SimRVSequences

Christina Nieuwoudt and Jinko Graham

2019-10-10

Contents

Introduction

Compared to case-control studies, family-based studies offer more power to detect causal rare variants, require smaller sample sizes, and detect sequencing errors more accurately [12]. However, data collection for family-based studies is both time-consuming and expensive. Furthermore, evaluating methodology to identify causal rare variants requires simulated data. In this vignette we illustrate how the methods provided by `SimRVSequences` may be used to to simulate sequence data for family-based studies of diploid organisms.

Please note that `SimRVSequences` does NOT allow users to simulate genotypes conditional on phenotype. Instead, `SimRVSequences` simulates haplotypes conditional on the genotype at a single causal rare variant (cRV), via the following algorithm. Additionally, `SimRVSequences` is only intended to simulate single-nucleotide variant (SNV) data for diploid organisms.

1. For each pedigree, we sample a cRV from a pool of SNVs specified by the user (see section 3.3); this is the familial cRV.
2. After identifying the familial cRV we sample founder haplotypes from haplotype data conditional on the founder's cRV status at the familial cRV.
3. Proceeding forward in time, from founders to more recent generations, for each parent/offspring pair we
    1. simulate recombination and formation of parental gametes (see section 6), and then
    2. perform a conditional gene drop to model inheritance of the cRV (see section 6).

User Requirements

To simulate sequence data for a family-based study users are expected to provide:

1. a sample of ascertained pedigrees, and
2. single-nucleotide variant (SNV) data from a sample of unrelated individuals representing the population of pedigree founders.

We streamline use of pedigrees simulated by the `R` package `SimRVPedigree` [8]. In section 3 of this vigentte, we outline three different SNV data options. The flow chart in Figure 1 illustrates the preparation required for each of the above inputs. We note that simulation of SNV data with SLiM [3] is NOT accomplished in `R`. However, we do provide pre- and post-processing methods to assist with importing and formatting the SNV data simulated by SLiM, which are discussed at length in section 3.1. In section 4 we briefly review simulating pedigree data using the `SimRVPedigree` package.

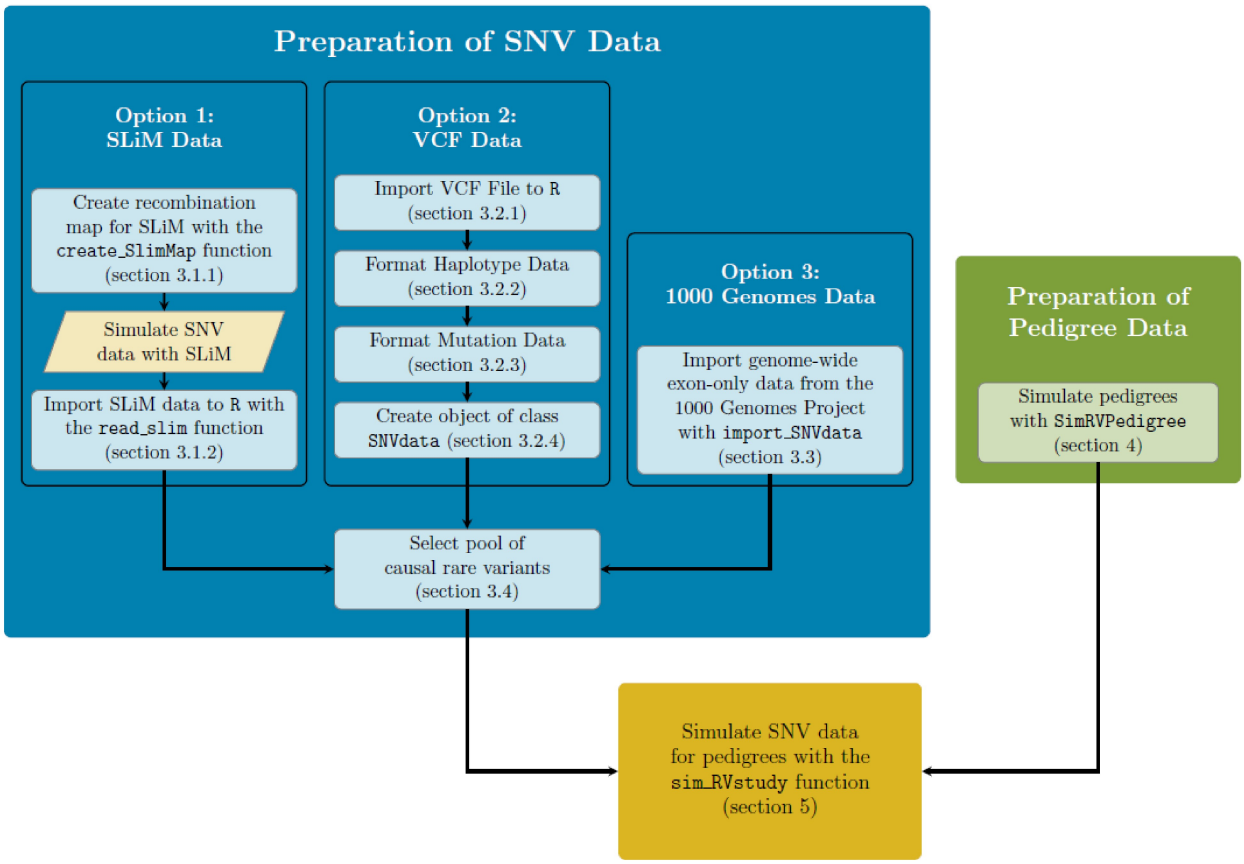


Figure 1: **Figure 1.** Data preparation flow chart. All tasks completed in R are enclosed in rectangular frames while the SLiM simulation component seen in option 1 of the Prepare SNV Data is enclosed in a parallelogram.

Prepare Single Nucleotide Variant Data

Before we can simulate genetic data for a sample of pedigrees, we require single-nucleotide variant (SNV) data for the pedigree founders. Several simulation programs can aid in this task (e.g. msprime and fastsimcoal); however, we have found the Eidos program SLiM [3] to be the most practical for simulation of genome-wide, exon-only SNV data for a large number of unrelated individuals. In section 3.1, we demonstrate how the methods provided by `SimRVSequences` can be used to assist users in simulating exon-only sequence data with SLiM. For users who would prefer to upload SNV data from exisiting VCF files, we provide a detailed demonstration of the process required to import and format VCF data in section 3.2. Additionally, we offer formatted genome-wide exon-only SNV data from tho 1000 Genomes Project (CITATION) which can be imported using our `import_SNVdata` function; we illustrate the usage of this function in section 3.3.

Option 1: Simulate SNV data with SLiM

One of SLiM's many features is that it simulates recombination hotspots using a user-specified recombination map. This recombination map may be utilized to simulate SNVs over unlinked regions (i.e. in different chromosomes) or in linked but non-contiguous regions (i.e in exon-only data). In section 3.1 we demonstrate how the `create_slimMap` function may be used to generate the recombination map required by SLiM to simulate exon-only SNV data over the entire human genome.

After simulating SNV data with SLiM, users may import the simulated data to `R` using the `read_slim` function. This task is demonstrated in section 3.2.

Finally, in section 3.3 we demonstrate how users may implement a pathway approach by specifying a pool of rare variants from which to select causal familial variants.

Create Recombination Map to Simulate Genome-Wide, Exon-Only Data with SLiM

The `SimRVSequences` function `create_slimMap` simplifies the task of creating the recombination map required by SLiM to simulate exon-only SNV data. The `create_slimMap` function has three arguments:

1. `exon_df`: A data frame that contains the positions of each exon to simulate. This data frame must contain the variables `chrom`, `exonStart`, and `exonEnd`. *We expect that `exon_df` does not contain any overlapping segments. Prior to supplying the exon data to `create_slimMap` users must combine overlapping exons into a single observation. Our `combine_exons` function may be used for this task (see appendix for details).*
2. `mutation_rate`: the per-site per-generation mutation rate, assumed to be constant across the genome. By default, `mutation_rate= 1E-8`, as in [4].
3. `recomb_rate`: the per-site per-generation recombination rate, assumed to be constant across the genome. By default, `recomb_rate= 1E-8`, as in [4].

We will illustrate usage of `create_slimMap` with the `hg_exons` data set.

```
# load the SimRVSequences library
library(SimRVSequences)

# load the hg_exons dataset
data("hg_exons")

# print the first four rows of hg_exons
head(hg_exons, n = 4)
```

```
##   chrom exonStart exonEnd               NCBIref
## 1     1     11874   12227          NR_046018.2
## 2     1     12613   12721          NR_046018.2
```

```
## 3     1     13221   14829 NR_046018.2, NR_024540.1
## 4     1     14970   15038              NR_024540.1
```

As seen in the output above, the `hg_exons` data set catalogs the position of each exon in the 22 human autosomes. The data contained in `hg_exons` were collected from the hg 38 reference genome with the UCSC Genome Browser [5, 6]. The variable `chrom` is the chromosome on which the exon resides, `exonStart` is the position of the first base pair in the exon, and `exonEnd` is the position of the last base pair in the exon. The variable `NCBIref` is the NCBI reference sequence accession numbers for the coding region in which the exon resides. In `hg_exons` overlapping exons have been combined into a single observation. When exons from genes with different NCBI accession numbers have been combined the variable `NCBIref` will contain multiple accession numbers separated by commas. We note that different accession numbers may exist for transcript variants of the same gene.

Since the exons in `hg_exons` have already been combined into non-overlapping segments, we supply this data frame to `create_slimMap`.

```
# create recombination map for exon-only data using the hg_exons dataset
s_map <- create_slimMap(exon_df = hg_exons)

# print first four rows of s_map
head(s_map, n = 4)
```

```
##   chrom segLength  recRate mutRate  exon simDist endPos
## 1     1     11873 0.00e+00   0e+00 FALSE       1      1
## 2     1       354 1.00e-08   1e-08  TRUE     354    355
## 3     1       385 3.85e-06   0e+00 FALSE       1    356
## 4     1       109 1.00e-08   1e-08  TRUE     109    465
```

The `create_slimMap` function returns a data frame with several variables:

1. `chrom`: the chromosome number.
2. `segLength`: the length of the segment in base pairs. *We assume that segments contain the positions listed in* ***`exonStart`*** *and* ***`exonEnd`****. Therefore, for a combined exon segment,* ***`segLength`*** *is calculated as* ***`exonEnd - exonStart + 1`****.*
3. `recRate`: the per-site per-generation recombination rate. Following [4], segments between exons on the same chromosome are simulated as a single base pair with `rec_rate` equal to recombination rate multiplied by the number of base pairs in the segment. For each chromosome, a single site is created between the last exon on the previous chromosome and the first exon of the current chromosome. This site will have recombination rate 0.5 to accommodate unlinked chromosomes.
4. `mutRate`: the per-site per-generation mutation rate. Since we are interested in exon-only data, the mutation rate outside exons is set to zero.
5. `exon`: logical variable, `TRUE` if segment is an exon and `FALSE` if not an exon.
6. `simDist`: the simulated exon length, in base pairs. When `exon = TRUE`, `simDist = segLength`; however, when `exon = FALSE`, `simDist = 1` since segments between exons on the same chromosome are simulated as a single base pair.
7. `endPos`: The simulated end position, in base pairs, of the segment.

The first row in the output above contains information about the genetic segment **before the first exon** on chromosome 1. Referring back to the `hg_exons` data we see that the first exon begins at position 11,874. Since the region before the first exon contains 11,873 base pairs `segLength = 11873` for this region. To avoid unnecessary computation in SLiM, `simDist = 1` so that this non-exonic region is simulated as a single base pair. The recombination rate for this segment is set to zero. However, the first non-exon segment for the next chromosome will be 0.5 so that chromosomes are unlinked. The mutation rate for this non-exon segment is set to zero since we are interested in exon-only data.

The second row of the output catalogs information for the first exon on chromosome 1. This exon contains 354 base pairs, hence `simDist = 354` for this segment. The per-site per-generation mutation and recombination rates for this exon are both 1E-08.

The third row contains the information for the second non-exon segment on chromosome 1. This segment contains 385 base pairs. Notice that like the first non-exon segment (i.e. row 1) we simulate this segment as a single site, i.e. `simDist = 1`. However, we set the recombination rate to `segLength` multiplied by the argument `recomb_rate` (i.e. 385*1E-8 = 3.85E-6).

Only three of the variables returned by `create_slimMap` are required by SLiM to simulate exon-only data: `recRate`, `mutRate`, and `endPos`. The other variables seen in the output above are used by our `read_slim` function to remap mutations to their correct positions when importing SLiM data to `R` (see section 3.2).

SLiM is written in a scripting language called Eidos. Unlike an `R` array, the first position in an Eidos array is 0. Therefore, we must shift the variable `endPos` forward 1 unit before supplying this data to SLiM.

```
# restrict output to the variables required by SLiM
slimMap <- s_map[, c("recRate", "mutRate", "endPos")]

# shift endPos up by one unit
slimMap$endPos <- slimMap$endPos - 1

# print first four rows of slimMap
head(slimMap, n = 4)
```

```
##    recRate mutRate endPos
## 1 0.00e+00   0e+00      0
## 2 1.00e-08   1e-08    354
## 3 3.85e-06   0e+00    355
## 4 1.00e-08   1e-08    464
```

We use SLiM to generate neutral mutations in exons and allowed them to accumulate over 44,000 generations, as in [4]. However, SLiM is incredibly versatile and can accommodate many different types of simulations. The creators of SLiM provide excellent resources and documentation to simulate forwards-in-time evolutionary data, which can be found at the SLiM website: https://messerlab.org/slim/.

Import SLiM Data to `R`

To import SLiM data to `R`, we provide the `read_slim` function, which has been tested for SLiM versions 2.0-3.1. Presently, the `read_slim` function is only appropriate for data produced by the outputFull() method in SLiM. We do not support output in MS or VCF data format (i.e. produced by outputVCFsample() or outputMSSample() in SLiM).

Our `read_slim` function has four arguments:

1. `file_path`: The file path of the .txt output file created by the outputFull() method in SLiM.
2. `recomb_map`: (Optional) A recombination map of the same format as the data frame returned by `create_slimMap`. Users who followed the instructions in section 3.1, may supply the output from `create_slimMap` to this argument.
3. `keep_maf`: The largest allele frequency for retained SNVs, by default `keep_maf = 0.01`. All variants with allele frequency greater than `keep_maf` will be removed. Please note that removing common variants is recommended for large data sets due to the limitations of data allocation in R.
4. `recode_recurrent` A logical argument that indicates if recurrent SNVs at the same position should be catalogued as single observation; by default, `recode_recurrent = TRUE`.
5. `pathway_df`: (Optional) A data frame that contains the positions for each exon in a pathway of interest. This data frame must contain the variables `chrom`, `exonStart`, and `exonEnd`. *We expect that `pathwayDF` does not contain any overlapping segments. Users may combine overlapping exons into a single observation with our `combine_exons` function (see appendix for details).*

To clarify, the argument `recomb_map` is used to remap mutations to their actual locations and chromosomes. This is necessary when data have been simulated over non-contiguous regions such as exon-only data. If

`recomb_map` is not provided, we assume that the SNV data have been simulated over a contiguous segment starting with the first base pair on chromosome 1.

In addition to reducing the size of the data, the argument `keep_maf` has practicable applicability. In family-based studies, common SNVs are generally filtered out prior to analysis. Users who intend to study common variants in addition to rare variants may need to run chromosome specific analyses to allow for allocation of large data sets in `R`.

When `TRUE`, the logical argument `recode_recurrent` indicates that recurrent SNVs should be recorded as a single observation. SLiM can model many types of mutations; e.g. neutral, beneficial, and deleterious. When different types of mutations occur at the same position carriers will experience different fitness effects depending on the carried mutation. However, when mutations at the same location have the same fitness effects, they represent a recurrent mutation. Even so, SLiM stores recurrent mutations separately and calculates their prevalence independently. When the argument `recode_recurrent = TRUE` we store recurrent mutations as a single observation and calculate the alternate allele frequency based on their combined prevalence. This convention allows for both reduction in storage and correct estimation of the alternate allele frequency of the mutation. Users who prefer to store recurrent mutations from independent lineages as unique entries should set `recode_recurrent = FALSE`.

The data frame `pathway_df` allows users to identify SNVs located within a pathway of interest when importing the data. For example, consider the apoptosis sub-pathway centered about the *TNFSF10* gene based on the 25 genes that have the highest interaction with this gene in the UCSC Genome Browser's Gene Interaction Tool [5, 6, 10]. The data for this sub-pathway are contained in the `hg_apopPath` data set.

```
# load the hg_apopPath data
data("hg_apopPath")

# View the first 4 observations of hg_apopPath
head(hg_apopPath, n = 4)
```

```
##   chrom exonStart   exonEnd
## 1     1 155689091 155689243
## 2     1 155709033 155709125
## 3     1 155709773 155709824
## 4     1 155717006 155717128
##                                                                    NCBIref
## 1 NM_001199851.1, NM_001199850.1, NM_001199849.1, NM_004632.3, NM_033657.2
## 2                                                             NM_001199849.1
## 3 NM_001199851.1, NM_001199850.1, NM_001199849.1, NM_004632.3, NM_033657.2
## 4                 NM_001199850.1, NM_001199849.1, NM_004632.3, NM_033657.2
##   gene
## 1 DAP3
## 2 DAP3
## 3 DAP3
## 4 DAP3
```

The `hg_apopPath` data set is similar the `hg_exons` data set discussed in section 3.1. However, `hg_apopPath` only catalogs the positions of exons contained in our sub-pathway.

The following code demonstrates how the `read_slim` function would be called if `create_slimMap` was used to create the recombination map for SLiM as in section 3.1.

```
# Let's suppose the output is saved in the
# current working directory and is named "slimOut.txt".
# We import the data using the read_slim function.
s_out <- read_slim(file_path  = "slimOut.txt",
                   recomb_map = create_slimMap(hg_exons),
                   pathway_df = hg_apopPath)
```

The `read_slim` function returns an object of class `SNVdata`. Objects of class `SNVdata` inherit from lists. The first item is a sparse matrix named `Haplotypes`, which contains the haplotype data for unrelated, diploid individuals. The second item is a data frame named `Mutations`, which catalogs the SNVs contained in `Haplotypes`.

We note that importing SLiM data can be time-consuming; importing exon-only mutation data simulated over the 22 human autosomes for a sample of 10,000 individuals took approximately 4 mins on a Windows OS with an i7-4790 @ 3.60GHz and 12 GB of RAM. For the purpose of demonstration, we use the `EXhaps` and `EXmuts` and data sets provided by `SimRVSequences`.

The `EXhaps` data set represents the sparse matrix `Haplotypes` in the `SNVdata` object returned by `read_slim`, and the `EXmuts` data set represents the `Mutations` data frame in the `SNVdata` object returned by `read_slim`. We note that since these toy data sets only contain 500 SNVs they are not representative of exon-only data over the entire human genome. Instead, they include 50 SNVs that were randomly sampled from genes in the apoptosis sub-pathway, and 450 SNVs sampled from outside the pathway.

```
# import the EXmuts dataset
data(EXmuts)

# view the first 4 observations of EXmuts
head(EXmuts, n = 4)
```

```
##   colID chrom position   afreq     marker pathwaySNV
## 1     1     1  1731297 0.00875  1_1731297      FALSE
## 2     2     1  3411725 0.00005  1_3411725      FALSE
## 3     3     1  3468928 0.00050  1_3468928      FALSE
## 4     4     1 21578402 0.00025 1_21578402      FALSE
```

The `EXmuts` data frame is used to catalog the SNVs in `EXhaps`. Each row in `EXmuts` represents an SNV. The variable `colID` associates the rows in `EXmuts` to the columns of `EXhaps`, `chrom` is the chromosome the SNV resides in, `position` is the position of the SNV in base pairs, `afreq` is the alternate allele frequency of the SNV, `marker` is a unique character identifier for the SNV, and `pathwaySNV` identifies SNVs located within the sub-pathway as `TRUE`.

```
# import the EXhaps dataset
data(EXhaps)

# dimensions of EXhaps
dim(EXhaps)
```

```
## [1] 20000   500
```

Looking at the output above we see that `EXhaps` contains 20,000 haplotypes with 500 SNVs.

```
# number of rows in EXmuts
nrow(EXmuts)
```

```
## [1] 500
```

Since `EXmuts` catalogs the SNVs in `EXhaps`, `EXmuts` will have 500 rows.

```
# View the first 30 mutations of the first 15 haplotypes in EXhaps
EXhaps[1:15, 1:30]
```

```
## 15 x 30 sparse Matrix of class "dgCMatrix"
##
##  [1,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [2,] . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . .
```

```
##  [3,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [4,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [5,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [6,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [7,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [8,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [9,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## [10,] . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . .
## [11,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## [12,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## [13,] . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . .
## [14,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## [15,] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

From the output above, we see that `EXhaps` is a sparse matrix of class dgCMatrix. The `Matrix` package [1] is used to create objects of this class. Entries of `1` represent a mutated allele, while the wild type is represented as '.'. Recall that we retain SNVs with alternate allele frequency less than or equal to `keep_maf`; hence, the majority of entries represent the wild type allele.

Option 2: Import and Format VCF File Data

Import VCF File to `R`

Importing vcf data using the `vcfR` package [2] is accomplished with the `read.vcfR` function. To import a vcf file the user supplies the file path to the argument `file` of the `read.vcfR` function. For example, suppose the vcf file named "exons_chr21.vcf.gz" was downloaded from https://github.com/simrvprojects/1000-Genomes-Exon-Data/tree/master/Exon%20Data and is stored in a folder named "Data" in the "C" directory. We import "exons_chr21.vcf.gz" using the `read.vcfR` function as follows:

```
# load the vcfR package
library(vcfR)
```

```
##
##    *****       ***   vcfR   ***       *****
##    This is vcfR 1.8.0
##      browseVignettes('vcfR') # Documentation
##      citation('vcfR') # Citation
##    *****       *****      *****       *****
```

```
# specify the file path
vcf_file_path <- "C:/Data/exons_chr21.vcf.gz"

#import vcf file
vcf_chrom21 <- read.vcfR(vcf_file_path)
```

```
# determine the structure of vcf_chrom21
str(vcf_chrom21)
```

```
## Formal class 'vcfR' [package "vcfR"] with 3 slots
## ..@ meta: chr [1:21] "##fileformat=VCFv4.3"
"##FILTER=<ID=PASS,Description=\"All filters passed\">"
"##fileDate=27022019_15h52m43s" "##source=IGSRpipeline" ...
## ..@ fix : chr [1:15524, 1:8] "21" "21" "21" "21" ...
## .. ..- attr(*, "dimnames")=List of 2
## .. .. ..$ : NULL
## .. .. ..$ : chr [1:8] "CHROM" "POS" "ID" "REF" ...
```

```
## ..@ gt : chr [1:15524, 1:2549] "GT" "GT" "GT" "GT" ...
## .. ..- attr(*, "dimnames")=List of 2
## .. .. ..$ : NULL
## .. .. ..$ : chr [1:2549] "FORMAT" "HG00096" "HG00097" "HG00099" ...
```

From the output above, we see that `vcf_chrom21` is an object of class `vcfR` with 3 slots. The first slot `@meta` contains meta data for the vcf file; the second slot `@fix` contains the fixed data, which describes the mutations contained in the vcf file; and the third slot `@gt` contains the genotype data. We discuss these components in detail in the following sections. Specifically, in section section XX we discuss the genotype data and in section XX we discuss the fixed and meta data. For additional information regarding objects of class `vcfR` please execute `help("vcfR-class")` in the console.

Format Haplotype Data

We now demonstrate how to format the genotype data into haplotype data. Recall, that the genotype data is stored in the `@gt` slot of the `vcfR` object.

```
# View the first five rows and columns of the genotype data
vcf_chrom21@gt[1:5, 1:5]
```

```
##      FORMAT HG00096 HG00097 HG00099 HG00100
## [1,] "GT"   "0|0"   "0|0"   "0|0"   "0|0"
## [2,] "GT"   "0|0"   "0|0"   "0|0"   "0|0"
## [3,] "GT"   "0|0"   "0|0"   "0|0"   "0|0"
## [4,] "GT"   "0|1"   "1|1"   "0|0"   "0|0"
## [5,] "GT"   "0|0"   "0|0"   "0|0"   "0|0"
```

From the output above, we see that the first column in `vcf_chrom21@gt` does not contain genotype data, but rather a variable named `FORMAT`. Following this variable, each column in `vcf_chrom21@gt` represents an individual identified by a unique character string, e.g. "HG00096". Each row in `vcf_chrom21@gt` represents a SNV.

```
#View first 4 mutations for the individual with ID "HG00096"
vcf_chrom21@gt[1:4, "HG00096"]
```

```
## [1] "0|0" "0|0" "0|0" "0|1"
```

From the output above, we see that the individual with ID "HG00096" is homozygous for the reference allele, i.e. "0|0", for the first three SNVs in the genotypes data. However, for the fourth SNV this individual is heterozygous for the alternate allele, i.e. "0|1". We note that this data is phased so that the genotypes for each individual are ordered. That is, for a given individual, the first allele may represent the paternally inherited allele while the second may represent the maternally inherited allele and this ordering is consistent in every genotype for this individual.

```
# Remove the variable named "FORMAT"
# and store the resulting data as genos
genos <- vcf_chrom21@gt[, -1]

# View the first 5 mutations (i.e. rows) for
# the first 3 individuals (i.e. columns)
genos[1:5, 1:3]
```

```
##      HG00096 HG00097 HG00099
## [1,] "0|0"   "0|0"   "0|0"
## [2,] "0|0"   "0|0"   "0|0"
## [3,] "0|0"   "0|0"   "0|0"
## [4,] "0|1"   "1|1"   "0|0"
```

```
## [5,] "0|0"   "0|0"   "0|0"
```

To format the genotype data we supply `genos` to the `genos2sparseMatrix` function, which is included in the `SimRVSequences` package. This function may be used to convert phased SNV data for diploid organisms into haplotype data stored as a sparse matrix. We note that this conversion makes use of the methods provided by the `Matrix` package [1].

```
# load the SimRVSequences package
library(SimRVSequences)

# Convert to sparseMatrix
haplotypes <- genos2sparseMatrix(genotypes = genos)
```

We note that `genos2sparseMatrix` transposes the supplied genotype data. Thus, in the returned matrix individuals are stored as rows and mutations are stored as columns. Additionally, since this is a diploid population, each individual will have two rows of data. That is, the haplotype data for the first individual in `genos`, named "HG00096", is stored in rows 1 and 2 of `haplotypes`, the haplotype data for the second individual in `genos`, named "HG00097", is stored in rows 3 and 4 of `haplotypes`, and so on.

```
# View haplotype data for the first
# 3 diploid individuals and first 5 SNVs
haplotypes[1:6, 1:5]
```

```
## 6 x 5 sparse Matrix of class "dgCMatrix"
##
## HG00096 . . . . .
## HG00096 . . . 1 .
## HG00097 . . . 1 .
## HG00097 . . . 1 .
## HG00099 . . . . .
## HG00099 . . . . .
```

From the output above we see that `haplotypes` is a sparse matrix of class `dgCMatrix`. Entries of `1` represent the alternate or mutated allele, while the reference allele or wild type alelle is represented as '`.`'. For additional information regarding objects of class `dgCMatrix` execute `help("dgCMatrix-class")` in the console.

Format Mutation Data

We now demonstrate how to manipulate the fixed data contained in the `vcfR` object. Recall, that the fixed data is stored in the `@fix` slot of the `vcfR` object and is used to describe the SNVs in the genotype data.

```
# View the first three rows of the fixed data
vcf_chrom21@fix[1:3, ]
```

```
## CHROM POS ID REF ALT QUAL FILTER
## [1,] "21" "13349301" NA "T" "C" NA "PASS"
## [2,] "21" "13349303" NA "G" "A" NA "PASS"
## [3,] "21" "13349361" NA "A" "G" NA "PASS"
## INFO
## [1,]
"AC=2;AN=5096;DP=219605;AF=0;EAS_AF=0;EUR_AF=0;AFR_AF=0;AMR_AF=0;SAS_AF=0;EX_TARGET;VT=SNP;NS=2548"
## [2,]
"AC=1;AN=5096;DP=219794;AF=0;EAS_AF=0;EUR_AF=0;AFR_AF=0;AMR_AF=0;SAS_AF=0;EX_TARGET;VT=SNP;NS=2548"
## [3,]
"AC=1;AN=5096;DP=178290;AF=0;EAS_AF=0;EUR_AF=0;AFR_AF=0;AMR_AF=0;SAS_AF=0;EX_TARGET;VT=SNP;NS=2548"
```

From the output above, we see that the fixed data contains several variables. According to http://www.internationalgenome.org/wiki/Analysis/vcf4.0/ the variables above may be interpreted as follows:

1. `CHROM` is the chromosome number of the SNV
2. `POS` is the position, in base pairs, of the SNV
3. `ID`, when available, is a unique identifier for the SNV, e.g. an rs number for a dbSNP.
4. `REF` is the reference allele for the SNV
5. `ALT` is the alternate allele for the SNV
6. `QUAL`, when available, is a quality score for the SNV
7. `FILTER` indicates whether or not the site has passed all filters. When passing all filters `FILTER = PASS`, otherwise `FILTER` is a character string which contains all non-passing filters.
8. `INFO` contains additional information for each SNV. Each variable in filter is separated by a semicolon.

Additional information for the variables contained in `INFO` is contained in the metadata of the vcf object. Recall that meta data in `vcfR` objects is stored in the `@meta` slot.

```
# View the meta data
vcf_chrom21@meta
```

```
## [1] "##fileformat=VCFv4.3"
## [2] "##FILTER=<ID=PASS,Description=\"All filters passed\">"
## [3] "##fileDate=27022019_15h52m43s"
## [4] "##source=IGSRpipeline"
## [5]
"##reference=ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/GRCh38_reference_genome/GRCh3
## [6] "##FORMAT=<ID=GT,Number=1,Type=String,Description=\"Phased Genotype\">"
## [7] "##contig=<ID=21>"
## [8] "##INFO=<ID=AF,Number=A,Type=Float,Description=\"Estimated allele
frequency in the range (0,1)\">"
## [9] "##INFO=<ID=AC,Number=A,Type=Integer,Description=\"Total number of
alternate alleles in called genotypes\">"
## [10] "##INFO=<ID=NS,Number=1,Type=Integer,Description=\"Number of samples
with data\">"
## [11] "##INFO=<ID=AN,Number=1,Type=Integer,Description=\"Total number of
alleles in called genotypes\">"
## [12] "##INFO=<ID=EAS_AF,Number=A,Type=Float,Description=\"Allele frequency
in the EAS populations calculated from AC and AN, in the range (0,1)\">"
## [13] "##INFO=<ID=EUR_AF,Number=A,Type=Float,Description=\"Allele frequency
in the EUR populations calculated from AC and AN, in the range (0,1)\">"
## [14] "##INFO=<ID=AFR_AF,Number=A,Type=Float,Description=\"Allele frequency
in the AFR populations calculated from AC and AN, in the range (0,1)\">"
## [15] "##INFO=<ID=AMR_AF,Number=A,Type=Float,Description=\"Allele frequency
in the AMR populations calculated from AC and AN, in the range (0,1)\">"
## [16] "##INFO=<ID=SAS_AF,Number=A,Type=Float,Description=\"Allele frequency
in the SAS populations calculated from AC and AN, in the range (0,1)\">"
## [17] "##INFO=<ID=VT,Number=.,Type=String,Description=\"indicates what type
of variant the line represents\">"
## [18] "##INFO=<ID=EX_TARGET,Number=0,Type=Flag,Description=\"indicates
whether a variant is within the exon pull down target boundaries\">"
## [19] "##INFO=<ID=DP,Number=1,Type=Integer,Description=\"Approximate read
depth; some reads may have been filtered\">"
## [20] "##bcftools_viewVersion=1.9-162-g33ecfe8+htslib-1.9-150-gc76b3b2"
## [21] "##bcftools_viewCommand=view --regions-file 1KG.exons.merged.bed
--types snps --min-alleles 2 --max-alleles 2 --include FILTER=\"PASS\"
--output-type z --output-file exons_chr21.vcf.gz
ALL.chr21.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz;
```

```
Date=Mon Jun 3 14:54:02 2019"
```

From the output above, we see that the first variable contained in `INFO` is stored in the eighth item of the meta data and is named `AF`. From the description, we see that `AF` represents the estimated alternate allele frequency. From the type, we see that this is a float variable.

To format the fixed data, including the `INFO` variables, the `vcfR` package offers the `vcfR2tidy` function. We now demonstrate how to use `vcfR2tidy` to format the fixed data. For additional information on the `vcfR2tidy` function please execute `help(vcfR2tidy)` in the console.

```
# Extract and store the mutation data using vcfR2tidy
#
# NOTE: setting info_only = TRUE since we do not need
# to re-process the genotype data
#
# To include INFO variables, we supply a list of variable
# names to "info_fields".  The names of these variables must
# corresponds with the INFO variable names defined in the meta data.
# We specify each variable type, by variable name, using the "info_types"
# argument.  Each INFO variable type is available in the meta data. Note
# that the info types for float varibles are set to numeric, i.e. "n".
#
muts <- vcfR2tidy(vcf_chrom21,
                  info_only = TRUE,
                  info_fields = c("AF", "AC", "NS", "AN",
                                  "EAS_AF", "EUR_AF", "AFR_AF",
                                  "AMR_AF", "SAS_AF", "DP"),
                  info_types = c(AF = "n", AC = "i", NS = "i", AN = "i",
                                 EAS_AF = "n", EUR_AF = "n", AFR_AF = "n",
                                 AMR_AF = "n", SAS_AF = "n", DP = "i"))

# Determine the structure of muts
str(muts)
```

```
## List of 2
## $ fix :Classes 'tbl_df', 'tbl' and 'data.frame': 15524 obs. of 17 variables:
## ..$ CHROM : chr [1:15524] "21" "21" "21" "21" ...
## ..$ POS : int [1:15524] 13349301 13349303 13349361 13544860 13939343
13939374 13939380 13939405 13939410 13939453 ...
## ..$ ID : chr [1:15524] NA NA NA NA ...
## ..$ REF : chr [1:15524] "T" "G" "A" "C" ...
## ..$ ALT : chr [1:15524] "C" "A" "G" "T" ...
## ..$ QUAL : num [1:15524] NA NA NA NA NA NA NA NA NA NA ...
## ..$ FILTER: chr [1:15524] "PASS" "PASS" "PASS" "PASS" ...
## ..$ AF : num [1:15524] 0 0 0 0.25 0 0 0 0 0 0 ...
## ..$ AC : int [1:15524] 2 1 1 1297 1 6 2 6 1 3 ...
## ..$ NS : int [1:15524] 2548 2548 2548 2548 2548 2548 2548 2548 2548 2548 ...
## ..$ AN : int [1:15524] 5096 5096 5096 5096 5096 5096 5096 5096 5096 5096 ...
## ..$ EAS_AF: num [1:15524] 0 0 0 0.3 0 0 0 0 0 0 ...
## ..$ EUR_AF: num [1:15524] 0 0 0 0.35 0 0 0 0 0 0 ...
## ..$ AFR_AF: num [1:15524] 0 0 0 0.13 0 0 0 0 0 0 ...
## ..$ AMR_AF: num [1:15524] 0 0 0 0.24 0 0 0 0 0 0 ...
## ..$ SAS_AF: num [1:15524] 0 0 0 0.28 0 0 0 0.01 0 0 ...
## ..$ DP : int [1:15524] 219605 219794 178290 375437 100283 106811 106028
111046 109940 86510 ...
## $ meta:Classes 'tbl_df', 'tbl' and 'data.frame': 10 obs. of 5 variables:
```

```
## ..$ Tag : chr [1:10] "INFO" "INFO" "INFO" "INFO" ...
## ..$ ID : chr [1:10] "AF" "AC" "NS" "AN" ...
## ..$ Number : chr [1:10] "A" "A" "1" "1" ...
## ..$ Type : chr [1:10] "Float" "Integer" "Integer" "Integer" ...
## ..$ Description: chr [1:10] "Estimated allele frequency in the range (0,1)"
"Total number of alternate alleles in called genotypes" "Number of samples with
data" "Total number of alleles in called genotypes" ...
```

From the output above, we see that `muts` contains the fixed data, including `INFO` variables as a `tbl_df` which inherits from objects of class `data.frame`.

```
#View the first three rows of the fixed data in muts
head(muts$fix, n = 3)
```

```
## # A tibble: 3 x 17
##   CHROM    POS ID    REF   ALT    QUAL FILTER    AF    AC    NS    AN
##   <chr>  <int> <chr> <chr> <chr> <dbl> <chr>  <dbl> <int> <int> <int>
## 1 21    1.33e7 <NA>  T     C        NA PASS       0     2  2548  5096
## 2 21    1.33e7 <NA>  G     A        NA PASS       0     1  2548  5096
## 3 21    1.33e7 <NA>  A     G        NA PASS       0     1  2548  5096
## # ... with 6 more variables: EAS_AF <dbl>, EUR_AF <dbl>, AFR_AF <dbl>,
## #   AMR_AF <dbl>, SAS_AF <dbl>, DP <int>
```

Notice that the alternate allele frequency, `AF`, is rounded to two decimal places. Users who prefer to retain more that 2 significant digits can re-calculate `AF` as the alternate allele count, i.e. `AC`, divided by the alelle number values, i.e. `AN`. In this context, the allele number is the total number of samples with non-missing data at the specified marker. See http://www.internationalgenome.org/category/allele-frequency/ for additional information.

```
#Recalculate the alternate allele frequency
muts$fix$AF <- muts$fix$AC/muts$fix$AN

#View the first three rows of the fixed data in muts
head(muts$fix, n = 3)
```

```
## # A tibble: 3 x 17
##   CHROM    POS ID    REF   ALT    QUAL FILTER      AF    AC    NS    AN
##   <chr>  <int> <chr> <chr> <chr> <dbl> <chr>    <dbl> <int> <int> <int>
## 1 21    1.33e7 <NA>  T     C        NA PASS   3.92e-4     2  2548  5096
## 2 21    1.33e7 <NA>  G     A        NA PASS   1.96e-4     1  2548  5096
## 3 21    1.33e7 <NA>  A     G        NA PASS   1.96e-4     1  2548  5096
## # ... with 6 more variables: EAS_AF <dbl>, EUR_AF <dbl>, AFR_AF <dbl>,
## #   AMR_AF <dbl>, SAS_AF <dbl>, DP <int>
```

Our `sim_RVstudy` function expects mutation data to be formatted as a data frame. Additionally, mutation data supplied to `sim_RVstudy` must contain the following variables:

1. `colID`: a numeric variable which associates the rows in the `mutations` data frame to the columns in the `haplotypes` matrix.
2. `chrom`: represents the chromosome in which the SNV resides.
3. `position`: represents the position of the SNV in base pairs.
4. `afreq`: (Optional) the alternate allele frequency of the SNV.

From the previous discussion, we know that the fixed data returned by `vcf2tidy` contains the variables `chrom`, `position`, and `afreq`. To make the output from the `vcf2tidy` function compatible with the format expected for the `sim_RVstudy` function we make the following changes.

```r
# store the fixed item in muts data as dataframe
mutations <- as.data.frame(muts$fix)
```

```r
# Create the variable colID to identify the column position of the mutation.
# NOTE: Since mutations are already in the correct order, we accomplish this task
# using the seq function.  Since the mutations are stored in the columns of
# the haplotypes matrix we determine the length of our sequence as the number of
# columns in haplotypes.
mutations$colID <- seq(1:dim(haplotypes)[2])

# View the first three rows of mutations
head(mutations, n = 3)
```

```
##   CHROM      POS   ID REF ALT QUAL FILTER           AF AC   NS   AN EAS_AF
## 1    21 13349301 <NA>   T   C   NA   PASS 0.0003924647  2 2548 5096      0
## 2    21 13349303 <NA>   G   A   NA   PASS 0.0001962323  1 2548 5096      0
## 3    21 13349361 <NA>   A   G   NA   PASS 0.0001962323  1 2548 5096      0
##   EUR_AF AFR_AF AMR_AF SAS_AF     DP colID
## 1      0      0      0      0 219605     1
## 2      0      0      0      0 219794     2
## 3      0      0      0      0 178290     3
```

```r
# Rename columns for consistency with expected format.
# "AF" should be renamed "afreq",
# "CHROM" should be renamed "chrom",
# and "POS" should be renamed "position".
colnames(mutations)[c(1, 2, 8)] = c("chrom", "position", "afreq")

# View the first three rows of mutations
head(mutations, n = 3)
```

```
##   chrom position   ID REF ALT QUAL FILTER        afreq AC   NS   AN EAS_AF
## 1    21 13349301 <NA>   T   C   NA   PASS 0.0003924647  2 2548 5096      0
## 2    21 13349303 <NA>   G   A   NA   PASS 0.0001962323  1 2548 5096      0
## 3    21 13349361 <NA>   A   G   NA   PASS 0.0001962323  1 2548 5096      0
##   EUR_AF AFR_AF AMR_AF SAS_AF     DP colID
## 1      0      0      0      0 219605     1
## 2      0      0      0      0 219794     2
## 3      0      0      0      0 178290     3
```

Create an Object of Class `SNVdata`

The constructor function for objects of class `SNVdata` has three arguments:

1. `Haplotypes` A sparse matrix of haplotype data, which contains the haplotypes for unrelated individuals representing the founder population.
2. `Mutations` A data frame that catalogs the SNVs in the columns of the `Haplotypes` matrix. `Mutations` must include the following variables:
   - `colID`: a numeric variable which associates the rows in `Mutations` to the columns in `Haplotypes`.
   - `chrom`: represents the chromosomal in which the SNV resides.
   - `position`: represents the chromosomal position of the SNV in base pairs.
   - `afreq`: (Optional) the alternate allele frequency of the SNV.

3. `Samples` (Optional) this argument allows users to provide data for the individuals whose haplotypes are stored in `Haplotypes`.

The sample data, which describes the individuals whose genotypes are contained in the vcf data for chromosome 21 (i.e. from sections XX and XX), may be obtained as follows:

```
file_path <- "https://raw.githubusercontent.com/simrvprojects/1000-Genomes-Exon-Data/master/Vignette%20I
SampleInfo <- read.csv(file_path)
```

```
#View the first 6 lines of the sample data
head(SampleInfo)
```

```
##    Sample Family.ID Population          Population.Description Gender
## 1 HG00096   HG00096        GBR British in England and Scotland   male
## 2 HG00097   HG00097        GBR British in England and Scotland female
## 3 HG00098   HG00098        GBR British in England and Scotland   male
## 4 HG00099   HG00099        GBR British in England and Scotland female
## 5 HG00100   HG00100        GBR British in England and Scotland female
## 6 HG00101   HG00101        GBR British in England and Scotland   male
##   Relationship Unexpected.Parent.Child Non.Paternity Siblings Grandparents
## 1
## 2
## 3
## 4
## 5
## 6
##   Avuncular Half.Siblings Unknown.Second.Order Third.Order Other.Comments
## 1                                                                     NA
## 2                                                                     NA
## 3                                                                     NA
## 4                                                                     NA
## 5                                                                     NA
## 6                                                                     NA
```

From the output above, we see that `SampleInfo` describes the individuals contained in the vcf file. For each individual, the cataloged information includes population descriptions as well as information on any relations that are included in this data set. We caution users against using haplotype data that includes related individuals as this may result in cryptic relatedness among pedigree founders. To avoid this, we must remove any relatives from the founder haplotype data. The `SampleInfo` data set includes 42 individuals who have been listed as third order or higher relatives. We reduced `SampleInfo` by randomly sampling one relative from each set of related individuals. This resulted in the removal of 22 individuals. Additionally, we removed any individuals from the `SampleInfo` whose genotypes were not contained in the vcf data. This resulted in the removal of additional 930 individuals. Users interested in following the sample-reduction process may find our script at: https://github.com/simrvprojects/SimRVSequences/blob/master/data-raw/format_sample_data.R.

To import the reduced sample data, we execute the following commands:

```
file_path <- 'https://raw.githubusercontent.com/simrvprojects/1000-Genomes-Exon-Data/master/Formatted-SI
SampleData <- read.csv(file_path)

# view the first 6 lines of SampleData
head(SampleData)
```

```
##    Sample Population          Population.Description Gender
## 1 HG00096        GBR British in England and Scotland   male
## 2 HG00097        GBR British in England and Scotland female
```

```
## 3 HG00099        GBR British in England and Scotland female
## 4 HG00100        GBR British in England and Scotland female
## 5 HG00101        GBR British in England and Scotland   male
## 6 HG00102        GBR British in England and Scotland female
```

Next, we must reduce the haplotype data to include only the individuals who are contained in SampleData. We achieve this as follows:

```
# View dimensions of haplotypes
dim(haplotypes)
```

```
## [1]  5096 15524
```

From the output above, we see that **haplotypes** contains 5096 haplotypes spanning 15,524 SNVs. Since these are the haplotypes of diploid organisms this matrix contains data for 2048 individuals.

```
# Recall that the row names in the haplotypes matrix are the sample IDs
row.names(haplotypes)[1:5]
```

```
## [1] "HG00096" "HG00096" "HG00097" "HG00097" "HG00099"
```

```
# Reduce the haplotype data to contain the
# unrelated individuals described in SampleData
haplotypes <- haplotypes[row.names(haplotypes) %in% SampleData$Sample, ]

# View the new dimensions of haplotypes
dim(haplotypes)
```

```
## [1]  5052 15524
```

Comparing the new dimensions to the original dimensions, we see that 44 rows have been removed, which corresponds to the removal of 22 diploid individuals.

Finally, we are ready to create an object of class **SNVdata**. To create this object we supply the required arguments to the **SNVdata** constructor function as follows.

```
#create SNVdata object for chromosome 21 without sample or meta data
SNVdata_chrom21 <- SNVdata(Haplotypes = haplotypes,
                           Mutations = mutations,
                           Samples = SampleData)
str(SNVdata_chrom21)
```

```
## List of 3
## $ Haplotypes:Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
## .. ..@ i : int [1:1452481] 2122 2821 282 1334 1 2 3 9 10 11 ...
## .. ..@ p : int [1:15525] 0 2 3 4 1293 1294 1299 1301 1307 1308 ...
## .. ..@ Dim : int [1:2] 5052 15524
## .. ..@ Dimnames:List of 2
## .. .. ..$ : chr [1:5052] "HG00096" "HG00096" "HG00097" "HG00097" ...
## .. .. ..$ : NULL
## .. ..@ x : num [1:1452481] 1 1 1 1 1 1 1 1 1 1 ...
## .. ..@ factors : list()
## $ Mutations :'data.frame': 15524 obs. of 19 variables:
## ..$ chrom : chr [1:15524] "21" "21" "21" "21" ...
## ..$ position: int [1:15524] 13349301 13349303 13349361 13544860 13939343
13939374 13939380 13939405 13939410 13939453 ...
## ..$ ID : chr [1:15524] NA NA NA NA ...
## ..$ REF : chr [1:15524] "T" "G" "A" "C" ...
## ..$ ALT : chr [1:15524] "C" "A" "G" "T" ...
```

```
## ..$ QUAL : num [1:15524] NA NA NA NA NA NA NA NA NA NA ...
## ..$ FILTER : chr [1:15524] "PASS" "PASS" "PASS" "PASS" ...
## ..$ afreq : num [1:15524] 0.000392 0.000196 0.000196 0.254513 0.000196 ...
## ..$ AC : int [1:15524] 2 1 1 1297 1 6 2 6 1 3 ...
## ..$ NS : int [1:15524] 2548 2548 2548 2548 2548 2548 2548 2548 2548 2548 ...
## ..$ AN : int [1:15524] 5096 5096 5096 5096 5096 5096 5096 5096 5096 5096 ...
## ..$ EAS_AF : num [1:15524] 0 0 0 0.3 0 0 0 0 0 0 ...
## ..$ EUR_AF : num [1:15524] 0 0 0 0.35 0 0 0 0 0 0 ...
## ..$ AFR_AF : num [1:15524] 0 0 0 0.13 0 0 0 0 0 0 ...
## ..$ AMR_AF : num [1:15524] 0 0 0 0.24 0 0 0 0 0 0 ...
## ..$ SAS_AF : num [1:15524] 0 0 0 0.28 0 0 0 0.01 0 0 ...
## ..$ DP : int [1:15524] 219605 219794 178290 375437 100283 106811 106028
111046 109940 86510 ...
## ..$ colID : int [1:15524] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ marker : chr [1:15524] "21_13349301" "21_13349303" "21_13349361"
"21_13544860" ...
## $ Samples :'data.frame': 2526 obs. of 4 variables:
## ..$ Sample : Factor w/ 2526 levels "HG00096","HG00097",..: 1 2 3 4 5 6 7 8 9
10 ...
## ..$ Population : Factor w/ 26 levels "ACB","ASW","BEB",..: 11 11 11 11 11 11
11 11 11 11 ...
## ..$ Population.Description: Factor w/ 26 levels "African Ancestry in
Southwest US",..: 4 4 4 4 4 4 4 4 4 4 ...
## ..$ Gender : Factor w/ 2 levels "female","male": 2 1 1 1 2 1 2 1 2 1 ...
## - attr(*, "class")= chr [1:2] "SNVdata" "list"
```

We note that `SNVdata` objects can be much smaller than the original `vcfR` objects for the same data set. To determine the size of each of these objects we use the `object_size` function from the `pryr` package.

```
library(pryr)

# Determine size of vcfR object for chromosome 21
object_size(vcf_chrom21)
```

```
## 321 MB
```

```
# Determine size of SNVdata object for chromosome 21
object_size(SNVdata_chrom21)
```

```
## 20.7 MB
```

We note that while the size reduction between the `vcfR` and `SNVdata` object for this chromosome is quite drastic, the size of the `SNVdata` object is very similar to the size of the zipped vcf file for chromosome 21. Even so, this format may be very useful for large, sparse, genomic data sets in R.

Option 3: Import Pre-Formatted 1000 Genomes Project Exon-Data

The SNVdata objects for each chromosome are stored in the "Formatted-SNVdata" folder in the GitHub repository.

The `import_SNVdata` function allows users to import formatted SNV data for each chromosome. The `import_SNVdata` function has two arguments:

1. `chrom` A number or list of numbers that specify which non-sex chromosomes to import.
2. `pathway_df` (Optional) A data frame that contains the positions for each exon in a pathway of interest. This data frame must contain the variables `chrom`, `exonStart`, and `exonEnd`. *We expect that* ***`pathwayDF`*** *does not contain any overlapping segments. Users may combine overlapping exons*

into a single observation with our `combine_exons` *function. For additional information regarding the* `combine_exons` *function please execute* `help(combine_exons)` *in the console.*

```
# load the hg_apopPath dataset
data("hg_apopPath")

#import SNV data for chromosomes 21 and 22 and identify
#SNVs located in the pathway defined by hg_apopPath
EXdata = import_SNVdata(chrom = 21:22,
                        pathway_df = hg_apopPath)

# determine the structure of EXdata
str(EXdata)
```

```
## List of 3
## $ Haplotypes:Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
## .. ..@ i : int [1:4217953] 2122 2821 282 1334 1 2 3 9 10 11 ...
## .. ..@ p : int [1:49526] 0 2 3 4 1293 1294 1299 1301 1307 1308 ...
## .. ..@ Dim : int [1:2] 5052 49525
## .. ..@ Dimnames:List of 2
## .. .. ..$ : chr [1:5052] "HG00096" "HG00096" "HG00097" "HG00097" ...
## .. .. ..$ : NULL
## .. ..@ x : num [1:4217953] 1 1 1 1 1 1 1 1 1 1 ...
## .. ..@ factors : list()
## $ Mutations :'data.frame': 49525 obs. of 18 variables:
## ..$ colID : int [1:49525] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ chrom : int [1:49525] 21 21 21 21 21 21 21 21 21 21 ...
## ..$ position : int [1:49525] 13349301 13349303 13349361 13544860 13939343
13939374 13939380 13939405 13939410 13939453 ...
## ..$ REF : chr [1:49525] "T" "G" "A" "C" ...
## ..$ ALT : chr [1:49525] "C" "A" "G" "T" ...
## ..$ FILTER : chr [1:49525] "PASS" "PASS" "PASS" "PASS" ...
## ..$ afreq : num [1:49525] 0.000392 0.000196 0.000196 0.254513 0.000196 ...
## ..$ AC : int [1:49525] 2 1 1 1297 1 6 2 6 1 3 ...
## ..$ NS : int [1:49525] 2548 2548 2548 2548 2548 2548 2548 2548 2548 2548 ...
## ..$ AN : int [1:49525] 5096 5096 5096 5096 5096 5096 5096 5096 5096 5096 ...
## ..$ EAS_AF : num [1:49525] 0 0 0 0.3 0 0 0 0 0 0 ...
## ..$ EUR_AF : num [1:49525] 0 0 0 0.35 0 0 0 0 0 0 ...
## ..$ AFR_AF : num [1:49525] 0 0 0 0.13 0 0 0 0 0 0 ...
## ..$ AMR_AF : num [1:49525] 0 0 0 0.24 0 0 0 0 0 0 ...
## ..$ SAS_AF : num [1:49525] 0 0 0 0.28 0 0 0 0.01 0 0 ...
## ..$ DP : int [1:49525] 219605 219794 178290 375437 100283 106811 106028
111046 109940 86510 ...
## ..$ marker : chr [1:49525] "21_13349301" "21_13349303" "21_13349361"
"21_13544860" ...
## ..$ pathwaySNV: logi [1:49525] FALSE FALSE FALSE FALSE FALSE FALSE ...
## $ Samples :'data.frame': 2526 obs. of 4 variables:
## ..$ Sample : chr [1:2526] "HG00096" "HG00097" "HG00099" "HG00100" ...
## ..$ Population : chr [1:2526] "GBR" "GBR" "GBR" "GBR" ...
## ..$ Population.Description: chr [1:2526] "British in England and Scotland"
"British in England and Scotland" "British in England and Scotland" "British in
England and Scotland" ...
## ..$ Gender : chr [1:2526] "male" "female" "female" "female" ...
## - attr(*, "class")= chr [1:2] "SNVdata" "list"
```

Note that any SNVs in the pathway contained in the `hg_apopPath` data will be identified by the variable `pathwaySNV` in the `Mutations` data frame of `EXdata`.

```
# View the first 6 SNVs contained in the pathway of interest
head(EXdata$Mutations[EXdata$Mutations$pathwaySNV, ])
```

```
##      colID chrom position REF ALT FILTER        afreq AC   NS   AN EAS_AF
## 3500  3412    21 33325047   G   A   PASS 0.0001962323  1 2548 5096      0
## 3501  3413    21 33325055   G   A   PASS 0.0001962323  1 2548 5096      0
## 3505  3417    21 33335562   G   A   PASS 0.0001962323  1 2548 5096      0
## 3506  3418    21 33335577   A   C   PASS 0.0001962323  1 2548 5096      0
## 3507  3419    21 33335578   A   G   PASS 0.0003924647  2 2548 5096      0
## 3508  3420    21 33335604   G   A   PASS 0.0003924647  2 2548 5096      0
##      EUR_AF AFR_AF AMR_AF SAS_AF     DP      marker pathwaySNV
## 3500      0      0      0      0 120758 21_33325047       TRUE
## 3501      0      0      0      0 120822 21_33325055       TRUE
## 3505      0      0      0      0 338954 21_33335562       TRUE
## 3506      0      0      0      0 357644 21_33335577       TRUE
## 3507      0      0      0      0 358184 21_33335578       TRUE
## 3508      0      0      0      0 378199 21_33335604       TRUE
```

Select Pool of Causal Variants

Our next task is to decide which mutations will be modeled as causal variants. We will focus on a pathway approach, which allows for causal variation in a pathway, or a set of related genes. This approach may also be used to model families which segregate different rare variants in the same gene. Furthermore, we note that implementation of a single causal variant is equivalent to a pathway containing a single gene with a single mutation.

For `SimRVpedigree`'s ascertainment process to be valid we require that the probability that a random individual carries a risk allele is small, i.e ≤ 0.002 [9]. Consider a set of n causal SNVs with alternate allele frequencies $p_1, p_2, ..., p_n$, and let $p_{tot} = \sum_{i=1}^{n} p_i$. For sufficiently rare SNVs, we may compute the cumulative carrier probability as $p_{tot}^2 + 2 * p_{tot}(1 - p_{tot})$. To satisfy the condition $p_{carrier} \leq 0.002$, we require that $1 - (1 - p_{tot})^2 \leq 0.002$, or that $p_{tot} \leq 0.001$.

We now demonstrate how users may create a new variable that identifies causal SNVs using the `EXmuts` data set from section 3.1.2.

```
# Recall that the variable pathwaySNV, which was created in
# section 3.2, is TRUE for any SNVs in the pathway of interest.
# Here we tabulate the alternate allele frequencies
# of the SNVs located in our pathway.
table(EXmuts$afreq[EXmuts$pathwaySNV == TRUE])
```

```
##
##   5e-05   1e-04 0.00015   2e-04 0.00035   4e-04 0.00045   5e-04 0.00055
##      12       4       6       1       1       1       1       1       1
## 0.00065   7e-04 0.00075   8e-04   9e-04 0.00095   0.001 0.00125  0.0014
##       1       1       1       1       2       1       1       1       1
## 0.00145 0.00165   0.002 0.00205  0.0021 0.00365 0.00505 0.00515  0.0052
##       1       1       1       1       1       1       1       1       1
## 0.00585 0.00635 0.00925
##       1       1       1
```

The output above tabulates variants in our pathway by their alternate allele frequencies. For example, our pathway contains 12 SNVs with alternate allele frequency 5e-05, 4 SNVs with alternate allele frequency 1e-04,

6 SNVs with alternate allele frequency 0.00015, and so on.

To obtain a pool of variants with $p_{tot} \sim 0.001$, let's choose the 12 SNVs with alternate allele frequency 5e-05, and 4 SNVs with alternate allele frequency 1e-4 to be our causal SNVs, so that $p_{tot} = 12(5 \times 10^{-5}) + 4(1 \times 10^{-4}) = 0.001$.

```
# Create the variable 'is_CRV', which is TRUE for SNVs
# in our pathway with alternate allele frequency 5e-05 or 1e-04,
# and FALSE otherwise
EXmuts$is_CRV <- EXmuts$pathwaySNV & EXmuts$afreq %in% c(5e-5, 1e-04)

# verify that sum of the alternate allele
# frequencies of causal SNVs is 0.001
sum(EXmuts$afreq[EXmuts$is_CRV])
```

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

```
# determine the number of variants in our pool of causal variants
sum(EXmuts$is_CRV)
```

```
## [1] 16
```

From the output above we see that we have selected a pool of 16 causal variants from our pathway with a cumulative allele frequency of 0.001.

```
# view first 4 observations of EXmuts
head(EXmuts, n = 4)
```

```
##   colID chrom position   afreq     marker pathwaySNV is_CRV
## 1     1     1  1731297 0.00875  1_1731297      FALSE  FALSE
## 2     2     1  3411725 0.00005  1_3411725      FALSE  FALSE
## 3     3     1  3468928 0.00050  1_3468928      FALSE  FALSE
## 4     4     1 21578402 0.00025 1_21578402      FALSE  FALSE
```

Observe from the output above that `EXmuts` now contains the variable `is_CRV`, which identifies potential causal rare variants. The `sim_RVstudy` function (discussed in section 5) samples a single causal variant for each family, according to its allele frequency in the population, from the SNVs for which `is_CRV = TRUE`. Hence, different families may segregate different causal rare variants. Upon identifying the familial cRV we then sample haplotypes for each founder from the distribution of haplotypes conditioned on the founder's cRV status. This ensures that the cRV is introduced by the correct founder.

Prepare Pedigree Data

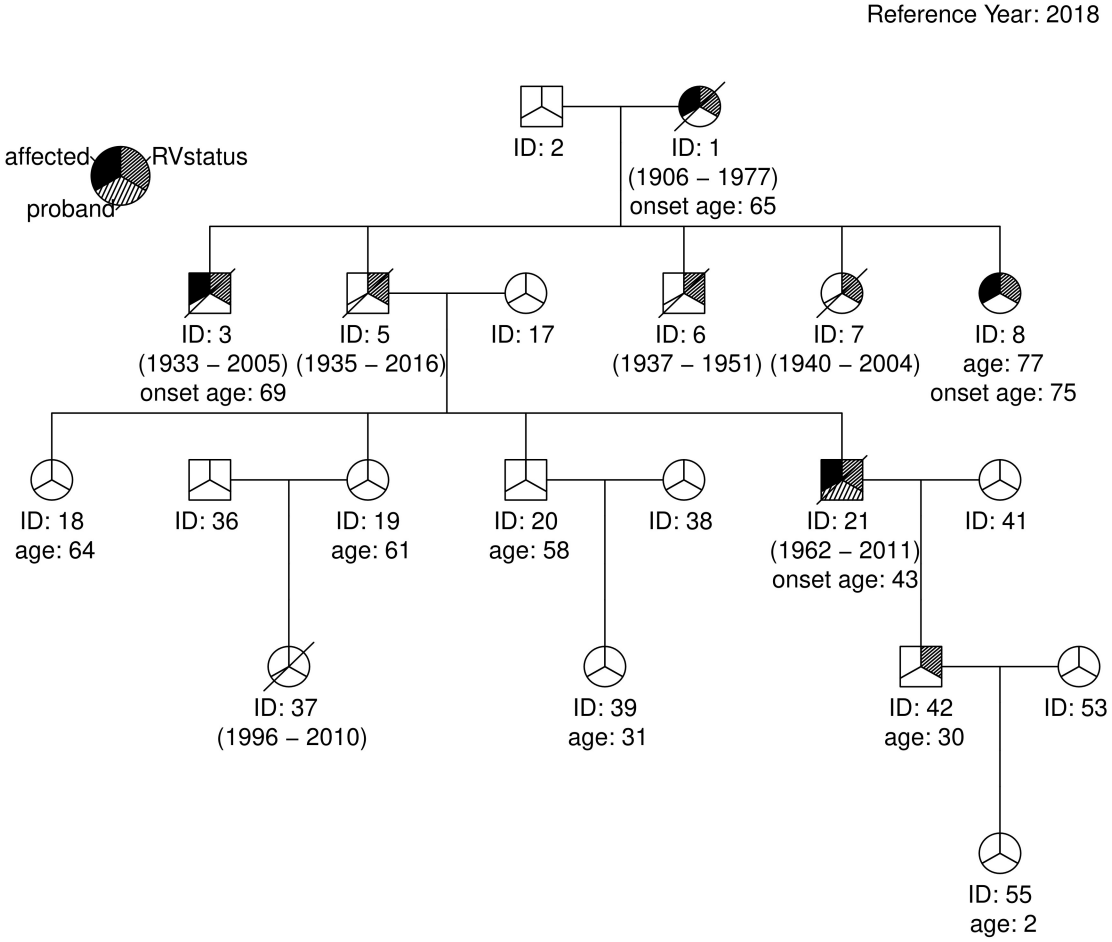
The R package `SimRVPedigree` [8] is used to simulate pedigrees ascertained for multiple disease-affected relatives. The following example demonstrates how users could use `SimRVPedigree` to simulate five pedigrees ascertained for three or more disease-affected relatives in parallel with the `doParallel`[7] and `doRNG`[2] packages.

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

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

# load libraries needed to simulate pedigrees in parallel.
library(doParallel)
```

```
library(doRNG)

npeds <- 5    #set the number of pedigrees to generate

cl <- makeCluster(2)   # create cluster
registerDoParallel(cl) # register cluster


# simulate a sample of five pedigrees using foreach
study_peds = foreach(i = seq(npeds), .combine = rbind,
                  .packages = c("SimRVPedigree"),
                  .options.RNG = 844090518
) %dorng% {
  # Simulate pedigrees ascertained for at least three disease-affected individuals,
  # according to the age-specific hazard rates in the `AgeSpecific_Hazards` data
  # set, ascertained from 1980 to 2010, with start year spanning
  # from 1900 to 1920, stop year set to 2018, and with genetic relative-risk 50.
  sim_RVped(hazard_rates = my_HR,
            GRR = 50, FamID = i,
            RVfounder = TRUE,
            founder_byears = c(1900, 1920),
            ascertain_span = c(1980, 2010),
            stop_year = 2018,
            recall_probs = c(1, 0.5, 0),
            num_affected = 3)[[2]]}

stopCluster(cl) #shut down cluster
```

For the purpose of demonstration, we will use the `study_peds` data set provided by `SimRVSequences`.

```
# import study peds
data("study_peds")

# Plot the pedigree with FamID 3
plot(study_peds[study_peds$FamID == 3, ],
     ref_year = 2018)
```



According to the legend, disease-affected individuals (IDs 1, 3, 8, and 21) are indicated by a solid shading in the upper-left third of their symbol. The proband (ID 21) is indicated by shading in the lower portion of his symbol. Individuals who have inherited the cRV (IDs 1, 3, 5, 6, 7, 8, 21 and 42) are indicated by shading in the upper-right portion of their symbol.

The seed founder (ID 1) and all of his descendants have age data relative to the reference year, which is displayed below their symbols. If a descendant has died by the reference year (IDs 1, 3, 5, 6, etc.) their year of birth and death are displayed in parentheses, and their symbol will have a diagonal line over it. Descendants who are still alive at the reference year have an age identifier. Descendants who have experienced disease-onset by the reference year will have a disease-onset age identifier.

```
# View the first 4 rows of study_peds
head(study_peds, n = 4)
```

```
##   FamID ID sex dadID momID affected DA1 DA2 birthYr onsetYr deathYr RR
## 1     1  1   1    NA    NA     TRUE   1   0    1902    1985    1987 50
## 2     1  2   0    NA    NA    FALSE   0   0      NA      NA      NA  1
## 3     1  3   0     2     1    FALSE   0   1    1922      NA    1942 50
```

```
## 4     1  4   1     2     1     TRUE   0   1    1925    1991    1991 50
##   available Gen proband
## 1      TRUE   1   FALSE
## 2     FALSE   1   FALSE
## 3      TRUE   2   FALSE
## 4      TRUE   2    TRUE
```

The columns displayed in the output above are described as follows:

1. `FamID`: family identification number
2. `ID`: individual identification number
3. `sex`: sex identification variable: `sex = 0` for males, and `sex = 1` females.
4. `dadID`: identification number of father
5. `momID`: identification number of mother
6. `affected`: disease-affection status: `affected = TRUE` if individual has developed disease, and `FALSE` otherwise.
7. `DA1`: paternally inherited allele at the cRV locus: `DA1 = 1` if the cRV is inherited, and `0` otherwise.
8. `DA2`: maternally inherited allele at the cRV locus: `DA2 = 1` if the cRV is inherited, and `0` otherwise.
9. `birthYr`: the individual's birth year.
10. `onsetYr`: the individual's year of disease onset, when applicable, and `NA` otherwise.
11. `deathYr`: the individual's year of death, when applicable, and `NA` otherwise.
12. `RR`: the individual's relative-risk of disease.
13. `available`: availability status: `available = TRUE` if individual is recalled by the proband, and `FALSE` if not recalled or a marry-in.
14. `Gen`: the individual's generation number relative to the eldest pedigree founder. That is, the seed founder will have `Gen = 1`, his or her offspring will have `Gen = 2`, etc.
15. `proband`: a proband identifier: `proband = TRUE` if the individual is the proband, and `FALSE` otherwise.

Not all of the variables above are required to simulate SNV data; the required variables are: `FamID`, `ID`, `sex`, `dadID`, `momID`, `affected`, `DA1`, and `DA2`. Please note, if `DA1` and `DA2` are not provided, the pedigree is assumed to be fully-sporadic, i.e. `DA1 = DA2 = 0` for all members.

To learn more about simulating pedigrees with `SimRVPedigree` please refer to `SimRVPedigree`'s documentation and vignette.

Simulate SNV data for Pedigrees

Simulation of sequence data for ascertained pedigrees is accomplished with the `sim_RVstudy` function. We will illustrate the usage of this function in section 5.1. The `sim_RVstudy` function returns an object of class `famStudy`. In section 5.2 we discuss objects class `famStudy` in detail. Finally, in section 5.3 we demonstrate the usage and output of the `summary` function for objects of class `famStudy`.

The `sim_RVstudy` function

Simulating SNV data for a sample of ascertained pedigrees is achieved with our `sim_RVstudy` function. This function has the following required and optional arguments:

Required Arguments:

1. `ped_files` A data frame of pedigrees. This data frame must contain the variables: `FamID`, `ID`, `sex`, `dadID`, `momID`, `affected`, `DA1`, and `DA2`. If `DA1` and `DA2` are not provided we assume the pedigrees are fully sporadic. See section 4 for additional details regarding these variables.
2. `SNV_data` An object of class `SNVdata`. See section 3.2.4 for more information on objects of class `SNVdata` or execute `help(SNVdata)` in the console.

Optional Arguments:

4. `affected_only` A logical argument. When `affected_only = TRUE`, we only simulate SNV data for the affected individuals and the family members that connect them along a line of descent. When `affected_only = FALSE`, SNV data is simulated for the entire pedigree. By default, `affected_only = TRUE`.
5. `remove_wild` A logical argument that determines if SNVs not carried by any member of the study should be removed from the data. By default, `remove_wild = TRUE`.
6. `pos_in_bp` A logical argument which indicates if the positions in `SNV_map` are listed in base pairs. By default, `pos_in_bp = TRUE`. If the positions in `SNV_map` are listed in centiMorgans set `pos_in_bp = FALSE` instead.
7. `gamma_params` The respective shape and rate parameters of the gamma distribution used to simulate the distance between chiasmata. By default, `gamma_params = c(2.63, 2*2.63)`, as in [11].
8. `burn_in` The "burn-in" distance in centiMorgans, as defined by [11], which is required before simulating the location of the first chiasmata with interference. By default, `burn_in = 1000`.

We will demonstrate the usage of the `sim_RVstudy` using the `EXhaps` and `EXmuts` datasets that were discussed in sections 3.1 and 3.3. Notice that the `sim_RVstudy` function expects and object of class `SNVdata`. We construct this object from the `EXmuts` and `EXhaps` dataset.

```
#construct an object of class SNVdata
ex_data <- SNVdata(Haplotypes = EXhaps,
                   Mutations = EXmuts)
```

Assuming readers have simulated pedigrees as in section 4, we may execute the `sim_RVstudy` function as follows.

```
set.seed(11956)
# simulate SNV data using sim_RVstudy
# study_seq <- sim_RVstudy(ped_files = study_peds,
#                          SNV_map = EXmuts,
#                          haplos = EXhaps)

study_seq <- sim_RVstudy(ped_files = study_peds,
                         SNV_data = ex_data)


# determine the class of study seq
class(study_seq)
```

```
## [1] "famStudy" "list"
```

From the output above, we see that the `sim_RVstudy` function returns an object of class `famStudy`, which inherits from the `list` class. More specifically, a `famStudy` object is a list that contains the following four items:

- a data frame named `ped_files`,
- a sparse matrix named `ped_haplos`,
- a data frame named `haplo_map`,
- and a data frame named `SNV_map`.

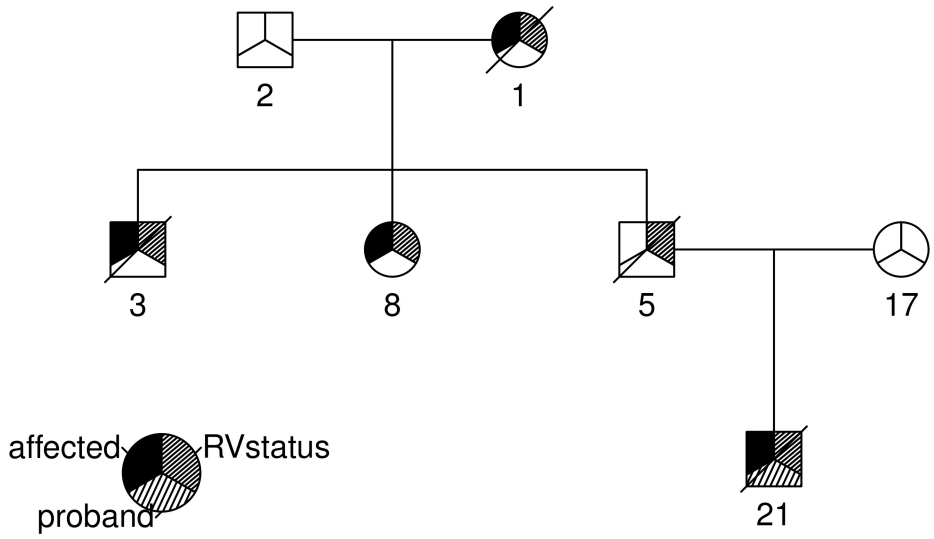
We will discuss each of the four items contained in the `famStudy` object in the following section.

Objects of class famStudy

The ped_files data frame

The data frame ped_files contained in the famStudy object is a potentially reduced set of pedigree files for the original families. Recall that, by default affected_only = TRUE, so the pedigrees supplied to sim_RVstudy are reduced to contain only the affected individuals and the individuals who connect them along a line of descent.

```
# plot the pedigree returned by sim_RVseq for family 3.
# Since we used the affected_only option the pedigree has
# been reduced to contain only disease-affected relatives.
plot(study_seq$ped_files[study_seq$ped_files$FamID == 3, ],
     location = "bottomleft")
```



Comparing the plot above to the plot of the same family in section 4, we can see that the number of relatives for whom we simulate SNV data is far less than the total number of relatives in the ascertained pedigree.

The ped_haplos matrix

Next, we look at the sparse matrix, ped_haplos, which is contained in the famStudy object. This sparse matrix contains simulated haplotype data.

```
# View the first 15 haplotypes in ped_haplos
study_seq$ped_haplos[1:15, ]
```

```
## 15 x 25 sparse Matrix of class "dgCMatrix"
```

```
##
##  [1,] . . . 1 . . . . . . . . . . . . . . . . . . . . . . . .
##  [2,] . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [3,] . . . . . . . . . . . . . . . . . . . . 1 . . . . . . .
##  [4,] 1 . . . 1 . . . . . . . . . . . . . . . . . . . . . . .
##  [5,] . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##  [6,] . . . 1 . . . . . . . . . . . . . . . . . . . . . . . .
##  [7,] 1 . . . 1 . . . . . . . . . . . . . 1 . . . . . . . .
##  [8,] . . . 1 . . . . . . . . . . . . . . . . . . . . . . . .
##  [9,] . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## [10,] . . . 1 . . . . . . . . . . . . . . . . . . . . . . . .
## [11,] . 1 . . . . . . . . . . . . . . . . . . . . . . 1 . . .
## [12,] . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## [13,] . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## [14,] . . . . . . . . . . . . . . . . . . . . . 1 . . . . . .
## [15,] . . . . . . . . . . . . . . . . . . . . . 1 . . . . . .
```

The `ped_haplos` matrix contains the haplotype data for each individual in the `ped_files` output.

```
# Determine the dimensions of ped_haplos
dim(study_seq$ped_haplos)
```

```
## [1] 62 25
```

```
# Determine the dimensions of EXhaps
dim(EXhaps)
```

```
## [1] 20000   500
```

Notice that `ped_haplos` only has 24 columns (i.e. mutations) even though `EXhaps`, which was supplied to `sim_RVstudy`, has a total of 500 columns. This occurs when the `remove_wild` setting is used to remove superfluous SNVs from our data. That is, when no one in the study carries a mutated allele at an SNV locus, that marker is removed from the data. Since we are focused on rare variation, and since `EXhaps` and `study_peds` are small toy data sets the reduction in data is quite significant.

The `SNV_map` data frame

In this subsection we discuss the `SNV_map` data frame contained in the `famStudy` object. Recall that one of `sim_RVstudy`'s arguments is also named `SNV_map`. When the `remove_wild` argument of `sim_RVstudy` is set to `FALSE`, the data frame supplied to `SNV_map` will be returned unchanged. However, when `remove_wild` is set to `TRUE`, the `SNV_map` contained in the `famStudy` object will be a reduced data frame that contains only SNVs that are carried by at least one study participant.

```
# View the first 4 observations in SNV_map
head(study_seq$SNV_map, n = 4)
```

```
##   colID chrom  position   afreq      marker pathwaySNV is_CRV
## 1     1     1  45043357 0.00310  1_45043357      FALSE  FALSE
## 2     2     1 236599491 0.00895 1_236599491      FALSE  FALSE
## 3     3     2   3960458 0.00535   2_3960458      FALSE  FALSE
## 4     4     2 201284953 0.00010 2_201284953       TRUE   TRUE
```

The output `SNV_map` catalogs the SNVs, i.e. columns, in `ped_haplos`. For example, the first column in `ped_haplos` is a SNV located on chromosome 1 at position 45043357.

The `haplo_map` data frame

Next, we look at the `haplo_map` data frame contained in the `famStudy` object. The `haplo_map` data frame is used to map the haplotypes (i.e. rows) in `ped_haplos` to the individuals in `ped_files`.

```
# view the first observations in haplo_map
head(study_seq$haplo_map)
```

```
##   FamID ID affected      FamCRV
## 1     1  1     TRUE 2_201284953
## 2     1  1     TRUE 2_201284953
## 3     1  2    FALSE 2_201284953
## 4     1  2    FALSE 2_201284953
## 5     1  4     TRUE 2_201284953
## 6     1  4     TRUE 2_201284953
```

Notice that in the output above individual 1 from family 1 is listed in rows 1 and 2, and individual 2 from family 1 is listed in rows 3 and 4. This is because the genetic material inherited from each parent is stored in its own row. **By convention, the first row represents the paternally inherited haplotype, while the second row represents the maternally inherited haplotype**. Additionally, note the variable `FamCRV` contained in `haplo_map`, which identifies each family's causal rare variant by marker name.

The following code demonstrates how to reduce `haplo_map` to quickly identify the cRV for each family.

```
# to quickly view the causal rare variant for
# each family we supply the appropriate columns of
# haplo_map to unique
unique(study_seq$haplo_map[, c("FamID", "FamCRV")])
```

```
##    FamID      FamCRV
## 1      1 2_201284953
## 11     2      no_CRV
## 19     3  8_23081797
## 33     4 2_201284953
## 41     5 10_89008915
```

From the output above, we see that families 1 and 4 segregate variant `2_201284953`, family 2 does not segregate any SNV in the pool of causal rare variants, family 3 segregates variant `10_89016102`, and family 5 segregates variant `10_89015798`. Note that family 2 does not segregate any of the SNVs in our pool of causal rare variants because it is a fully sporadic pedigree; i.e. there is no genetic predisposition to disease in this pedigree.

The `summary.famStudy` function

To obtain summary information for the simulated sequence data we supply objects of class `famStudy` to the `summary` function. Recall that the output of the `sim_RVstudy` function is an object of class `famStudy`; hence, users may supply the output of `sim_RVstudy` directly to `summary`.

```
# supply the famStudy object, returned by
# sim_RVstudy, to the summary function
study_summary <- summary(study_seq)

# determine the class of study_summary
class(study_summary)
```

```
## [1] "list"
```

```
# view the names of the items in study_summary
names(study_summary)
```

```
## [1] "fam_allele_count" "pathway_count"
```

From the output above, we see that when supplied an object of class `famStudy` the `summary` function returns a list containing two items: `fam_allele_count` and `pathway_count`. The item `fam_allele_count` is a matrix that includes counts of the SNVs shared among the disease-affected relatives in each pedigree.

```
# view fam_allele_count
study_summary$fam_allele_count
```

```
##      FamID 1_45043357 1_236599491 2_3960458 2_201284953 3_47610321
## [1,]     1          1           0         0           4          1
## [2,]     2          0           3         0           0          0
## [3,]     3          0           0         3           0          0
## [4,]     4          0           0         0           3          0
## [5,]     5          0           0         0           0          0
##      3_49154162 3_129583578 4_2249748 4_168989553 5_77691934 5_115614755
## [1,]          0           0         0           0          0           0
## [2,]          0           0         0           0          0           0
## [3,]          1           1         0           1          0           0
## [4,]          0           0         0           0          0           1
## [5,]          0           0         1           0          1           0
##      7_101314288 8_23081797 8_144514988 10_89008915 12_8946339 12_26936616
## [1,]           0          0           0           0          1           0
## [2,]           0          0           0           0          0           0
## [3,]           2          4           2           0          0           2
## [4,]           0          0           0           0          0           0
## [5,]           0          0           0           4          0           0
##      15_63129900 17_58007302 21_33437240 21_33449698 22_50460703
## [1,]           0           0           0           0           0
## [2,]           1           2           0           0           0
## [3,]           0           0           0           3           2
## [4,]           0           0           2           0           0
## [5,]           0           0           0           0           0
```

Looking at the output above, we see that only one affected relative in family 1 carries a copy of the first SNV, `1_45043357`. It is noteworthy that the disease-affected relative in families 1 and 4 carry so many copies of SNV `2_201284953`; not surprisingly, this is the causal rare variant for these two families.

The item `pathway_count` is a data frame that catalogs the total number of SNVs observed in disease-affected study participants.

```
# view pathway_count
study_summary$pathway_count
```

```
##    chrom  position      marker total is_CRV pathwaySNV
## 1      1  45043357  1_45043357     1  FALSE      FALSE
## 2      1 236599491 1_236599491     3  FALSE      FALSE
## 3      2   3960458   2_3960458     3  FALSE      FALSE
## 4      2 201284953 2_201284953     7   TRUE       TRUE
## 5      3  47610321  3_47610321     1  FALSE      FALSE
## 6      3  49154162  3_49154162     1  FALSE      FALSE
## 7      3 129583578 3_129583578     1  FALSE      FALSE
## 8      4   2249748   4_2249748     1  FALSE      FALSE
## 9      4 168989553 4_168989553     1  FALSE      FALSE
```

```
## 10     5  77691934  5_77691934     1  FALSE      FALSE
## 11     5 115614755 5_115614755     1  FALSE      FALSE
## 12     7 101314288 7_101314288     2  FALSE      FALSE
## 14     8  23081797  8_23081797     4   TRUE       TRUE
## 15     8 144514988 8_144514988     2  FALSE      FALSE
## 16    10  89008915 10_89008915     4   TRUE       TRUE
## 17    12   8946339  12_8946339     1  FALSE      FALSE
## 18    12  26936616 12_26936616     2  FALSE      FALSE
## 19    15  63129900 15_63129900     1  FALSE      FALSE
## 22    17  58007302 17_58007302     2  FALSE      FALSE
## 23    21  33437240 21_33437240     2  FALSE       TRUE
## 24    21  33449698 21_33449698     3  FALSE       TRUE
## 25    22  50460703 22_50460703     2  FALSE      FALSE
```

From `pathway_count`, we see that there are several SNVs carried by more than 1 disease-affected study participants; however, only 5 of the SNVs carried by more than one individual reside in the pathway of interest.

`SimRVSequences` Model

In this section we describe the model assumptions employed by `SimRVSequences` to simulate single-nucleotide variant (SNV) data for pedigrees.

1. Given a sample of ascertained pedigrees we allow families to segregate different rare variants, but assume that within a family genetic cases are due to a single, **causal rare variant (cRV)** that increases disease susceptibility. We assume that only one copy of the familial cRV is introduced to the pedigree and that the cRV status of every pedigree member is known. The `R` package `SimRVPedigree` [8] may be used to simulate pedigrees that meet these criteria (see section 4).

2. Recall that users are expected to specify a pool of cRVs which may represent cRVs located in a pathway, or different cRVs in the same gene. The size of the pool is entirely up to the user: it can contain many SNVs or only a single SNV. However, we urge those who are using `SimRVPedigree` in conjunction with `SimRVSequences` to choose cRVs so that their cumulative carrier probability is less than or equal to 0.002 [9]. From this pool we sample a single cRV for each pedigree with replacement.

3. Founder haplotypes are sampled from a user-specified, population distribution of haplotypes, conditional on their cRV status.
   - For the founder who introduces the cRV we sample one haplotype from the haplotypes that carry the familial cRV and the other from the haplotypes that do not carry ANY of the cRVs in our pool. This ensures that this founder introduces only a single copy of the familial cRV.
   - For the pedigree founders who do NOT introduce a cRV we sample their haplotypes from the haplotypes that do not carry ANY of the cRVs in our pool. Hence, fully sporadic families will not segregate any cRVs, nor will any non-carrier individual who marries into a genetic family.

4. We use a conditional gene-drop algorithm to model inheritance from parent to offspring, which can be described as follows.

- First, we simulate the formation of a parent's gametes.
  1. Following [11], we model genetic recombination with chiasmata interference. To accomplish this the distance between chiasmata is modelled by a gamma distribution. Additionally, we use a "burn-in" process before the start of the chromosome to simulate the location of the first chiasmata. The default parameter settings for the gamma distribution are `shape = 2.63` and `rate = 2*2.63`, and `burn_in = 1000`. To model recombination without chiasmata interference, i.e. according to Haldane's model, set `shape = 1` and `rate = 2` and `burn_in = 0`. **Please note, presently**

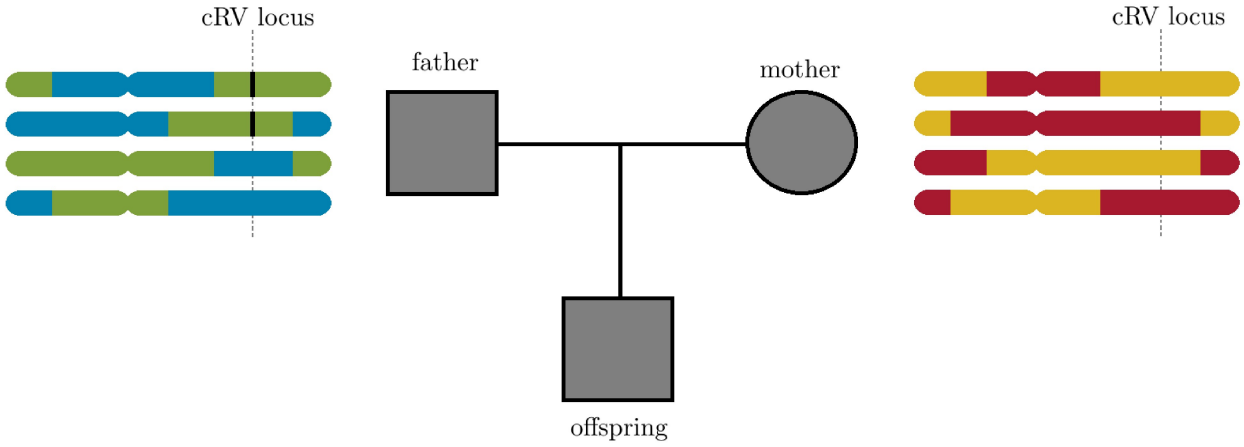


Figure 2: **Figure 2.** A family trio with possible maternal and paternal recombinant chromatids for a diploid organism with a single set of homologous chromosomes. The cRV locus is indicated by a dashed vertical line over the chromatids, and the cRV is indicated by a black vertical bar within the chromatids. The top two chromatids of the father contain the only copies of the cRV.

`SimRVSequences` only permits simulation of recombination in autosomes, i.e. non-sex chromosomes.

2. We assume no chromatid interference so that non-sister chromatids are equally likely to participate in a crossover event.
3. To simulate the formation of gametes we assume that homologous chromatids are assigned to one of four gamete cells with equal probability. This assignment occurs independently for non-homologous chromosomes.

- We use a conditional gene drop algorithm to determine which of the four gametes is transmitted from parent to offspring by considering the cRV status of the parent and the offspring, as follows.
  - Case 1: If **both the parent and offspring carry the cRV** we sample the inherited gamete from the two possible parental gametes that carry the cRV. For example, in the father to offspring transmission in Figure 2, we choose from the paternal gametes represented by the top two chromatids.
  - Case 2: If the **parent carries the cRV but the offspring does not**, we sample the inherited gamete from the two possible parental gametes that do not carry the cRV. For example, in the father to offspring transmission in Figure 2, we choose from the paternal gametes represented by the bottom two chromatids.
  - Case 3: If the **parent is not a carrier of the cRV**, then the cRV status of the offspring is irrelevant for this parent's transmission because the offspring cannot inherit it from this parent. In this scenario, we sample the inherited gamete from the four parental gametes with equal probability. For example, this would be the case in the mother to offspring transmission in figure 2, since she is not a carrier of the cRV.

We use these assumptions in our algorithm to simulate sequence data for pedigrees (see introduction.)

Method Overview

The methods provided by `SimRVSequences` are described as follows.

- Methods to assist with creating the recombination map for SLiM:
  1. `combine_exons`: This function will be useful to users who would like to combine overlapping exon observations into a single observation. Upon supplying a data frame which contains overlapping

exon segments, this function will return a data frame with combined segments (see appendix).

2. `create_slimMap`: This function is used both before and after simulating SLiM data. Prior to simulating the SNV data with SLiM, this function can be used to create the recombination map required by SLiM to simulate exon-only SNV data (see section 3.1.1). Additionally, if this function was used to create the recombination map users may then supply the output of this function to `read_slim` to remap mutations to their correct locations when importing SLiM data to `R` (see section 3.1.2).

- Method to assist with importing data to `R`.
  3. `read_slim`: This function is used to import to `R` data stored to a .txt file by SLiM's outputfull() method (see section 3.1.2).
  4. `imposrt_SNVdata`: This function is used to import pre-formatted exon-only SNV data from the 1000 Genomes Project (see section 3.3).
- Methods to assist with formatting SNV data:
  5. `genos2sparseMatrix`: This function is used to store genotypes as a sparse matrix (see section 3.2.2).
  6. `SNVdata`: This is the constuctor function for objects of class `SNVdata` (see section 2.3.4).
- Methods to assist with simulation and basic manipulation of SNV data for pedigrees
  7. `sim_RVstudy`: This function is used to simulate the SNV data for a sample of pedigrees (see section 5.1).
  8. `summary.famStudy`: This function is used to obtain a summary of the SNVs shared by the disease-affected relatives in each pedigree (see section 5.3).

Timing

All of the following timing estimates were performed on a Windows OS with an i7-4790 @ 3.60GHz and 12 GB of RAM.

Importing genome-wide, exon-only SNV data for 10,000 diploid individuals from the .txt file produced by SLiM's outputfull() method, using the `read_slim` function required approximately 4 minutes. The imported data originally contained a total of 306,825 SNVs over 20,000 haplotypes.

For timing of `sim_RVstudy`, we explore the effects of the `affected_only` and `remove_wild` arguments. When `affected_only = TRUE`, sequence data is only simulated for disease-affected relatives and the family members that connect them along a line of descent, otherwise if `affected_only = FALSE` sequence data is simulated for the entire pedigree. When `remove_wild = TRUE` the size of the data is reduced by removing from the data SNVs not carried by any study participant; otherwise if `remove_wild = FALSE` no data reduction occurs.

The following timings were performed on a sample of 100 pedigrees, each of which contained at least 3 relatives affected by a lymphoid cancer. To simulate these pedigrees we used the same settings as in the parallel processing example of section 4. For timing, we used genome-wide, exon-only SNV data for 10,000 diploid individuals simulated by SLiM. After reducing the SLiM data to contain SNVs with alternate allele frequency less than or equal to 0.01, there were a total of 178,997 SNVs in our data. We chose a cRV pool of size 20 at random from the SNVs with a alternate allele frequency of 5e-05 located in the sub-apoptosis pathway described by the `hg_apopPath` data set (see section 3.2 and 3.3).

Number of Pedigrees	`affected_only`	`remove_wild`	time (in minutes)
100	TRUE	TRUE	0.976
100	TRUE	FALSE	1.267
100	FALSE	TRUE	2.032
100	FALSE	FALSE	3.051

For the sample of 100 pedigrees used to time `sim_RVstudy`, setting `affected_only = FALSE` resulted in simulation of sequence data for a total of 1473 individuals, when `affected_only = TRUE` the data were reduced to contain a total of 487 individuals. On average, this resulted in a reduction of 9.86 individuals from each pedigree.

References

[1] Douglas Bates and Martin Maechler (2018). **Matrix: Sparse and Dense Matrix Classes and Methods**. *R package version 1.2-14.* https://CRAN.R-project.org/package=Matrix

[2] Renaud Gaujoux (2017). **doRNG: Generic Reproducible Parallel Backend for 'foreach' Loops.** *R package version 1.6.6* https://CRAN.R-project.org/package=doRNG.

[3] Benjamin C. Haller and Philipp W. Messer (2017). *Slim 2: Flexible, interactive forward genetic simulations.* Molecular Biology and Evolution; 34(1), pp. 230-240.

[4] Kelley Harris and Rasmus Nielsen (2016). *The genetic cost of neanderthal introgression.* Genetics, 203(2): pp. 881-891.

[5] Donna Karolchik, Angela S. Hinrichs, Terrence S. Furey, Krishna M. Roskin, Charles W. Sugnet, David Haussler, and W. James Kent (2004). *The UCSC Table Browser data retrieval tool.* Nucleic Acids Res, 32: D493-6.

[6] W. James Kent, Charles W. Sugnet, Terrence S. Furey, Krishna M. Roskin, Tom H. Pringle, Alan M. Zahler, and David Haussler (2002). *The human genome browser at UCSC.* Genome Res, 12(6):996-1006.

[7] Microsoft Corporation and Steve Weston (2017). **doParallel: Foreach Parallel Adaptor for the 'parallel' Package.** *R package version 1.0.11.* https://CRAN.R-project.org/package=doParallel.

[8] Christina Nieuwoudt and Jinko Graham (2018). **SimRVPedigree: Simulate Pedigrees Ascertained for a Rare Disease.** *R package version 0.1.0.* https://CRAN.R-project.org/package=SimRVPedigree.

[9] Christina Nieuwoudt, Samantha J Jones, Angela Brooks-Wilson, and Jinko Graham (2018). *Simulating pedigrees ascertained for multiple disease-affected relatives.* Source Code for Biology and Medicine, 13:2.

[10] Hoifung Poon, Chris Quirk, Charlie DeZiel, David Heckerman (2018). *Literome: PubMed-scale genomic knowledge base in the cloud.* Bioinformatics, 30:2840-2842.

[11] Roeland E. Voorrips, Chris A Maliepaard. (2012). *The simulation of meiosis in diploid and tetraploid organisms using various genetic models.* BMC Bioinformatics, 13:248.

[12] Ellen M. Wijsman (2012). *The role of large pedigrees in an era of high-throughput sequencing.* Human Genetics 131, pp. 1555-1563.

Appendix

The `combine_exons` function is used to combine overlapping exons into single segments.

This function has only one argument called `exon_data`, which is a data frame that includes the following variables

1. `chrom` a chromosome identifier,
2. `exonStart` the first position of the exon in base pairs, and
3. `exonEnd` the last position of the exon in base pairs.

Usage of the `combine_exons` function is illustrated below.

```
# create an example data frame that contains the
# the variables: chrom, exonStart, and exonEnd
exDat <- data.frame(chrom     = c(1, 1, 1, 2, 2, 2),
                    exonStart = c(1, 2, 5, 1, 3, 3),
                    exonEnd   = c(3, 4, 7, 4, 5, 6))

# View exDat data set
exDat
```

```
##   chrom exonStart exonEnd
## 1     1         1       3
## 2     1         2       4
## 3     1         5       7
## 4     2         1       4
## 5     2         3       5
## 6     2         3       6
```

From the output above, we see that the first two exons in chromosome one overlap (i.e. the exons with ranges [1, 3] and [2, 4]), as do all three exons in chromosome two.

```
# supply exDat to combine_exons
# and view the results
combine_exons(exDat)
```

```
##   chrom exonStart exonEnd
## 1     1         1       7
## 2     2         1       6
```

After supplying the data to `combine_exons` we see that the segments [1, 3] and [2, 4] on chromosome one have been combined in to a single segment: [1, 4]. Similarly, the three overlapping exons from chromosome two: [1, 4], [3, 5], and [3, 6], have now been combined into the segment [1, 6].