareSNVs$position)]
    RareSNVs <- RareSNVs[order(RareSNVs$position), ]
    RareSNVs$colID <- 1:nrow(RareSNVs)

    # Remove the old tempID.
    RareSNVs <-  RareSNVs[, -1]

    # Change the row names and column names of SNV data frame.
    RareMutData_new <- RareSNVs
    row.names(RareMutData_new) = NULL

    # Create unique SNV names.
    RareMutData_new$SNV <- make.unique(paste0(RareMutData_new$chrom,
                                              sep = "_",
                                              RareMutData_new$position))

    # Reduce RareMutData, to the columns we actually need.
    RareMutData_new <- RareMutData_new[, c("colID", "chrom", "position",
                                           "afreq", "SNV", "type",
                                           "selCoef")]
    # Store the SNVs and haplotypes chromosome-by-chromosome.
    output[[k]] <- list(Haplotypes = GenoData,
                        Mutations = RareMutData_new)

              }

stopCluster(cl)

# Save the result
```

```
save(out, file = "Chromwide.Rdata")
```

1.4 Identify the pathway SNVs

- We then identify the SNVs which lie on the pathway of interest in the study. We use the `identify_pathwaySNVs()` function in the `SimRVSequences` package. We supply the apoptosis sub-pathway centered about the TNFSF10 gene in the UCSC Genome Browser's Gene Interaction Tool as discussed in Nieuwoudt, Brooks-Wilson, and Graham (2020). The data in this sub-pathway are contained in the `hg_apopPath` data set in `SimRVSequences` R package.
- The following code chunk explains the above step.

```
# Load the output generated from the previous code chunk.
load("Chromwide.Rdata")

#----------------------#
# Identify Pathway SNVs #
#----------------------#
pathway_out <- lapply(out, function(x){
  RareMutData_pathway = SimRVSequences:::identify_pathwaySNVs(markerDF =
                            x$Mutations, pathwayDF = hg_apopPath )})
```

- The final outcome of this function is adding an additional column to the SNV data frame label as `pathwaySNV`. This column identifies SNVs lie on the pathway as `TRUE`.
- Finally, we combine all haplotypes matrices and mutation data frames separately for each chromosome.

```
# Get the final output using a for loop.
# Loop over 1:22, i.e, number of chromosomes.
# There are 22 lists and each list contains another two lists.
# The first list is the haplotypes matrix and
# second list is the mutation dataframe, corresponding to each chromosome.
# The format of the output is discussed in section 1.5.

slim_out <- lapply(1:22, function(x){list(Haplotypes = out[[x]]$Haplotypes,
                                          Mutations = pathway_out[[x]])})
```

- We discuss the format of the chromosome-by-chromosome output in next subsection.

1.5 Format of chromosome-by-chromosome data.

- Below we use the `str` function to display the structure of the first element of the list, for chromosome 1.

```
# Get the structure of each list elements of output.
str(slim_out[[1]])
```

```
## List of 2
##  $ Haplotypes:Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. ..@ i       : int [1:8343173] 86549 579 1814 3424 4089 4909 5912 6565 7663 8422 ...
##   .. ..@ p       : int [1:84665] 0 1 144 146 189 295 349 362 366 367 ...
##   .. ..@ Dim     : int [1:2] 107752 84664
##   .. ..@ Dimnames:List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : NULL
##   .. ..@ x       : num [1:8343173] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..@ factors : list()
##  $ Mutations :'data.frame':  84664 obs. of  8 variables:
```

```
##   ..$ colID     : int [1:84664] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ chrom     : int [1:84664] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ position  : num [1:84664] 11951 11975 12224 12626 13231 ...
##   ..$ afreq     : num [1:84664] 9.28e-06 1.33e-03 1.86e-05 3.99e-04 9.84e-04 ...
##   ..$ SNV       : chr [1:84664] "1_11951" "1_11975" "1_12224" "1_12626" ...
##   ..$ type      : Factor w/ 2 levels "m1","m2": 1 1 1 1 2 2 1 1 1 1 ...
##   ..$ selCoef   : num [1:84664] -3.64e-03 -1.33e-06 -2.45e-02 -4.53e-03 0.00 ...
##   ..$ pathwaySNV: logi [1:84664] FALSE FALSE FALSE FALSE FALSE FALSE ...
```

- The `slim_out` object is a list of 22 elements. Each list elements corresponds to a chromosome and contains two items, a haplotypes matrix and a mutation data frame.
- The haplotypes matrix represents the founder haplotypes of the American Admixed population, which is required by the `SimRVSequences` R package to simulate sequence data for individuals in pedigrees.
- The second element is the mutation data frame which represents the mutations that reside on chromosome 1.
- The dimensions of these elements are as follows.

```
# dimensions of the list element 1
dim(slim_out[[1]]$Haplotypes)
```

```
## [1] 107752  84664
```

```
dim(slim_out[[1]]$Mutations)
```

```
## [1] 84664     8
```

- The number of columns in haplotypes matrix is equal to the number of rows in the mutation data frame.
- There are 84,664 mutations that lie on the first chromosome.
- Let's print the first ten rows and columns in the haplotypes matrix of the first chromosome to visualize the format of the output.

```
# The first item (haplotypes) of the first element (chromosome 1) of SLiM output
slim_out[[1]]$Haplotypes[1:10, 1:10]
```

```
## 10 x 10 sparse Matrix of class "dgCMatrix"
##
##  [1,] . . . . . . . . . .
##  [2,] . . . . . . . . . .
##  [3,] . . . . . . . . . .
##  [4,] . . . . . . . . . .
##  [5,] . . . . . . . . . .
##  [6,] . . . . . . . . . .
##  [7,] . . . . . . . . . .
##  [8,] . . . . . . . . . .
##  [9,] . . . . . . . . . .
## [10,] . . . . . . . . . .
```

- The haplotypes matrix of founders is a sparse matrix of class dgCMatrix from the `Matrix` package.
- In this haplotypes matrix, columns represent the single-nucleotide variants (SNVs) and rows represent the individual's exome.
- The entry "1" in this matrix represents the derived (mutated) allele and the "." represents the ancestral allele.
- Now we print the first ten rows and columns in the mutation data frame.

```
# The second item (mutations) of the first element (chromosome 1) of slim output
slim_out[[1]]$Mutations[1:10, ]
```

```
##    colID chrom position        afreq     SNV type       selCoef pathwaySNV
## 1      1     1    11951 9.280570e-06 1_11951   m1 -3.638366e-03      FALSE
## 2      2     1    11975 1.327122e-03 1_11975   m1 -1.328507e-06      FALSE
## 3      3     1    12224 1.856114e-05 1_12224   m1 -2.454038e-02      FALSE
## 4      4     1    12626 3.990645e-04 1_12626   m1 -4.526557e-03      FALSE
## 5      5     1    13231 9.837404e-04 1_13231   m2  0.000000e+00      FALSE
## 6      6     1    13411 5.011508e-04 1_13411   m2  0.000000e+00      FALSE
## 7      7     1    14129 1.206474e-04 1_14129   m1 -1.906579e-02      FALSE
## 8      8     1    14148 3.712228e-05 1_14148   m1 -2.180460e-10      FALSE
## 9      9     1    14180 9.280570e-06 1_14180   m1 -3.852785e-02      FALSE
## 10    10     1    14228 2.784171e-05 1_14228   m1 -8.308619e-02      FALSE
```

- The rows of the mutation data frame represent the SNVs.
- The columns represents each SNV's characteristics.
- The variable `colID` links the rows in the mutation data frame to the columns of haplotypes matrix, `chrom` is the chromosome the SNV lies in, `position` is the position of the SNV in base pairs, `afreq` is the population derived allele frequency of the SNV, `SNV` is an unique character identifier for the SNV, and `pathwaySNV` identifies whether or not SNVs are located within the apoptosis sub-pathway. (Nieuwoudt, Brooks-Wilson, and Graham (2020) consider the apoptosis sub-pathway centered about the TNFSF10 gene in the UCSC Genome Browser's Gene Interaction Tool.)
- The next task is to select the causal rare variants (cRV).

2 Select causal variants

- We follow Nieuwoudt, Brooks-Wilson, and Graham (2020), which considers SNVs in genes on an apoptosis sub-pathway as candidates for the causal variants.
- We set the cumulative probability of a sequence in the population carrying a risk variant to be less than or equal to 0.001.
- We select causal variants from singletons in the population, in proportion to their absolute selection coefficients. In the SLiM simulation, selection coefficients are always less than or equal to zero (i.e. variants are deleterious or selectively neutral).
- We create a function, `select_cRV()`, for this purpose. This function has two required arguments. They are:

1. `chrm_by_out` - represents the chromosome-by-chromosome results in the `slim_out` R object.
2. `cumAF` - specifies the cumulative probability of a sequence carrying a risk variant in the population. Here, we set `cumAF` to be 0.001 (Nieuwoudt, Brooks-Wilson, and Graham 2020).

- After specifying the arguments above, `select_cRV()` selects all the SNV data frames and the haplotypes matrices, separately from the `slim_out` R object.
- Then we identify the singleton SNVs which lie on the specified pathway (apoptosis sub-pathway) from haplotypes matrices. For this, we get the column sum of the haplotypes matrices and select the columns with column sum equal to one.
- We then assign a weight to each selected singleton according to the value of its selection coefficient. The weights are calculated as:

$$w_i = \frac{|S_i|}{|\sum_i^N S_i|}$$

where w_i is weight of i^{th} SNV; S_i is the selection coefficient of the i^{th} SNV and N is the total number of singletons in the specified pathway.

- We consider these weights as the sampling probabilities for drawing cRVs from the pool of singleton SNVs.
- Then the function,`select_cRV`, randomly samples cRVs until their cumulative population derived allele frequency is less than 0.001.
- The `select_cRV()` function is shown below.

```
select_cRV <- function(chrm_by_out, cumAF){

  # Select all the mutation data frames in the slim_out object.
  SNV_df <- lapply( chrm_by_out, `[[`, 'Mutations')

  # Select all haplotypes matrices in the slim_out object.
  haplo <- lapply( chrm_by_out, `[[`, 'Haplotypes')

  # Get the ColIDs of singletons under each chromosome.
  sing_colID <- lapply(lapply(haplo, function(x){
    which(colSums(x) == 1)}), unlist)

  # Select only singletons in the SNV data frames.
  singletons <- lapply(1:22, function(x){
    SNV_df[[x]][SNV_df[[x]]$colID %in% sing_colID[[x]], ]})

  # Combine all the 22 data frames (contains only singletons)
  # into one data frame.
  SNV_singletons <- do.call(rbind, singletons)

  # Assign weights to each marker based on their selection coefficient values.
  SNV_singletons$weight <-
    abs(SNV_singletons$selCoef)/abs(sum(SNV_singletons$selCoef))

  # Select SNVs(singletons) which lie on our pathway of interest.
  SNV_pathway <- SNV_singletons[SNV_singletons$pathwaySNV == TRUE, ]

  # Initialize vectors to store the cumulative sum of allele frequencies and
  # cRVs.
  cum <- 0
  cSNVs <- c()

  # The loop runs till the cumulative sum of the allele frequency
  # is less than or equal to CumAF (0.001).
  while (cum <= cumAF) {
    # Select a SNV proportional to its weights.
    selected_cRVs <- sample(SNV_pathway$SNV, 1,
                            prob = c(SNV_pathway$weight))
    # Get the allele frequency of the selected SNV.
    af <-  SNV_pathway[SNV_pathway$SNV == selected_cRVs, 4]
    # Update the cumulative sum of the allele frequencies of causal SNVs.
```

```
    cum <- cum + af
    # Remove the selected SNV from the SNV data frame.
    # If not we may get this same SNV again.
    SNV_pathway <- SNV_pathway[-(which(SNV_pathway$SNV == selected_cRVs)),
                               ]
    # Save the selected cRVs in a vector
    cSNVs <- c(cSNVs, selected_cRVs)
  }
  # Create a new variable in our SNV data frame to represent
  # whether a SNV is a cRV or not.
  SNV_combine <- do.call("rbind", SNV_df)
  SNV_combine$is_CRV <- SNV_combine$SNV %in% cSNVs

  return(SNV_combine)
}
```

- Let's view how to use the `select_cRV` function.

```
# Set a seed value.
set.seed(1987)

# Run the function.
cRV_data <- select_cRV(chrm_by_out = slim_out , cumAF = 0.001)

# Display the function output.
cRV_data[1:10, ]
```

```
##    colID chrom position        afreq     SNV type       selCoef pathwaySNV
## 1      1     1    11951 9.280570e-06 1_11951   m1 -3.638366e-03      FALSE
## 2      2     1    11975 1.327122e-03 1_11975   m1 -1.328507e-06      FALSE
## 3      3     1    12224 1.856114e-05 1_12224   m1 -2.454038e-02      FALSE
## 4      4     1    12626 3.990645e-04 1_12626   m1 -4.526557e-03      FALSE
## 5      5     1    13231 9.837404e-04 1_13231   m2  0.000000e+00      FALSE
## 6      6     1    13411 5.011508e-04 1_13411   m2  0.000000e+00      FALSE
## 7      7     1    14129 1.206474e-04 1_14129   m1 -1.906579e-02      FALSE
## 8      8     1    14148 3.712228e-05 1_14148   m1 -2.180460e-10      FALSE
## 9      9     1    14180 9.280570e-06 1_14180   m1 -3.852785e-02      FALSE
## 10    10     1    14228 2.784171e-05 1_14228   m1 -8.308619e-02      FALSE
##    is_CRV
## 1   FALSE
## 2   FALSE
## 3   FALSE
## 4   FALSE
## 5   FALSE
## 6   FALSE
## 7   FALSE
## 8   FALSE
## 9   FALSE
## 10  FALSE
```

- The output of the `select_cRV()` function is a mutation data frame with an additional column, `is_CRV`. This column catalogs which SNVs are selected as the candidates for the causal variants.
- Then we print the number of cRVs in the population, their cumulative allele frequency and in which chromosome they reside as follows.

```r
# Display the number of selected cRVs.
length(which(cRV_data$is_CRV == TRUE))
```

```
## [1] 108
```

```r
# Print the cumulative allele frequency.
round(sum(cRV_data$afreq[which(cRV_data$is_CRV==TRUE)]), 5)
```

```
## [1] 0.001
```

```r
# Display the number of cRVs that are selected under each chromosome.
table(cRV_data[cRV_data$is_CRV ==  TRUE, ]$chrom)
```

```
##
##  1  2  3  4  5  6  7  8 10 11 16 17 18 19 21 22
##  1 21  5  3  8  8  5 11  8  4  4  4 13  1  7  5
```

- According to the above outputs, there are 108 SNVs sampled as cRVs. Their cumulative population derived allele frequency is 0.001.
- The table output explains the number of cRVs that are selected from each chromosome. As an example, only 1 cRV resides in chromosome 1; 21 cRVs reside in chromosome 2 and so forth.
- Next we simulate sequence data for the affected individuals in the 150 ascertained pedigrees. The next section illustrates this task.

3 Simulate genetic data for affected pedigree members

- To simulate genetic sequence data for affected individuals in our ascertained pedigrees, we need the following:

1. Single-nucleotide variant (SNV) data for unrelated pedigree founders.
2. A sample of ascertained pedigrees.

- The first and second supplementary materials (Supplementary Material 1-A: Simulate SNV data for pedigree founders and Supplementary Material 2: Simulate ascertained pedigrees) go over how to get these.
- In the `SimRVSequences` R package, the function `sim_RVstudy()` is designed to simulate genetic sequence data. However, to get the chromosome-by-chromosome results, we modified the `sim_RVstudy()` function.
- The following sub section explains the modifications that we add to simulate genetic sequence data chromosome-by-chromosome.

3.1 Add modifications to simulate data chromosome-by-chromosome

- The first modification is we add an additional argument to the `sim_RVstudy()` function. The new argument, `fam_RVs`, gives the identities of the familial cRVs. Then inside the function, we check whether the familial cRV lies on the corresponding chromosome or not. If the familial cRV resides on the chromosome, then we use the conditional gene drop to simulate sequences as in Nieuwoudt, Brooks-Wilson, and Graham (2020). If not, we simulate them by assuming that family is a sporadic family and genetic materials from parent to offspring transmit according to the Mendelian law.
- The `sim_RVstudy()` function with these new changes can be found in Appendix file. In the code chunk, line number 1328 to 1338, you can find the changes added to the function.
- A required argument of the `sim_RVstudy()` function is `SNV_data`, which is an object of class `SNVdata`. We use the `SNVdata()` function in the `SimRVSequences` R package to convert haplotypes matrices and SNV dataframes into an object of class `SNVdata`. To align this function with our changes, we

modified the `SNVdata()` and the `check_SNV_map()` functions. The `SNVdata()` function, is essentially the same as in the `SimRVSequences` R package except we have changed the label of a column in the SNV dataframe from `marker` to `SNV`. We add the same change to the the `check_SNV_map()` function as well. However, `check_SNV_map()` does not work if `is_CRV` column is `FALSE` for all SNVs in the mutation dataframe. This can happen if there is no cRV on the given chromosome. Therefore, we remove the step where the `check_SNV_map()` function checks whether `is_CRV` column is FALSE for all the SNVs in the mutation dataframe. The modified versions of `SNVdata()` and `check_SNV_map()` functions can be found in Appendix file. Note: We rename these two functions as, `SNVdata_new()` and `check_SNV_map_new()`.

- The above are the modifications that we added to the functions in the `SimRVSequences` package to simulate the genetic sequences of affected members in pedigrees.
- We name the new function as `sim_RVstudy_new()` to avoid the conflicts between two functions. Hereafter we use the `sim_RVstudy_new` name instead `sim_RVstudy`.
- To simulate the genetic sequence data chromosome-by-chromosome, for affected members in families, we loop over the number of chromosomes and for each chromosome we apply the `simRV_study_new()` function.
- The next sub section discusses how we set values for the arguments in the `sim_RVstudy_new()` function.

3.2 Set the arguments in the `sim_RVstudy_new()` function.

- The required arguments for the `sim_RVstudy_new()` function are as follows.

(1). `fam_RVs`- the causal cRVs.

- We sample familial cRVs based on their population derived allele frequencies.
- The following R code is used for this task.

```
# Load all 150 pedigrees. This will be explained in
# the second argument again.
study_peds <- read.table("study_peds.txt", header=TRUE, sep= " ")

# Collect list of FamIDs.
FamIDs <- unique(study_peds$FamID)

# Set the sampling probabilities for causal rare variants.
# When the derived allele frequencies are provided, we sample causal rare
# variants according to their allele frequency.
sample_prob <- cRV_data$afreq[cRV_data$is_CRV]/
    sum(cRV_data$afreq[cRV_data$is_CRV])


set.seed(1987)
# Sample the familial cRV from the pool of potential cRVs with replacement.
familial_RVs <- sample(x = cRV_data$SNV[cRV_data$is_CRV],
                  size = length(FamIDs),
                  prob = sample_prob,
                  replace = TRUE)
# Display first five candidates for familial cRVs.
familial_RVs[1:5]
```

```
## [1] "2_201170321" "18_63125128" "10_89015222" "6_108683999" "1_155738619"
```

- The final output, `familial_RVs` is a vector of length 150 that contains familial cRVs. We use `Fam_RVs` for the first argument, in the `sim_RVstudy_new()` function.

- The next two required arguments are the same as in the `sim_RVstudy()` function.

(2). `ped_files`- represents the ascertained pedigrees. We read `study_peds.text` file (we discussed about this in Supplementary Material 2: Simulate ascertained pedigrees) and set the resulting R object as the second argument.

```
study_peds <- read.table("study_peds.txt", header= TRUE, sep= " ")
```

(3). `SNV_data`- represents an object of class `SNVdata`.

- We use the haplotypes matrices and SNV dataframes of all 22 chromosomes and create all of them as SNVdata objects. Since we loop over chromosome number, we supply all the haplotypes matrices and SNV dataframes to this argument as an object in class SNVdata. When loop runs, it selects haplotypes matrix and SNV dataframe corresponding to the given chromosome.
- Note:You need to load the modified versions of the `SNVdata_new()` and `check_SNV_map_new()` functions in the Appendix before running the next code chunk.
- The following code explains how we apply the `SNVdata_new()` function.

```
# Apply the SNVdata_new function to 22 haplotypes matrices and
# SNV dataframes
chrom_data <- lapply(slim_out, function(x){
  SNV_data = SNVdata_new(Haplotypes = x$Haplotypes,
                         Mutations = x$Mutations)})
```

- We supply `chrom_data` as the third argument, `SNV_data`, in the `sim_RVstudy_new()` function. This `chrom_data` object contains all the SNVs and haplotypes matrices for each chromosome.
- The above three arguments are the required arguments for the function `sim_RVstudy_new()` and the remaining optional arguments are set according to Nieuwoudt, Brooks-Wilson, and Graham (2020).
- Among these optional arguments, we use the default setting `affected_only = TRUE` because our interest is to simulate sequence data for the disease-affected members and the family members that connect them along a line of descent. The reason for this is these studies sequence disease-affected members because they are more likely to carrying a causal variant for the disease.
- Also, `remove_wild = TRUE` is set to the default value to remove any SNV which is not carried by any member of the study. This will help to reduce the size of our simulated sequence data (Nieuwoudt, Brooks-Wilson, and Graham 2020).
- To run the `sim_RVstudy_new()` function, it is required to run internal functions (non-exported) in the `SimRVSequences` R package. These functions are in Appendix.
- Then we obtain the chromosome-by-chromosome genetic sequences for disease-affected members in families as follows. We loop over the number of chromosomes and for each chromosome we apply the `simRV_study_new()` function.
- Note: Before running the following code chunk, you need to copy the `simRV_study_new()` function and specific non-exported functions (see Appendix) in the `SimRVSequences` R package to your work space. These copied-in functions are the same as in the `SimRVsequences` R package except they use the `Matrix` package's `which()` function instead of the base R `which()` function, unlike in the original `SimRVSequences`.

```
# Simulate SNV data for affected family members
set.seed(1987)

study_seq <- lapply(1:22, function(x){sim_RVstudy_new(fam_RVs = familial_RVs,
                                                      ped_files = study_peds,
                                                      SNV_data = chrom_data[[x]]
                                                      )})
```

- Note: To simulate sequence data for disease-affected members in family takes approximately 6 minutes on a Windows OS with an i7-8550U @ 1.8GHz,16GB of RAM.
- The table 1 illustrates the sequence simulation times that required for different chromosomes (chromosome 1, 2, 8 and 9) on a Windows OS with an i7-8550U @ 1.8GHz,16GB of RAM.

Table 1: Simulation time for selected chromosomes.

Chromosome	No.of SNVs	No.of causal SNVs	Time in seconds
1	84664	1	36.23
2	60995	21	53.63
8	31396	11	22.35
9	34248	0	19.57

- Table 1 indicates that chromosome 1 sequence data simulation time is less than chromosome 2, despite having more SNVs. However, chromosome 1 has fewer causal SNVs than chromosome 2. By contrast, chromosome 8 has more causal SNVs than chromosome 1 yet takes less time because it has fewer SNVs overall. This suggests that the simulation time depends on both the overall number of SNVs and the number of causal SNVs on chromosome.
- The `sim_RVstudy_new()` function returns the same set of outputs as in the `sim_RVstudy()` function. The following sub section discusses about these outputs.

3.3 Discuss the `sim_RVstudy_new()` function output.

- The output `study_seq` is a list that contains 22 list elements.
- Let's discuss the output format of the first chromosome. (The output format for the other chromosomes is the same.)
- There are four items (lists) contained in each list element of the `study_seq`. They are:

(1). The `ped_files` data frame.

- This data frame represents the details about the individuals in the pedigrees. When we set `affected_only = TRUE`, the results contain only the affected individuals and the individuals who connect them along a line of descent.

```
# View the first 4 individuals in ped files data frame (in first chromosome).
head(study_seq[[1]]$ped_files, n = 4)
```

```
##    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
## 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
## 25     1 20   0    19     8     TRUE   0   1    1957    1997    2016      TRUE
##    Gen proband
## 1    1   FALSE
## 3    2   FALSE
## 5    2    TRUE
## 25   4   FALSE
```

(2). The sparse matrix `ped_haplos`.

- This contains simulated haplotypes data for the disease-affected individuals and the individuals who connect them along a line of descent in the ascertained pedigrees (all the individuals in `ped_files`).

```
# View the first 10 haplotypes (first chromosome) in ped_haplos
# of first 10 individuals.
```

```
study_seq[[1]]$ped_haplos[1:10, 1:10]
```

```
## 10 x 10 sparse Matrix of class "dgCMatrix"
##
##  [1,] . . . . . . . . . .
##  [2,] . . . . . . . . . .
##  [3,] . . . . . . . . . .
##  [4,] . . . . . . . . . .
##  [5,] . . . . . . . . . .
##  [6,] . . . . . . . . . .
##  [7,] . . . . . . . . . .
##  [8,] . . . . . . . . . .
##  [9,] . . . . . . . . . .
## [10,] . . . . . . . . . .
```

- In this haplotypes matrix, columns represent the single-nucleotide variants (SNVs) and rows represent the individual's exome.
- The entry "1" in this matrix represents the derived (mutated) allele and the "." represents the ancestral allele.

(3). The `SNV_map` data frame.

- Since we set the default value (`TRUE`) for the argument `remove_wild`, this dataframe contains only SNVs that are carried by at least one individual in the family. (If we set it to `FALSE`, the `SNV_map` contains all the SNVs.)

```
# View the first 4 observations in SNV_map

head(study_seq[[1]]$SNV_map, n = 4)
```

```
##   colID chrom position          afreq     SNV type         selCoef pathwaySNV
## 1     1     1    11975 0.001327121538 1_11975   m1 -1.32850710e-06      FALSE
## 2     2     1    13231 0.000983740441 1_13231   m2  0.00000000e+00      FALSE
## 3     3     1    14230 0.001568416364 1_14230   m2  0.00000000e+00      FALSE
## 4     4     1    14231 0.000631078773 1_14231   m1 -1.46789732e-03      FALSE
##   is_CRV
## 1  FALSE
## 2  FALSE
## 3  FALSE
## 4  FALSE
```

- The rows of the dataframe represent the SNVs carried by at least one individual in the family. The columns are the characteristics of the SNVs that we explained in the subsection 1.5.

(4). The `haplo_map` data frame.

- This data frame maps the haplotypes in the `ped_haplos` to the individuals in `ped_files`.

```
# view the observations in haplo_map

head(study_seq[[1]]$haplo_map, n= 18)
```

```
##    FamID ID affected FamCRV
## 1      1  1     TRUE no_CRV
## 2      1  1     TRUE no_CRV
## 3      1  2    FALSE no_CRV
## 4      1  2    FALSE no_CRV
```

```
## 5       1  6    FALSE no_CRV
## 6       1  6    FALSE no_CRV
## 7       1 19    FALSE no_CRV
## 8       1 19    FALSE no_CRV
## 9       1  3     TRUE no_CRV
## 10      1  3     TRUE no_CRV
## 11      1  4     TRUE no_CRV
## 12      1  4     TRUE no_CRV
## 13      1  8    FALSE no_CRV
## 14      1  8    FALSE no_CRV
## 15      1 20     TRUE no_CRV
## 16      1 20     TRUE no_CRV
## 17      2  1     TRUE no_CRV
## 18      2  1     TRUE no_CRV
```

- According to the above output, we can see that, for each individual the genetic material inherited from each parent is stored separately under each chromosome. (Note: The above output is based on chromosome 1.)
- The last column indicates the familial cRV ID. If any family does not have a cRV lie on the selected chromosome, it indicates as `no_CRV`. As an example for family ID 1 does not carry a cRV on chromosome 1.
- The final aim of our study is to deliver simulated data in human-readable flat file formats.
- The next section of this document discusses how we achieve this task.

4 Generate data files

- The structure of the data files is designed similar to PLINK (Weeks 2010) file formats.
- We create three data file formats for each chromosome using the outputs returned by the `sim_RVstudy_new()` function. They are:

(1). `.sam` file, which represents the sample (ascertained pedigree) information.

(2). `.geno` file, which represents the genotype information.

(3). `.var` file, which represents the SNV information.

- Let's discuss how we generate these file formats.

4.1 `.sam` file

- The `.sam` file contains pedigree information of all the disease-affected individuals and the individuals who connect them along a line of descent.
- It will be helpful to print the structure of the output (`study_seq` R object) of the `sim_RVstudy_new()` function that was discussed in the previous subsection,3.3. These outputs are used throughout this section to generate our file formats.
- Let's print the output for chromosome 1.

```
# The structure of the study_seq object.
# This object is a list of length 22.
# We print the first list of this object.
str(study_seq[[1]])
```

```
## List of 4
##  $ ped_files :'data.frame':  1247 obs. of  14 variables:
```

```
##   ..$ FamID    : int [1:1247] 1 1 1 1 1 1 1 1 2 2 ...
##   ..$ ID       : int [1:1247] 1 3 4 20 2 8 19 6 1 3 ...
##   ..$ sex      : int [1:1247] 1 1 0 0 0 1 0 0 1 0 ...
##   ..$ dadID    : int [1:1247] NA 2 2 19 NA 6 NA NA NA 2 ...
##   ..$ momID    : int [1:1247] NA 1 1 8 NA 3 NA NA NA 1 ...
##   ..$ affected : logi [1:1247] TRUE TRUE TRUE TRUE FALSE FALSE ...
##   ..$ DA1      : int [1:1247] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ DA2      : int [1:1247] 1 1 1 1 0 1 0 0 1 1 ...
##   ..$ birthYr  : int [1:1247] 1881 1901 1910 1957 NA 1924 NA NA 1914 1931 ...
##   ..$ onsetYr  : int [1:1247] 1952 1970 2000 1997 NA NA NA NA 1987 1979 ...
##   ..$ deathYr  : int [1:1247] 1955 1981 2002 2016 NA 1957 NA NA 1990 2014 ...
##   ..$ available: logi [1:1247] TRUE TRUE TRUE TRUE FALSE TRUE ...
##   ..$ Gen      : int [1:1247] 1 2 2 4 1 3 3 2 1 2 ...
##   ..$ proband  : logi [1:1247] FALSE FALSE TRUE FALSE FALSE FALSE ...
##  $ ped_haplos:Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. ..@ i       : int [1:193736] 1795 2083 553 558 1173 1178 1013 1017 1799 1807 ...
##   .. ..@ p       : int [1:19526] 0 2 6 8 10 12 19 24 27 32 ...
##   .. ..@ Dim     : int [1:2] 2494 19525
##   .. ..@ Dimnames:List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : NULL
##   .. ..@ x       : num [1:193736] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..@ factors : list()
##  $ haplo_map :'data.frame':  2494 obs. of  4 variables:
##   ..$ FamID   : int [1:2494] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ ID      : int [1:2494] 1 1 2 2 6 6 19 19 3 3 ...
##   ..$ affected: logi [1:2494] TRUE TRUE FALSE FALSE FALSE FALSE ...
##   ..$ FamCRV  : chr [1:2494] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
##  $ SNV_map   :'data.frame':  19525 obs. of  9 variables:
##   ..$ colID     : int [1:19525] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ chrom     : int [1:19525] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ position  : num [1:19525] 11975 13231 14230 14231 14274 ...
##   ..$ afreq     : num [1:19525] 1.33e-03 9.84e-04 1.57e-03 6.31e-04 9.28e-06 ...
##   ..$ SNV       : chr [1:19525] "1_11975" "1_13231" "1_14230" "1_14231" ...
##   ..$ type      : Factor w/ 2 levels "m1","m2": 1 2 2 1 1 1 1 1 1 2 ...
##   ..$ selCoef   : num [1:19525] -1.33e-06 0.00 0.00 -1.47e-03 -2.26e-08 ...
##   ..$ pathwaySNV: logi [1:19525] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ is_CRV    : logi [1:19525] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  - attr(*, "class")= chr [1:2] "famStudy" "list"
```

- To generate the `.sam` file, we create the function, `plink_format_samp()`, and this function has an argument, `peds`, which gives the pedigree information.
- we supply the first list element, `ped_files`, of the `sim_RVstudy_new()` function output as this argument.
- The function selects specific columns of the `ped_files` data frame and adjusts them using a format similar to the `.psam` PLINK file.

```
# Get the information for .psam file

plink_format_samp <- function(peds){

  # Convert sex. In PLINK 1 is male 2 is female.
  # We have 0 s to represent male and 1 for female.
  peds$sex[peds$sex == 1] <- c(2)
  peds$sex[peds$sex == 0] <- c(1)
```

```
  # Affected variable consists logical values.
  # Need to change it as character to assign values to
  # represent the phenotype.
  peds$affected <- as.character(peds$affected)

  # If affected is NA consider it as missing.
  # In PLINK missing is denoted as 0 or -9.
  peds$affected[is.na(peds$affected)] <- c(0)
  # Non-affected is represented as 1 in PLINK.
  peds$affected[peds$affected == "FALSE"] <- c(1)
  # Affected is represented as 2 in PLINK.
  peds$affected[peds$affected == "TRUE"] <- c(2)

  peds$FamID <- as.numeric(peds$FamID)
  peds$affected <- as.numeric(peds$affected)

  # Create the data frame with required columns.
  psam_file <- data.frame(peds$FamID, peds$ID, peds$dadID, peds$momID,
                          peds$sex, peds$affected,
                          peds$birthYr, peds$deathYr, peds$proband)

  colnames(psam_file) <- c("#FID", "IID", "PAT", "MAT", "SEX", "PHENO1",
                           "BIRTHYr", "DEATHYr", "PROBAND")
  return(psam_file)
}
```

- The following code chunk explains how we apply the `plink_format_samp()` function and the output of the function for chromosome 21.

```
# Run the function by supplying chromosome one ped files result
sample_data <- plink_format_samp(study_seq[[21]]$ped_files)

# Print first 6 individuals
head(sample_data, n= 6)
```

```
##   #FID IID PAT MAT SEX PHENO1 BIRTHYr DEATHYr PROBAND
## 1    1   1  NA  NA   2      2    1881    1955   FALSE
## 2    1   3   2   1   2      2    1901    1981   FALSE
## 3    1   4   2   1   1      2    1910    2002    TRUE
## 4    1  20  19   8   1      2    1957    2016   FALSE
## 5    1   2  NA  NA   1      1      NA      NA   FALSE
## 6    1   8   6   3   2      1    1924    1957   FALSE
```

- The rows of the above output represents the disease-affected individuals and the individuals who connect them along a line of descent in all pedigrees.
- The columns of the output can be described as follows:

(1). `FID`- the identification number of the family that the individual belongs to.

(2). `IID`- the individual identification number.

(3). `PAT`- the father's identification number.

(4). `MAT`- the mother's identification number.

(5). `SEX`- the individual's sex. The labelled values 1 and 2 represent male and female, respectively.

(6). `PHENO1`- the disease-affected status. The value 1 represents not affected and 2 represents affected.

(7). `BIRTHYr`- the individual's birth year.

(8). `DEATHYr`- the death year of the individual. If the individual is still alive, the `NA` value is used.

(9). `PROBAND`- the proband of the study. It is a logical value.

- We save the `.sam` file as a text file (`sample_info.txt`) and that can be found in our Zenodo repository.
- The following code chunk represents how we save the sample output as a text file.

```
# Write the file to a text file
write.table(sample_data, "sample_info.txt", row.names=FALSE, quote = FALSE)
```

4.2 .geno file

- The `.geno` file is in gene dosage format which gives the number of copies of a specific allele present in the individual.
- We use the second list element of the `sim_RVstudy_new()` function output, the sparse matrix `ped_haplos`, and convert multi-locus genotypes to gene dosage format.
- We have 22 `.geno` files that represent gene dosage data for each chromosome.
- The following function, `gene_data()`, is used to convert the multi-locus genotypes to gene dosage format. This function has an argument, `geno`, and we supply the sparse matrix `ped_haplos` to this argument.

```
# Get the haplotypes and convert them to genotype format gene dosage

gene_data <- function(geno){

  gene_dosage <- list()
  IDs <- seq(from = 1, to = nrow(geno), by = 2)

  # Get the column sums.
  for(i in 1: length(IDs)){
    gene_dosage[[i]] <- colSums(geno[IDs[i]:(IDs[i] + 1), ])
    genotypes <- do.call(rbind, gene_dosage)

  }
  genotypes <- do.call(rbind, gene_dosage)
  return(genotypes)
}
```

- Let's consider chromosome 21 as an example. To convert multi-locus genotypes from chromosome 21 to gene-dosage format, the `gene_data()` function takes approximately 15 seconds on a Windows OS with an i7-8550U @ 1.8GHz,16GB of RAM.
- We apply the `gene_data()` function to chromosome 21's multi-locus genotypes as follows.

```
# Apply the function to 21 st chromosome
genotype_data <-  gene_data(study_seq[[21]]$ped_haplos)
```

- Let's view the output of the `gene_data()` function.

```
# View the first five rows and columns
genotype_data[1:5, 1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
```

```
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
## [5,]    0    0    0    0    0
```

- The rows of the output represent the affected individuals and the individuals who connect them along a line of descent in all pedigrees.
- The columns represent the SNVs that reside on chromosome 21. Each cell value represents the dosage of the derived allele of an SNV. The cell values can be 0, 1 or 2.
- As an example, the first cell value in the above output is 0, which means that the derived allele of SNV1 is not present in either genome of the first individual. Because these are rare SNVs, we expect most of our cell values to be 0.
- We apply the `gene_data()` function to all chromosomes as follows.

```
# Apply function to all chromosomes
genotype_data <-  lapply(study_seq, function(x){
  result <- gene_data(x$ped_haplos)
  colnames(result) <- x$SNV_map$SNV
  result
})
```

- There are 22 `.geno` files and we save them as text files as follows. They are named as `genotypes_chr_i.txt` where "i" indicates the chromosome number. These text files can be found in the Zenodo repository.

```
# Write the results to 22 text files
for(i in 1:22){
  write.table(genotype_data[[i]],
              paste0("genotypes_chr_",i,".txt"),
              row.names=FALSE, quote = FALSE)
}
```

4.3 `.var` file

- The `.var` file represents the variant information; i.e. the information about the SNVs in the columns of the `.geno` file.
- We create the function `variant_data()` to obtain the SNV data information.
- We supply the fourth list element of the `sim_RVstudy_new()` function output, the `SNV_map` dataframe, as the argument of the `variant_data()` function.
- We select a sub set of relevant characteristics of SNVs and store them in a dataframe with the following function.

```
# Get the variant information to create the .pvar file

variant_data <- function(variant){
  # Chromosome number.
  CHROM <- variant$chrom
  # Position.
  POS <- variant$position
  # Reference allele.
  REF <- rep("A", length(CHROM))
  # Alternate allele.
```

```
  ALT <- rep("T", length(CHROM))
  # Selection coefficient.
  sel_coef <- variant$selCoef
  # Population allele frequency.
  pop_afreq <- variant$afreq
  # Pathway SNV or not.
  pathwaySNV <- variant$pathwaySNV
  # Causal SNV or not.
  C_SNV <- variant$is_CRV
  # label the type as NS and S where NS- non-synonymous and
  # S- Synonymous.
  levels(variant$type) <- c("NS", "S")
  # Type of the SNV
  Type <- variant$type

  # Create the data frame.
  SNV <- data.frame(CHROM, POS, REF, ALT,
                    pop_afreq, sel_coef, pathwaySNV, C_SNV, Type)

  return(SNV)

}
```

- We get `.var` files for each chromosome, separately.
- Let's view the variant data of chromosome 21 as follows.

```
# Run the function on chromosome 21 SNV_map data
variant_info<- variant_data(study_seq[[21]]$SNV_map)

# View the first 4 rows of the resulting data frame
head(variant_info, n = 4)
```

```
##   CHROM     POS REF ALT      pop_afreq        sel_coef pathwaySNV C_SNV Type
## 1    21 5590862   A   T 6.49639914e-05 -0.078572489300      FALSE FALSE   NS
## 2    21 5590947   A   T 2.04172544e-04 -0.000163735545      FALSE FALSE   NS
## 3    21 5591101   A   T 2.13453115e-04  0.000000000000      FALSE FALSE    S
## 4    21 5591550   A   T 8.35251318e-05 -0.006679104180      FALSE FALSE   NS
```

- The rows of the above output catalog the SNVs on chromosome 21 which are carried by at least one study participant.
- The columns catalog the information about these SNVs. They are:

(1). `CHROM`- the chromosome number of the SNV.

(2). `POS`- the position, in base pairs, on the chromosome where the SNV resides.

(3). `REF`- the reference allele for the SNV

(4). `ALT`- the alternate allele for the SNV

(5). `pop_afreq`- the population alternate allele frequency for the SNV.

(6). `sel_coef`- the selection coefficient for the SNV.

(7). `pathwaySNV`- whether or not the SNV lies on our disease pathway.

(8). `C_SNV`- whether or not the SNV is a causal variant.

(9). `Type`- whether the SNV is a synonymous (S) or non-synonymous (NS) SNV.

- We apply the function to all 22 chromosomes and save them to `SNV_map` object. Then we write them to 22 text files as follows.

```
# Apply function to all 22 chromosomes
SNV_map <- lapply(study_seq, function(x){
  variant_data(x$SNV_map)
})


# Write the results separately to text files
for(i in 1:22){
  write.table(SNV_map[[i]],
              paste0("SNV_map_chr_",i,".txt"),
              row.names=FALSE, quote = FALSE)
}
```

- We save all 22 variant information files as `SNV_map_chr_i.txt` where "i" indicates the chromosome number and these files can be found in our Zenodo repository.
- Finally, we create a function to identify the cRV for each family.

4.4 List the familial cRVs

- We create the function `familial_cRV()` to identify the familial cRVs.
- We supply the third list element of the `sim_RVstudy_new()` function output, the `haplo_map` dataframe, as the argument of the `familial_cRV()` function. Then select the familial cRV for each family.

```
# Get the familial_cRVs

familial_cRV <- function(cRV){

  # From haplomap data frame get the familial
  # cRVs from the last column FamCRV.
  f_CRV <- lapply(cRV, function(x){
    unique(x[which(x$FamCRV != "no_CRV"), ])
  })

  # Combine all of them into a single data frame.
  family_CRV <- do.call("rbind", f_CRV)
  # Order them the data frame according to the family ID.
  family_CRV <- family_CRV[order(as.numeric(family_CRV$FamID)), ]
  # Get the family IDS that are not carrying a cRV,
  # by comparing two data frames.
  no_CRV <- as.numeric(setdiff(cRV[[1]]$FamID, family_CRV$FamID))
  no_CRVfam <- rep(c("no_CRV"), length(no_CRV))
  # Create a data frame with families without a cRV.
  df <- data.frame(no_CRV, no_CRVfam)
  # Give the same column names as in families with a cRV.
  colnames(df) <- colnames(family_CRV)
  # Combine both of the data frames.
  family_CRV <- rbind(family_CRV, df)
  # Order the final data frame based on the family ID.
  family_CRV <- family_CRV[order(as.numeric(family_CRV$FamID)), ]

  return(family_CRV)
```

```
}
```

- The function returns a data frame which catalogs the family identification number and the familial cRV.
- We apply the function as follows and get the familial cRV for each family.

```
# Get the unique FamIDs and familial cRVs
# by using the haplo_map data frame.
uniq_cRV <- lapply(study_seq, function(x){
  unique(x$haplo_map[, c("FamID", "FamCRV")])})

# Apply the function.
cRVS <- familial_cRV(uniq_cRV)

# View the output
head(cRVS, n = 6)
```

```
##    FamID      FamCRV
## 1      1 2_201170321
## 17     2 18_63125128
## 31     3 10_89015222
## 51     4 6_108683999
## 67     5 1_155738619
## 79     6 1_155738619
```

- The above output catalogs the familial cRVs for each family. As an example, the family ID 1 has a cRV labeled as "2_201170321." In the label, the digits before the "_" indicates the chromosome number on which the cRV resides and the digits after the "_" represent the base-pair position of the cRV on that chromosome.
- There are some families do not have cRVs. Let's look at them.

```
# Select families which do not carry cRVs
cRVS[cRVS$FamCRV == "no_CRV", ]
```

```
##    FamID FamCRV
## 11    72 no_CRV
## 2     95 no_CRV
## 3    103 no_CRV
```

- According the above output, the families with ID 72, 95 and 103 do not carry a cRV. For these families we can say that the disease has occurred sporadically.
- Finally, we save the results in a text file, `familial_cRV.txt`, and this file also can be found in the Zenodo repository.

```
# Save results in a text file.
write.table(familial_cRV(uniq_cRV),
            "familial_cRV.txt",
            row.names=FALSE, quote = FALSE)
```

- For reference, we print the R version and the version of the `SimRVSequences` R package.

```
library(SimRVSequences)

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] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] reshape2_1.4.4       doRNG_1.8.2          rngtools_1.5.2
##  [4] doParallel_1.0.16    iterators_1.0.13     foreach_1.5.1
##  [7] data.table_1.14.0    Matrix_1.2-12        forcats_0.5.1
## [10] stringr_1.4.0        dplyr_1.0.7          purrr_0.3.4
## [13] readr_2.0.1          tidyr_1.1.3          tibble_3.1.4
## [16] ggplot2_3.3.5        tidyverse_1.3.1      SimRVSequences_0.2.7
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.7          lubridate_1.7.10    lattice_0.20-44
##  [4] assertthat_0.2.1    digest_0.6.27       utf8_1.2.2
##  [7] R6_2.5.1            cellranger_1.1.0    plyr_1.8.6
## [10] backports_1.2.1     reprex_2.0.1        evaluate_0.14
## [13] httr_1.4.2          pillar_1.6.2        rlang_0.4.11
## [16] readxl_1.3.1        rstudioapi_0.13     kinship2_1.8.5
## [19] rmarkdown_2.11      munsell_0.5.0       broom_0.7.9
## [22] compiler_4.1.1      modelr_0.1.8        xfun_0.25
## [25] pkgconfig_2.0.3     SimRVPedigree_0.4.4 htmltools_0.5.2
## [28] tidyselect_1.1.1    codetools_0.2-18    intervals_0.15.2
## [31] quadprog_1.5-8      fansi_0.5.0         crayon_1.4.1
## [34] tzdb_0.1.2          dbplyr_2.1.1        withr_2.4.2
## [37] grid_4.1.1          jsonlite_1.7.2      gtable_0.3.0
## [40] lifecycle_1.0.0     DBI_1.1.1           magrittr_2.0.1
## [43] scales_1.1.1        cli_3.1.0           stringi_1.7.4
## [46] fs_1.5.0            xml2_1.3.2          ellipsis_0.3.2
## [49] generics_0.1.0      vctrs_0.3.8         tools_4.1.1
## [52] glue_1.4.2          hms_1.1.0           fastmap_1.1.0
## [55] yaml_2.2.1          colorspace_2.0-2    rvest_1.0.1
## [58] knitr_1.34          haven_2.4.3
```

Appendix

- The following is the `simRV_study_new()`function. The modifications added to the original `simRV_study()` function can be found on lines 1328 to 1338 of the following R code chunk.

```
sim_RVstudy_new <- function(ped_files, SNV_data, fam_RVs,
                            affected_only = TRUE,
                            remove_wild = TRUE,
                            pos_in_bp = TRUE,
                            gamma_params = c(2.63, 2.63/0.5),
                            burn_in = 1000,
                            SNV_map = NULL, haplos = NULL){

  if (!(is.null(SNV_map)) | !is.null(haplos)) {
    stop("Arguments 'SNV_map' and 'haplos' have been deprecated.
         \n Instead, please supply to argument 'SNV_data' an object of class SNVdata.
         Execute help(SNVdata) for more information." )
  }

  if (!(SimRVSequences:::is.SNVdata(SNV_data))) {
    stop("Expecting SNV_data to be an object of class SNVdata")
  }


  #check to see if DA1 and DA2 are both missing, if so
  #assume fully sporadic and issue warning
  if (is.null(ped_files$DA1) & is.null(ped_files$DA2)) {
    ped_files$DA1 <- 0
    ped_files$DA2 <- 0
    warning("\n The variables DA1 and DA2 are missing from ped_files.
            \n Assuming fully sporadic ...
            \n...setting DA1 = DA2 = 0 for all pedigrees.")
  }


  #check ped_files for possible issues
  SimRVSequences:::check_peds(ped_files)

  #assign generation number if not included in ped_file
  if(!"Gen" %in% colnames(ped_files)){
    ped_files$Gen <- unlist(lapply(unique(ped_files$FamID),
                                   function(x){SimRVSequences:::assign_gen(ped_files[ped_files$FamID ==
  }

  # save mutations and haplotypes in SNV_map and haplos objects
  SNV_map = SNV_data$Mutations
  haplos = SNV_data$Haplotypes

  #check to see that the sample contains affected relatives when the
  #affected_only setting is used
  if (affected_only & all(ped_files$affected  == FALSE)) {
    stop("\n There are no disease-affected relatives in this sample of pedigrees.
         \n To simulate data for pedigrees without disease-affected
         relatives use affected_only = FALSE.")
```

```
}

#collect list of FamIDs
FamIDs <- unique(ped_files$FamID)

#check for pedigree formatting issues
for (i in FamIDs){
  SimRVSequences:::check_ped(ped_files[ped_files$FamID == i, ])
}

#Reduce to affected-only pedigrees
if (affected_only) {
  #reduce pedigrees to contain only disease-affected relative and
  #the individuals who connect them along a line of descent.
  Afams <- lapply(FamIDs, function(x){
  SimRVSequences:::affected_onlyPed(ped_file = ped_files[which(ped_files$FamID == x),])
  })

  #combine the reduced pedigrees
  ped_files <- do.call("rbind", Afams)
  pedfiles <- ped_files
  #check to see if any pedigrees were removed due to lack of
  #disease affected relatives and issue warning for removed pedigrees
  removed_peds <- setdiff(FamIDs, unique(ped_files$FamID))

  if (length(removed_peds) > 0){
    FamIDs <- unique(ped_files$FamID)
    warning("\n There are no disease-affected relatives in the pedigrees with FamID: ",
            paste0(removed_peds, collapse = ", "),
            "\n These pedigrees have been removed from ped_files.")
  }

}

# Add is_CRV column again if it is not present
if (is.null(SNV_map$is_CRV)) {
  SNV_map$is_CRV = FALSE
  # warning("The variable is_CRV is missing from SNV_map.",
  #         "\n ... randomly sampling one SNV to be the cRV for all pedigrees.")
}

# Check whether any candidates for the familial cRV lies on the chromosome.
for(k in 1:length(FamIDs)){
  if(any(SNV_map$SNV == fam_RVs[k])){
    ped_files[ped_files$FamID == k, ]$DA1 <- ped_files[ped_files$FamID ==
                                                       k, ]$DA1
    ped_files[ped_files$FamID == k, ]$DA2 <- ped_files[ped_files$FamID ==
                                                       k, ]$DA2
  } else {
    ped_files[ped_files$FamID == k, ]$DA1 <- 0
    ped_files[ped_files$FamID == k, ]$DA2 <- 0
  }
}
```

```
#Given the location of familial risk variants, sample familial founder
#haplotypes from conditional haplotype distribution
 f_genos <- lapply(c(1:length(FamIDs)), function(x){
 sim_FGenos(founder_ids = ped_files$ID[which(ped_files$FamID == FamIDs[x]
                                             & is.na(ped_files$dadID))],
            RV_founder = ped_files$ID[which(ped_files$FamID == FamIDs[x]
                                            & is.na(ped_files$dadID)
                                            & (ped_files$DA1 + ped_files$DA2) != 0)],
            founder_pat_allele = ped_files$DA1[which(ped_files$FamID == FamIDs[x]
                                                     & is.na(ped_files$dadID))],
            founder_mat_allele = ped_files$DA2[which(ped_files$FamID == FamIDs[x]
                                                     & is.na(ped_files$dadID))],
            haplos, RV_col_loc = which(SNV_map$SNV == fam_RVs[x]),
            RV_pool_loc = SNV_map$colID[SNV_map$is_CRV])
})


#If desired by user, we now reduce the size of the data by removing
#markers not carried by any member of our study.
if (remove_wild) {
  reduced_dat <- SimRVSequences:::remove_allWild(f_haps = f_genos, SNV_map)
  f_genos <- reduced_dat[[1]]
  SNV_map <- reduced_dat[[2]]
}

#create chrom_map, this is used to determine the segments over
#which we will simulate genetic recombination
chrom_map <- SimRVSequences:::create_chrom_map(SNV_map)

#convert from base pairs to centiMorgan
if (pos_in_bp) {
  options(digits = 9)
  chrom_map$start_pos <- SimRVSequences:::convert_BP_to_cM(chrom_map$start_pos)
  chrom_map$end_pos <- SimRVSequences:::convert_BP_to_cM(chrom_map$end_pos)
  SNV_map$position <- SimRVSequences:::convert_BP_to_cM(SNV_map$position)
}

#simulate non-founder haploypes via conditional gene drop
ped_seqs <- lapply(c(1:length(FamIDs)), function(x){
  sim_seq(ped_file = ped_files[ped_files$FamID == FamIDs[x], ],
          founder_genos = f_genos[[x]],
          SNV_map, chrom_map,
          RV_marker = fam_RVs[x],
          burn_in, gamma_params)
})

ped_haplos <- do.call("rbind", lapply(ped_seqs, function(x){x$ped_genos}))
haplo_map <- do.call("rbind", lapply(ped_seqs, function(x){x$geno_map}))

#convert back to base pairs if we converted to CM
if (pos_in_bp) {
  options(digits = 9)
  SNV_map$position <- SimRVSequences:::convert_CM_to_BP(SNV_map$position)
```

```
  }

  return(SimRVSequences:::famStudy(list(ped_files = pedfiles, ped_haplos = ped_haplos,
                       haplo_map = haplo_map, SNV_map = SNV_map)))
}
```

- The following functions are the copies of the non-exported functions in `SimRVSequences` R package that are required for this document. These functions need to be copied into your R work-space for this document to work. These functions are the same as in the `SimRVsequences` R package except they use the `Matrix` package `which()` function instead of the base R `which()` function, unlike in the original `SimRVSequences`.

```
# Draw Founder Genotypes from Haplotype Distribution Given Familial Risk Variant
sim_FGenos <- function(founder_ids, RV_founder,
                       founder_pat_allele, founder_mat_allele,
                       haplos, RV_col_loc, RV_pool_loc) {

  #Determine which haplotypes carry the familial rare variant and which do not
  #Determine which haplotypes carry the familial cRV
  RV_hap_loc <- which(haplos[, RV_col_loc] == 1)

  #Determine which haplotypes do not carry ANY crv in the pool
  no_CRVrows <- SimRVSequences:::find_no_cSNV_rows(haplos, RV_pool_loc)



  #here we handle the fully sporatic families
  #i.e. families that do not segregate any cSNVs
  #In this case, the haplotypes for ALL founders
  #is sampled from no_CRVhaps
  if(length(RV_founder) == 0){
    #sample all founder data from this pool
    founder_genos <- haplos[sample(x = no_CRVrows,
                                   size = 2*length(founder_ids),
                                   replace = TRUE), ]
  } else {

    #sample the paternally inherited founder haplotypes
    pat_inherited_haps <- sapply(founder_pat_allele, function(x){
      if(x == 0){
        SimRVSequences:::resample(x = no_CRVrows, size = 1)
      } else {
        SimRVSequences:::resample(x = RV_hap_loc, size = 1)
      }})

    #sample the maternally inherited founder haplotypes
    mat_inherited_haps <- sapply(founder_mat_allele, function(x){
      if(x == 0){
        SimRVSequences:::resample(x = no_CRVrows, size = 1)
      } else {
        SimRVSequences:::resample(x = RV_hap_loc, size = 1)
      }})

    #pull the sampled haplotypes from the haplos matrix
```

```
    founder_genos <- haplos[c(pat_inherited_haps, mat_inherited_haps), ]
  }

  #create IDs to associate founders to rows in founder_genos
  founder_genos_ID <- rep(founder_ids, 2)

  #re-order so that founder haplotypes appear in order
  founder_genos <- founder_genos[order(founder_genos_ID), ]
  founder_genos_ID <- founder_genos_ID[order(founder_genos_ID)]


  return(list(founder_genos, founder_genos_ID))
}


sim_seq <- function(ped_file, founder_genos,
                    SNV_map, chrom_map, RV_marker,
                    burn_in = 1000, gamma_params = c(2.63, 2.63/0.5)){

  #Get parent/offspring information
  #i.e. for each offspring find RV_status,
  #parent IDs, and parent alleles at RV locus
  PO_info <- SimRVSequences:::get_parOffInfo(ped_file)
  PO_info <- PO_info[order(PO_info$Gen, PO_info$offspring_ID),]

  ped_genos <- founder_genos[[1]]
  ped_geno_IDs <- founder_genos[[2]]

  #determine the chromosome number and location of the familial RV locus
  #then store as a data frame with chrom in the first column
  RVL <- SNV_map[which(SNV_map$SNV == RV_marker),
                 which(colnames(SNV_map) %in% c("chrom", "position"))]

  if(colnames(RVL[1]) != "chrom"){
    RVL <- RVL[, c(2, 1)]
  }

  #for each offspring simulate transmission of parental data
  for (i in 1:nrow(PO_info)) {
    #simulate recombination events for this parent offspring pair
    loop_gams <- SimRVSequences:::sim_gameteInheritance(RV_locus = RVL,
                                        parent_RValleles = PO_info[i, c(6, 7)],
                                        offspring_RVstatus = PO_info[i, 5],
                                        chrom_map,
                                        allele_IDs = c(1, 2),
                                        burn_in, gamma_params)

    #construct offspring's inherited material from this parent
    loop_seq <- lapply(c(1:nrow(chrom_map)),
                       function(x){
                         SimRVSequences:::reconstruct_fromHaplotype(
                           parental_genotypes = ped_genos[which(ped_geno_IDs == PO_info[i, 4]),
                                                          which(SNV_map$chrom == chrom_map$chrom[x])],
```

```
                          CSNV_map = SNV_map[which(SNV_map$chrom == chrom_map$chrom[x]),],
                          inherited_haplotype = loop_gams$haplotypes[[x]],
                          chiasmata_locations = loop_gams$cross_locations[[x]],
                          REDchrom_map = chrom_map[x, ])
                      })

    #append ID for this haplotype to the list of IDs
    ped_geno_IDs <- c(ped_geno_IDs, PO_info[i, 1])

    ped_genos <- rbind(ped_genos, unlist(loop_seq))
  }

  #Determine if this is a sporadic pedigree
  printed_FamRV <- ifelse(all(ped_file[, c("DA1", "DA2")] == 0), "no_CRV", RV_marker)

  #create a data.frame to store identifying info
  geno_map <- data.frame(FamID = rep(ped_file$FamID[1], length(ped_geno_IDs)),
                         ID = ped_geno_IDs,
                         affected =  rep(FALSE, length(ped_geno_IDs)),
                         FamCRV = rep(printed_FamRV, length(ped_geno_IDs)),
                         stringsAsFactors = FALSE)

  #identify affected individuals
  geno_map$affected[geno_map$ID %in% ped_file$ID[ped_file$affected]] <- TRUE

  #Return the genomes matrix and a data.frame containing identifying
  #information for the of IDs to identify the
  #family member to whom
  return(list(ped_genos = ped_genos, geno_map = geno_map))
}
```

- The following functions, `SNVdata()` and `check_SNV_map`, are essentially the same as in the `SimRVSequences` R package except we change the label of one column of the SNV dataframe from `marker` to `SNV`.

```
# Constructor function for an object of class SNVdata
SNVdata_new <- function(Haplotypes, Mutations, Samples = NULL) {

  #check SNV_map for possible issues
  check_SNV_map_new(Mutations)

  if (!"SNV" %in% colnames(Mutations)) {
    Mutations$SNV <- make.unique(paste0(Mutations$chrom, sep = "_", Mutations$position))
  }

  if (nrow(Mutations) != ncol(Haplotypes)) {
    stop("\n nrow(Mutations) != ncol(Haplotypes).
         \n Mutations must catalog every SNV in Haplotypes.")
  }


  #create list containing all relevant of SNVdata information
  SNV_data = list(Haplotypes = Haplotypes,
                  Mutations = Mutations,
```

```
                  Samples = Samples)

  class(SNV_data) <- c("SNVdata", class(SNV_data))
  return(SNV_data)
}

# Check SNV_map for possible issues: modified version

check_SNV_map_new <- function(SNV_map){
  #check to see if SNV_map contains the column information we expect
  # and check to see if we have any missing values.

  ## Check colID variable
  if (!"colID" %in% colnames(SNV_map)) {
    stop('The variable "colID" is missing from SNV_map.')
  }
  if (any(is.na(SNV_map$colID))) {
    stop('Error SNV_map: The variable "colID" contains missing values.')
  }
  if (any(duplicated(SNV_map$colID))) {
    stop('Error SNV_map: The variable "colID" contains duplicate values.')
  }

  ## Check chrom variable
  if (!"chrom" %in% colnames(SNV_map)) {
    stop('The variable "chrom" is missing from SNV_map.')
  }
  if (any(is.na(SNV_map$chrom))) {
    stop('Error SNV_map: The variable "chrom" contains missing values.')
  }

  ## Check position variable
  if (!"position" %in% colnames(SNV_map)) {
    stop('The variable "position" is missing from SNV_map.')
  }

  if (any(is.na(SNV_map$position))) {
    stop('Error SNV_map: The variable "position" contains missing values.')
  }

  # Check to see if marker variable exists, and if so do all SNVs have a unique name
  if ("SNV" %in% colnames(SNV_map)) {
    if (length(unique(SNV_map$SNV)) != nrow(SNV_map)) {
      stop('Expecting each SNV to have a unique SNV name in SNV_map.')
    }
    if (any(is.na(SNV_map$SNV))) {
      stop('Error SNV_map: The variable "marker" contains missing values.')
    }
  }

}
```

References

Nieuwoudt, Christina, Angela Brooks-Wilson, and Jinko Graham. 2020. "SimRVSequences: An R package to simulate genetic sequence data for pedigrees." *Bioinformatics* 36 (7): 2295–97. https://doi.org/10.1093/bioinformatics/btz881.

Weeks, Jonathan P. 2010. "plink: An R Package for Linking Mixed-Format Tests Using IRT-Based Methods." *Journal of Statistical Software* 35 (12): 1–33. http://www.jstatsoft.org/v35/i12/.