

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

Nirodha Epasinghege Dona, Jinko Graham

2022-01-01

Contents

1	Read and process SLiM data in R	2
1.1	Extract and select the rare variants.	3
1.2	Extract genomes of rare SNVs.	5
1.3	Process data chromosome-by-chromosome.	5
1.4	Identify the pathway SNVs	7
1.5	Format of chromosome-by-chromosome data.	8
2	Select causal variants	9
3	Simulate genetic data for affected pedigree members	12
3.1	Modify <code>sim_RVstudy()</code> to simulate data chromosome-by-chromosome	12
3.2	Set arguments to <code>sim_RVstudy_new()</code>	17
3.3	Discuss the <code>sim_RVstudy_new()</code> output	21
4	Generate data files	23
4.1	<code>.sam</code> file	24
4.2	<code>.geno</code> file	25
4.3	<code>.var</code> file	26
4.4	List the familial cRVs	28
	References	32

This document discusses the third and final step in our work-flow to simulate exome-sequencing data in pedigrees ascertained to have four or more relatives affected with lymphoid cancer (the orange box labelled 3).

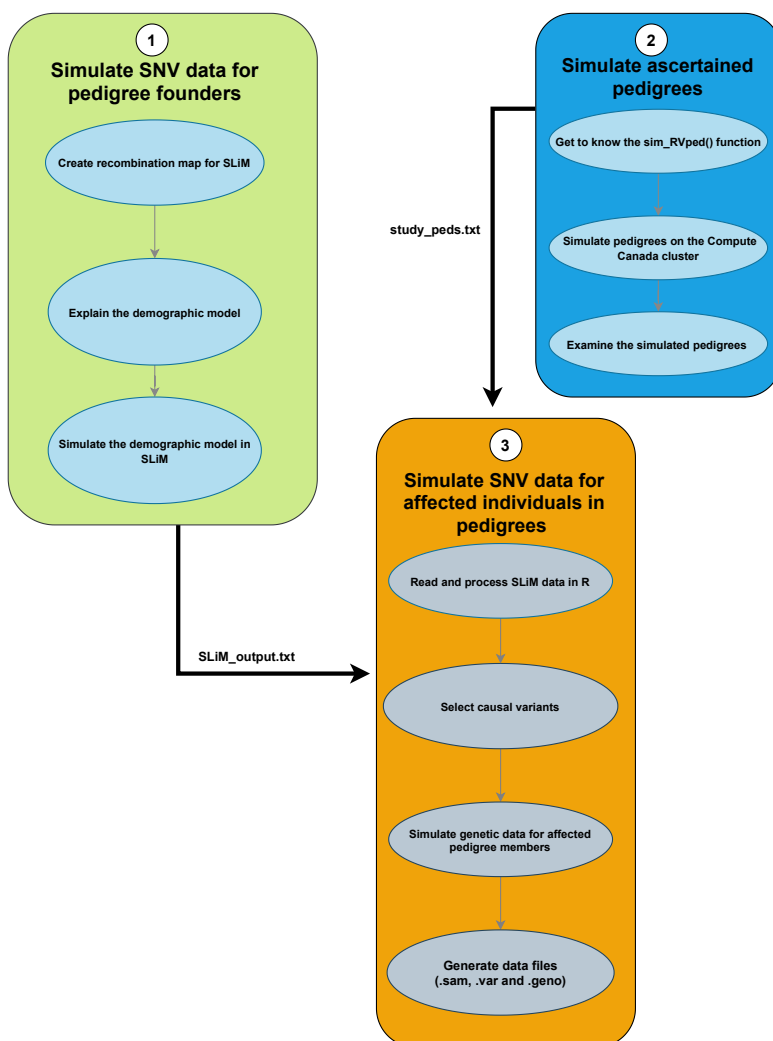


Figure 1: Work-flow for simulating the exome-sequencing data for ascertained pedigrees.

1 Read and process SLiM data in R

Our final goal is to simulate exome sequences for disease-affected members of the ascertained pedigrees. In the first supplementary materials document we obtained exome sequences for an American-admixed population. In the second supplementary materials document we obtained the ascertained pedigrees. Now, we have only to select sequences for the pedigree founders from the population, and then “drop” them through the pedigrees to descendants. We use the gene-dropping functions available in the `SimRVSequences` (Nieuwoudt,

Brooks-Wilson, and Graham 2020) R package, which require sparse matrices of SNV sequences. Unfortunately, the large population size and number of single-nucleotide variants (SNVs) exceeds R's memory capacity for a single sparse matrix. Therefore, we read in the population sequences and create the sparse matrices chromosome-by-chromosome, as described next.

We start by reading the SLiM simulation output, `SLiM_output.txt`, into R. This text file is of size approximately 6 GB and contains all the exome sequences in the American-admixed population. The file takes approximately 1 minute to read on a Windows OS with an i7-8550U @ 1.8GHz, 16GB of RAM.

```
library(Matrix) #this package is required throughout this document
# Read the text file to R.
# Note: Change the path for the file as necessary.
exData <- readLines("/Users/jgraham/OneDrive - Simon Fraser University (1sfu)/NirodhaStuff/Data/SLiM_ou
```

We next select rare SNVs based on their population derived (mutated) allele frequencies, as described in the next subsection.

1.1 Extract and select the rare variants.

First, we find the line numbers of the mutation and genome header sections in the SLiM output. The formatting of the SLiM output is described in first supplementary materials document.

```
# 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 SNVs 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 in the simulated American-admixed population. The rows of this data frame correspond to SNVs and the columns to the following SNV characteristics of interest:

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. To select only the rare SNVs, we assign a `colID` to each based on a threshold value for its minor-allele frequency (MAF). Rare SNVs with MAFs below the threshold are assigned non-zero `colID`s that increase according to their physical order on the exome. Common SNVs are assigned a `colID` of zero, as they will be discarded.

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

# Variants with MAF below the threshold are assigned non-zero
# colIDs according their physical order on the exome.

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

We next create a new data frame, `RareMutData`, of rare SNVs in the American-admixed population.

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

To loop through the chromosomes and separate their data, we need to provide the chromosome on which each SNV resides. We identify the chromosome of each SNV with the `reMap_mutations()` internal function in the `SimRVSequences` package. The `reMap_mutations()` internal function requires a recombination map identifying the exon positions on chromosomes. The recombination map is obtained by calling the `create_slimMap()` function in the `SimRVSequences` R package.

```

# 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)

```

The call to `reMap_mutation()` adds a new column, `chr`, to `RareMutData`. We are now in a position to extract genomes from the SLiM output, as discussed in the next subsection.

1.2 Extract genomes of 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 an exome sequence 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 the sequences.

```

# 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)
})

```

For the American-admixed population, we now have the rare SNVs (`RareMutdata`) and the sequences (`RareGenomes`) comprised of them. These sequences will be separated chromosome-by-chromosome, as described in the next subsection.

1.3 Process data chromosome-by-chromosome.

The code chunk below separates the large number of mutations and genomes in the SLiM output by chromosome. It takes approximately 16 hours to run on a Windows OS with an i7-8550U @ 1.8GHz, 16GB. Therefore, we recommend against running it to knit the document. Instead, load the `Chromwide.Rdata` file which can be found in the Zenodo repository. We loop over the 22 chromosomes with the `foreach()` function to parallelize the R code.

We start by creating the haplotypes matrix for the corresponding chromosome. The haplotypes matrix is a sparse matrix of class `dgCMatrix`. The rows of the matrix correspond to haplotypes in the population and the columns to rare SNVs that lie on the targeted chromosome. For any chromosome, the number of rows in the haplotypes matrix is the number of haplotypes in the population (107,752), or the length of the `RareGenomes` R object. We get the IDs for SNVs that lie on a particular chromosome from the `RareMutData` R object and match them to the column IDs of the haplotypes in the `RareGenome` R object. The SNVs are ordered according to their base-pair position along the chromosome. The SNVs in the columns of the chromosome-specific haplotypes matrix `GenoData` and in the rows of the chromosome-specific mutation data-frame `RareMutData_new` are named according to their chromosome and base-pair position. The following code chunk implements 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 the mutation 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

To identify the SNVs that lie on the pathway of interest, 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.

```

# Load the output generated from the previous code chunk.
load("/Users/jgraham/OneDrive - Simon Fraser University (1sfu)/NirodhaStuff/Data/Chromwide.Rdata")

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

```

The call to `identify_pathwaySNVs()` adds an additional column to the mutation data frame labelled `pathwaySNV`. This column identifies SNVs that lie on the pathway as `TRUE`.

Finally, we combine the chromosome-specific haplotypes matrices with the chromosome-specific mutation

data frame as follows.

```
# Get the final output using a for-loop.
# Loop over the chromosomes to get a list 22 elements.
# Each element is itself a list which contains the haplotypes
# matrix and the mutation data frame for that chromosome.

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 element corresponds to a chromosome and is itself a list with two elements, a haplotypes matrix and a mutation data frame. The haplotypes matrix contains the chromosome-specific exome sequences (genomes) of all 107,752 individuals in the simulated American-admixed population. For pedigree founders, these exome sequences are sampled from the haplotypes matrix. The mutation data frame contains information on the SNVs that reside on the chromosome. For 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. The first chromosome has 84,664 SNVs. Let's print the first ten rows and columns of its haplotypes matrix.


```
# 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 is a sparse matrix of class `dgCMatrix` from the `Matrix` package. Rows correspond to individuals and columns correspond to SNVs on the individual's exome from chromosome 1. Entries with “1” and “.” indicate the derived (mutated) allele and the ancestral allele, respectively.

Let's print the first ten rows of the mutation data frame for chromosome 1.

```
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 and columns of the mutation data frame represent the SNVs and their characteristics, respectively. The column variable `colID` links the rows in the mutation data frame to the columns of haplotypes matrix, `chrom` is the chromosome on which the SNV lies, `position` is the position of the SNV along the chromosome in base pairs, `afreq` is the SNV's population derived allele frequency, `SNV` is a unique character identifier for the SNV, and `pathwaySNV` identifies whether or not SNVs are located within the apoptosis sub-pathway of interest.

The next task is to select the causal rare variants (cRV).

2 Select causal variants

We create a function, `select_cRV()`, to select causal rare variants (cRVs). `select_cRV()` considers SNVs in genes on an apoptosis sub-pathway as candidates for cRVs. Among these, cRVs are selected from population singletons, on the basis of their absolute selection coefficients, until the cumulative probability of a sequence carrying a cRV is 0.001. Selection coefficients are less than or equal to zero because mutations are deleterious or selectively neutral in the SLiM simulation. The function has two required arguments:

1. `chrn_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.

`select_cRV()` selects the relevant mutation data frames and haplotypes matrices from the `slim_out` R object.

Singleton SNVs which lie on the specified pathway (apoptosis sub-pathway) are determined from the haplotypes matrices of the second argument. A weight is assigned to each singleton in the pathway 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. These weights are used as the sampling probabilities for drawing causal rare variants (cRVs) from the pool of singleton SNVs. The cRVs are sampled randomly with these weights until their cumulative, derived-allele frequency in the population is 0.001.

The `select_cRV()` function is:

```
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 mutation 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 while 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$SNV == selected_cRVs, 4]
# Update the cumulative sum of the allele frequencies of causal SNVs.
cum <- cum + af
# Remove the selected SNV from the mutation data frame.
# If not we may get this same SNV again.
SNV_pathway <- SNV_pathway[-(which(SNV_pathway$SNV == selected_cRVs)),
]
# Save the selected cRVs in a vector
cSNVs <- c(cSNVs, selected_cRVs)
}
# Create a new variable in our mutation 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)
}

```

Below is an example call to `select_cRV()`.

```

# 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 function is a mutation data frame with an additional column, `is_CRV`. This column catalogs which SNVs are selected as causal variants. We may then print the number of cRVs in the population, their

cumulative allele frequency and their chromosomes as follows.

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

## [1] 108

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

## [1] 0.001

# 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, 108 SNVs are sampled as cRVs. Their cumulative derived-allele frequency in the population is 0.001. The table summarizes the number of cRVs on each chromosome. As an example, only one cRV is on chromosome 1; 21 cRVs reside in chromosome 2 and so forth.

We are now ready to simulate exome sequences for the affected individuals in the 150 ascertained pedigrees, as described in the next section.

3 Simulate genetic data for affected pedigree members

The `sim_RVstudy()` function of the `SimRVSequences` R package simulates genetic sequence data in pedigrees, but expects only a single population database of sequences as an argument in the form of a sparse matrix of SNV haplotypes and an associated mutation data frame. Unfortunately, we cannot use `sim_RVstudy()` because the number of individuals and SNVs in our American-admixed population far exceeds R's memory capacity to fit the population sequences into a single sparse matrix. We therefore modify `sim_RVstudy()` and various supporting functions in the `simRVSequences` package to handle chromosome-specific databases, as described in the next subsections.

3.1 Modify `sim_RVstudy()` to simulate data chromosome-by-chromosome

We add a new argument, `fam_RVs`, to `sim_RVstudy()` that identifies the familial cRVs. This argument is used to check whether or not the familial cRV lies on the targeted chromosome. If the familial cRV is on the targeted chromosome, the chromosome is segregated through the pedigree with conditional gene-dropping (Nieuwoudt, Brooks-Wilson, and Graham 2020). Otherwise, the chromosome is segregated through the pedigree according to Mendelian law. The updated `sim_RVstudy_new()` function below has these changes as marked in the comments for the source code.

```
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 lie on the chromosome. This next block of code is
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, reduce the size of the data by removing
#markers not carried by any member of the 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 haplotypes 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)))
}

```

`sim_RVstudy()` requires the argument `SNV_data`, an object of class `SNVdata`. The `SNVdata()` function in the `SimRVSequences` R package converts haplotypes matrices and mutation data frames into an object of class `SNVdata`. We modified the `SNVdata()` and the `check_SNV_map()` functions of the package to align with our changes in `sim_RVstudy_new()`. The `SNVdata()` and `check_SNV_map()` functions remain the same except that the mutation data frame provided as an argument now has a column with assumed name `SNV` rather than `marker`. When the targeted chromosome contains no cRV, the original `check_SNV_map()` function exits with an error. The original function exits because it inappropriately checks whether the `is_CRV` column is `FALSE` for all the SNVs in the mutation data frame. To avoid the inappropriate exit, we remove the check from the function. The modified versions of `SNVdata()` and `check_SNV_map()` are renamed as `SNVdata_new()` and `check_SNV_map_new()` and defined in the next code chunk.

```

# Define the SNVdata_new() and check_SNV_map_new() functions

# 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.')
    }
  }
}

```

The next subsection discusses how we set the arguments of `sim_RVstudy_new()`.

3.2 Set arguments to `sim_RVstudy_new()`.

`sim_RVstudy_new()` requires three arguments: `fam_RVs` giving the cRV for each ascertained pedigree, `ped_files` giving the ascertained pedigrees and `SNV_data` giving the chromosome-specific exome sequences and associated mutation data frames for everyone in the American admixed population. We prepare these three arguments as follows.

(1). `fam_RVs`

Familial cRVs are sampled on the basis of their population derived allele frequencies as follows.

```

# Load all 150 pedigrees.
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.

(2). `ped_files` is a data frame that represents the ascertained pedigrees. For this, we read in the `study_peds.text` file prepared in the second Supplementary Material document on how to simulate ascertained pedigrees.

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

The final output, `study_peds` is a data frame with information on the 150 ascertained pedigrees.

(3). `SNV_data` gives a database of exome sequences for a single chromosome, for everyone in the American-admixed population. This argument is an object of class `SNVdata`. Objects of class `SNVdata` are comprised of a sparse matrix of exome sequences together with an associated data frame of mutation information.

We first make a list, by chromosome, of objects of class `SNVdata` by applying the `SNVdata_new()` function:

```
# Apply the SNVdata_new function to 22 SNVdata objects comprised
# of sparse matrices of SNV sequences and mutation data frames.
chrom_data <- lapply(slim_out, function(x){
  SNV_data = SNVdata_new(Haplotypes = x$Haplotypes,
                        Mutations = x$Mutations))})
```

The `SNV_data` argument of `sim_RVstudy_new()` will be extracted from the appropriate list element of the `chrom_data` object.

The remaining arguments to `sim_RVstudy_new()` are optional and set with the defaults of the original `sim_RVstudy()` function. For example, we use the default value `affected_only = TRUE` to simulate sequence data for disease-affected members in the pedigree, as is typical for exome-sequencing studies of families ascertained for multiple affected relatives. Affected relatives from such families are more likely to carry a cRV. We also set the default value of `remove_wild = TRUE`, to shrink the sequence data for the study by removing monomorphic SNVs.

To obtain the genetic sequences for disease-affected family members, we loop over chromosomes and apply the `simRV_study_new()` function to each. First, however, we load functions required by `simRV_study_new()`. These functions are slightly modified versions of `sim_FGenos()` and `sim_seq()`, two non-exported functions from the `SimRVSequences` R package. The modified versions are the same as their counterparts in `SimRVsequences`, except they use the `Matrix` package's `which()` function instead of base R's.

```
# 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 sporadic 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))
}

#Now the modified version of the sim_seq() function.
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))
}

```

We are now ready to call `sim_RVstudy_new()` on each chromosome.

```

# Simulate exome sequences of SNVs 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]]
)})

```

Simulating exome-wide sequences for disease-affected members in the study families takes about 6 minutes

on a Windows OS with an i7-8550U @ 1.8GHz, 16GB of RAM. The times to simulate chromosomes 1, 2, 8 and 9 are shown in Table 1.

Table 1: Simulation time for selected chromosomes.

Chromosome	No. of RVs	No. of cRVs	Time (s)
1	84664	1	36.23
2	60995	21	53.63
8	31396	11	22.35
9	34248	0	19.57

From the table, we see that chromosome 1 takes less time to simulate than chromosome 2, despite having more rare variants. This may be because chromosome 1 has fewer cRVs than chromosome 2. By contrast, chromosome 8 has more cRVs than chromosome 1 yet takes less time to simulate because it has fewer RVs overall. Simulation time therefore depends on both the overall number of RVs and the number of cRVs on chromosome.

The `sim_RVstudy_new()` function returns the same set of outputs as the `sim_RVstudy()` function, as discussed in the next subsection.

3.3 Discuss the `sim_RVstudy_new()` output

The output `study_seq` from the call to `sim_RVstudy_new()` is a list containing 22 elements, one for each chromosome. As the output format of each chromosome is the same, we focus on the first chromosome. Each element of the list `study_seq` is itself a list containing four elements as follows.

- (1). The `ped_files` data frame gives 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 within a pedigree. Note that the `ped_files` data frame is exactly the same for all 22 chromosomes; though wasteful of space this repetition will be convenient for looping.

```
# View the first 4 individuals in the ped_files data frame (the same regardless of 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` contains simulated SNVs on the exome sequences of the disease-affected individuals and the individuals connecting them in the ascertained pedigrees.

```
# View the first 10 SNVs of the first 10 exome sequences on the first chromosome.
```

```
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,] . . . . .
```

Rows of this sparse matrix correspond to exomes and columns to RVs on the first chromosome. The entry “1” represents the derived (mutated) allele and “.” the ancestral allele.

(3). The `SNV_map` data frame contains information about RVs in the study. Since the `remove_wild` argument of `sim_RVstudy_new()` is set to its default value of `TRUE`, this data frame contains only RVs carried by at least one study individual.

```
# View the first 4 rows of 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 data frame represent the RVs carried by at least one individual in the study. The columns are characteristics of the RVs explained in subsection 1.5 of this document.

(4). The `haplo_map` data frame maps the exome sequences in `ped_haplos` to the individuals in `ped_files`. The rows of `haplo_map` correspond to sequences and the columns to characteristics of individuals to which these sequences belong.

```
# View the first 18 rows of 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
```

We can see that the two sequences of an individual are stored in consecutive rows of the data frame. The `FamCRV` column of the data frame gives the identifier of the familial cRV. If a family does not have a cRV on the selected chromosome, the entry of `FamCRV` is `no_CRV`. For example, family ID 1 does not carry a cRV on chromosome 1.

With the complete data now available in the list `study_seq`, our final task is to deliver it in human-readable flat-file formats, as described next.

4 Generate data files

Throughout this section, we will refer to `study_seq` list object generated in the previous subsection. The list element of `study_seq` for chromosome 21 has the following structure.

```
# The study_seq object is a list of length 22 elements.
# We print the 21st element of study_seq, for chromosome 21.
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"
```

The `study_seq` object is used to create a `.sam` file containing information about genotyped individuals in the ascertained pedigrees, a `.geno` file for each chromosome containing RV genotypes, and a `.var` file for each chromosome containing information about RVs. As described next, the data files are in flat-file format similar to PLINK files (Purcell et al. 2007).

4.1 .sam file

The `.sam` file contains pedigree information about the disease-affected individuals and the individuals connecting them along a line of descent in their pedigrees. We assume these individuals would be prioritized for exome sequencing in our study. The function `plink_format_samp()` generates the `.sam` file using the argument, `peds`. The argument `peds` is a data frame giving information on the study pedigrees. The function selects specific columns of the `peds` data frame and aligns them in a format similar to the `.psam` PLINK file.

```
# Get the information for .sam 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", "PHEN01",
                          "BIRTHYr", "DEATHYr", "PROBAND")

  return(psam_file)
}
```

To get the `.sam` file, we apply the `plink_format_samp()` function to chromosome 21. Note that the `.sam` file is the same regardless of which chromosome is used.


```
# Call the function for chromosome 21
sample_data <- plink_format_samp(study_seq[[21]]$ped_files)
```

```
# How many individuals?
nrow(sample_data)
```

```
## [1] 1247
```

```
# Print the 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 `sample_data` are the 1247 genotyped individuals in the study pedigrees. The individuals are either disease-affected or connect disease-affected individuals along a line of descent in a study pedigree. The columns of `sample_data` contain information about the individuals 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, with 1 and 2 corresponding to male and female, respectively.
6. PHENO1- the disease-affected status, with 1 and 2 corresponding to unaffected and affected, respectively.
7. BIRTHYr- the individual's birth year.
8. DEATHYr- the death year of the individual, with NA indicating that the individual is still alive at the end of the study.
9. PROBAND- a logical value indicating whether or not the individual is the proband for their pedigree.

We save the `.sam` file as a text file, `sample_info.txt`, as follows. The text file can be found in our Zenodo repository.

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

4.2 .geno file

The `.geno` file gives the RV genotypes in gene-dosage format. An individual's dosage of the derived allele is the number of copies they inherited from their parents (i.e. 0, 1 or 2). The `gene_data()` function below converts RV-haplotype pairs for individuals into genotypes in gene-dosage format.

```
# Get the haplotypes and convert them to genotypes in gene-dosage format.
```

```
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 call `gene_data()` on chromosome 21 as an example. The function's argument, `geno`, is filled with the sparse matrix `ped_haplos` from the `study_seq` output.

```

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

```

To convert chromosome 21 haplotypes to individual genotypes in gene-dosage format, `gene_data()` takes approximately 15 seconds on a Windows OS with an i7-8550U @ 1.8GHz, 16GB of RAM. Let's view the first five rows and columns of the data frame returned by `gene_data()`.

```

# 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 data frame represent the 1247 genotyped individuals in our study. The columns represent RVs that reside on the exome of chromosome 21. Each entry of the data frame gives the dosage of the derived allele of an RV (i.e. 0, 1 or 2). Most of the entries are 0, as would be expected for RVs.

The `gene_data()` function is applied to all the 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
})

```

Below, the resulting chromosome-specific `.geno` files are written to text files named `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 contains information about the SNVs in the columns of the `.geno` file. The `variant_data()` function below selects the relevant characteristics of SNVs and stores them in a data frame.

```

# Get the variant information to create the .var 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)
}

```

Let's call `variant_data()` on chromosome 21 as an example. The function's argument, `variant`, is filled with the `SNV_map` data frame from the `study_seq` output.

```

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

```

The function returns the data frame `variant_info`. Let's view information about the first four RVs in `variant_info`.

```

# 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 `variant_info` contain exomic RVs on chromosome 21 that are carried by at least one study participant. The columns give the following information about these RVs:

1. CHROM- the chromosome number of the RV.
2. POS- the RV position, in base pairs, on the chromosome.
3. REF- the reference allele for the RV

4. ALT- the alternate allele for the RV
5. pop_afreq- the population alternate allele frequency for the RV.
6. sel_coef- the selection coefficient for the RV.
7. pathwaySNV- whether or not the RV comes from a gene in our disease pathway.
8. C_SNV- whether or not the RV is causal.
9. Type- whether the RV is a synonymous (S) or non-synonymous (NS) mutation.

The `variant_data()` function is applied to all the chromosomes as follows.

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

Below, the resulting chromosome-specific `.var` files are written to text files named `SNV_map_chr_i.txt`, where “i” indicates the chromosome number. These text files can be found in the Zenodo repository.

```
# 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)
}
```

The next section provides a data frame listing the cRVs for each ascertained family.

4.4 List the familial cRVs

First, we obtain a list of family-specific cRVs by chromosome. Each list item corresponds to a chromosome and is a data frame with the family identifiers and the familial cRVs on that chromosome.

```
# Get the FamIDs and their familial cRVs, by chromosome
famcRV_bychrom <- lapply(study_seq, function(x){
  unique(x$haplo_map[, c("FamID", "FamCRV")]))
str(famcRV_bychrom)
```

```
## List of 22
## $ : 'data.frame': 150 obs. of 2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame': 150 obs. of 2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "2_201170321" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame': 150 obs. of 2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame': 150 obs. of 2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame': 150 obs. of 2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame': 150 obs. of 2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
```

```
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "6_108683999" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "10_89015222" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "18_63125128" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no_CRV" "no_CRV" "no_CRV" "no_CRV" ...
## $ : 'data.frame':      150 obs. of  2 variables:
## ..$ FamID : int [1:150] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ FamCRV: chr [1:150] "no CRV" "no CRV" "no CRV" "no CRV" ...
```

From the output of `str()`, we see that each list item corresponds to a data frame giving the family identifier and its cRV, if any, for the particular chromosome.

Next, we create the function `familial_crv()` to collapse `famRV_bychrom` into a **single data frame** containing the familial identifier and cRV for each family **across all chromosomes**.

```

# Get the familial_cRVs
familial_cRV <- function(cRV_bychrom){
  # From haplomap data frame get the familial
  # cRVs from the last column FamCRV.
  f_cRV <- lapply(cRV_bychrom, 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$FamID)), ]
  # Get the family IDS that are not carrying a cRV,
  # by comparing two data frames.
  no_CRV <- as.numeric(setdiff(cRV_bychrom[[1]]$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)
}

```

We apply the function as follows and get the familial cRV for each family.

```

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

# 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 output gives the cRVs for each family. As an example, family ID 1 has a cRV labeled “2_201170321” indicating that it is on chromosome 2 in base-pair position 20117032. Although not shown in the output, there are a few families without a cRV:

```

# 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

```

Three families with IDs 72, 95 and 103 do not carry a cRV. These families have affected individuals with

sporadically occurring disease.

We save the results in a text file, `familial_cRV.txt`, which can be found in the Zenodo repository.

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

As a final step, for future reference, we provide the R version and the version of the `SimRVSequences` R package that was used to generate this document.

```
library(SimRVSequences)
sessionInfo()

## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS 10.16
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_CA.UTF-8/en_CA.UTF-8/en_CA.UTF-8/C/en_CA.UTF-8/en_CA.UTF-8
##
## 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
## [4] doParallel_1.0.15 iterators_1.0.12 foreach_1.5.0
## [7] data.table_1.14.0 Matrix_1.2-18 forcats_0.5.0
## [10] stringr_1.4.0 dplyr_1.0.2 purrr_0.3.4
## [13] readr_1.3.1 tidyr_1.1.1 tibble_3.0.3
## [16] ggplot2_3.3.5 tidyverse_1.3.0 SimRVSequences_0.2.7
##
## loaded via a namespace (and not attached):
## [1] Rcpp_1.0.7 lubridate_1.7.9 lattice_0.20-41
## [4] assertthat_0.2.1 digest_0.6.25 R6_2.4.1
## [7] cellranger_1.1.0 plyr_1.8.6 backports_1.1.8
## [10] reprex_0.3.0 evaluate_0.14 httr_1.4.2
## [13] pillar_1.4.6 rlang_0.4.12 readxl_1.3.1
## [16] rstudioapi_0.11 kinship2_1.8.5 blob_1.2.1
## [19] rmarkdown_2.11 munsell_0.5.0 broom_0.7.10
## [22] compiler_4.0.2 modelr_0.1.8 xfun_0.28
## [25] pkgconfig_2.0.3 SimRVPedigree_0.4.4 htmltools_0.5.2
## [28] tidyselect_1.1.0 intervals_0.15.2 quadprog_1.5-8
## [31] codetools_0.2-16 fansi_0.4.1 crayon_1.3.4
## [34] dbplyr_1.4.4 withr_2.2.0 grid_4.0.2
## [37] jsonlite_1.7.0 gtable_0.3.0 lifecycle_0.2.0
## [40] DBI_1.1.0 magrittr_1.5 scales_1.1.1
## [43] cli_2.0.2 stringi_1.4.6 fs_1.5.0
## [46] xml2_1.3.2 ellipsis_0.3.1 generics_0.1.1
## [49] vctrs_0.3.2 tools_4.0.2 glue_1.4.1
```

```
## [52] hms_0.5.3          fastmap_1.1.0      yaml_2.2.1
## [55] colorspace_1.4-1    rvest_0.3.6        knitr_1.29
## [58] haven_2.3.1
```

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>.
- Purcell, S., B. Neale, K. Todd-Brown, L. Thomas, M. A. Ferreira, D. Bender, J. Maller, et al. 2007. “PLINK: a tool set for whole-genome association and population-based linkage analyses.” *Am J Hum Genet* 81 (3): 559–75.