Working with RVPedigree

Lennart C. Karssen, Karim Oualkacha & Celia M.T. Greenwood

December 3, 2015

Contents

1 Introduction

RVPedigree is an R package that contains methods to perform autosomal rare variant association analyses with a quantitative phenotype, specifically for the situation when there are related individuals or families in the data. There are 5 variants of such a test included in the package, and in particular RVPedigree includes methods that test for association with non-normally distributed phenotypes; improved power is possible by appropriate modelling of the distributions.

The main user-visible functions of the package are listed below. The `RVPedigree()` function is the main function that can be used to analyse multiple genomic regions in one go. The `*.region()` functions can be used to run a given method on a single genomic region specified by its start and end base pair position.

- `ASKAT.region()`: ASKAT [?] performs a test of association when the phenotype (specifically, the residuals after adjusting the phenotype for fixed covariates) follows a normal distribution (see § 3.1).
- `NormalizedASKAT.region()`: For ASKAT-normalized, ASKAT is implemented for th rank-normalized residuals of the phenotype, after adjusting for covariates (see § 3.2).
- `VCC1.region()`, `VCC2.region()` and `VCC3.region()` are copula-based tests of association for non-normal traits, using three different p-value approximations. In Ref. [1], these three tests are known as VC-C1, VC-C2 and VC-C3.
  - *VC-C1* estimates the p-value with an asymptotic approximation, which has a small liberal bias for small p-values (see § 3.3),
  - *VC-C2* uses permutation of founder haplotypes to estimate the p-values (see § 3.4),
  - *VC-C3* generates the empirical distribution of the test statistics under the null, and uses this to estimate the p-values (see § 3.5).
- `RVPedigree()`: main function that can be used to run a one of the methods above on one or more genomic regions (see § 4).

If the phenotype is not normal, *VC-C1* is the computationally fastest method, and these p-values can be used to triage which regions of the genome should be re-analyzed with *VC-C2* or *VC-C3* to obtain more accurate significance estimates. Both of these two methods have accurate type 1 errors, but the last one (*VC-C3*) is much faster.

In order to run any of the examples below, the package needs to be loaded first, of course:

```
library(RVPedigree)

## Loading required package:  foreach
## Loading required package:  doParallel
## Loading required package:  iterators
## Loading required package:  parallel
```

2 Loading the input data

Before running an association test with one (or more) of the methods, the following data needs to be present[1]:

- *Phenotype data*; phenotype data should be present in the form of an R vector (one value for each individual). It is up to you (as user) to create the vector, for example by reading it from a CSV file using R's `read.csv()` function or like this:

```
  pheno.file <- system.file("extdata", "pheno.dat",
                            package="RVPedigree")
  pheno.table <- read.table(pheno.file, header=FALSE)
  dim(pheno.table)

## [1] 600   1

  pheno <- pheno.table[, 1]
  head(pheno)

## [1]  1.2506154  2.9575878  5.2475735 -0.6855776  1.3271224  2.7389056
```

- *Covariate data*; this data should be present in the form of a matrix. Like the phenotype data it us up to you to load this data. For example:

```
  covar.file <- system.file("extdata", "covariates.dat",
                            package="RVPedigree")
  covar <- as.matrix(read.table(covar.file, header=FALSE))
  dim(covar)

## [1] 600   3
```

- *Pedigree data*; data on familial relationships needs to be available in *Linkage* format. This format is also used by Plink (which is used to read the genotype data, see below) and consists of four columns (separated by spaces) for each individual in the pedigree:
  - Family or pedigree ID
  - Individual ID
  - Father ID
  - Mother ID

[1] The example code in this vignette uses files that are included in the RVPedigree package, that is why the `package` argument to the `system.file()` function is used.

See e.g. the Plink website for more details.

- *Genotype data*; genotype data can be loaded from Plink files, see §2.1.
- *Relationship matrix*; each of the methods requires a (symmetric) relationship matrix ($2 \times$ the kinship matrix). The function `GetRelMatrix` can be used to create such a matrix if you do not have one. See §2.2 for details.

2.1 Genotype data

For genotype data RVPedigree accepts Plink input files. These can be in `.ped` or `.bed` format. The main RVPedigree functions have two keywords for loading genotype data: `type`, which specifies the type of file (values are `ped` or `bed`) and `filename`, which is the full filename (including path if necessary) to the main plink file (`file.ped` or `file.bed`) containing the genotype data (as well as the pedigree data).

Some other formats for handling genotype data can interface well with Plink. For example, for users of GenABEL, data can be output from GenABEL format to Plink `.ped` or `.bed` format using the function `export.plink()` from GenABEL [2].

It is important to note that for the *VC-C2* and *VC-C3* methods, the genotype data in a region are assumed to be phased, so that the first alleles at all variants are assumed to lie on the same chromosome, and likewise for the second alleles. More on this topic can be found in sections **??**.

RVPedigree is designed for autosomal genotypes and should not be used for X or Y chromosome analyses.

2.2 Loading or creating a relationship matrix

The relationship matrix (twice the kinship matrix) can be calculated in two ways: by using pedigree data or by using genomic data. The option `datatype` of the `GetRelMatrix()` function is used to tell the `GetRelMatrix()` function which method to use. This can be either `datatype="pedigree"` or `datatype="genomic"`. Obviously if you don't have (full) pedigree data only the `"genomic"` option makes sense.

Depending on the file format in which you have stored your pedigree/genotypes, you need to tell `GetRelMatrix` how to read the data. If your data is in Plink `.ped` or `.bed` format, you need to have the `snpStats` package installed[3].

For now we assume you have your kinship matrix already stored in an Rdata file, then loading it and converting it to a relationship matrix Φ is easy:

```
kinship.file <- system.file("extdata", "kinmat.Rdata",
                            package="RVPedigree")
load(kinship.file)
```

[2]See http://www.genabel.org/GenABEL/export.plink.html

[3]See http://www.bioconductor.org/packages/release/bioc/html/snpStats.html.

```
dim(kin1)

## [1] 600 600

rel.mat <- 2 * kin1
```

2.2.1 A relationship matrix from data in Plink `ped` format

For data in Plink's `.ped` format, the following command creates a relationship matrix based on the pedigree information in the first few columns of a file called `mypedfile.ped`:

```
rel.mat.pedigree.ped <- GetRelMatrix(datatype="pedigree",
                                     plinkbasefile="mypedfile")
```

To compute the genomic relationship matrix from the same file the RVPedigree package uses Plink v1.90[4]. In order to tell RVPedigree where to find the Plink 1.90 binary, you need to specify the path to it using the `path2Plink` option[5]. For example:

```
rel.mat.genomic.ped <- GetRelMatrix(datatype="genomic",
                                    path2Plink="/tools/genepi/Plink_1.90/plink",
                                    plinkbasefile="mypedfile")
```

Here we assume that the Plink 1.90 binary can be found in the directory `/tools/genepi/Plink_1.90/`.

2.2.2 A relationship matrix from data in Plink `bed` format

For data in Plink's `.bed` format, the following command creates a relationship matrix based on the pedigree information in the first few columns of a file called `mybedfile.fam`:

```
rel.mat.pedigree.bed <- GetRelMatrix(datatype="pedigree",
                                     is.binary=TRUE,
                                     plinkbasefile="mybedfile")
```

To compute the genomic relationship matrix from the data in the `.bed` file, use:

[4]Plink 1.90 is currently in beta stage, but it is expected that it will become a stable Plink 2 in the near future, see http://pngu.mgh.harvard.edu/~purcell/plink/plink2.shtml. The latest binaries can be downloaded from https://www.cog-genomics.org/plink2.

[5]The reason why you need to explicitly enter the (full) path is the following: The name of the Plink 1.90 binary is **plink**, just like the 'regular' Plink 1.07. Given that Plink 1.90 is still beta we assume that on most systems running **plink** from the Linux command line means you get v1.07. You can check this by running **plink** and checking the output if that command.

```r
rel.mat.genomic.bed <- GetRelMatrix(datatype="genomic",
                                    path2Plink="plink_1.90",
                                    is.binary=TRUE,
                                    plinkbasefile="mybedfile")
```

In this example we assumed that the binary for Plink 1.90 is called `plink_1.90` and that it can be found somewhere in your `PATH`[6].

2.2.3 A relationship matrix from data in GenABEL `gwaa-data` format

If you are a user of the GenABEL R package[7] you probably have your data stored as a GenABEL object (i.e. GenABEL's own `gwaa-data` format). The genotype data in a GenABEL object can be used like this to create a relationship matrix. The following code loads the data and prints the number of SNPs and the number of samples in the data set:

```r
genabel.data <- system.file("extdata", "gwaa.data.RData",
                            package="RVPedigree")
library(GenABEL)

## Loading required package:  MASS
## Loading required package:  GenABEL.data

load(genabel.data)
nsnps(data1)

## [1] 3573

nids(data1)

## [1] 128
```

Note that this is a different data set than the one used before, and it will be used in §4.

To compute the genomic relationship matrix for this data set, use the `GetRelMatrix()` function from RVPedigree:

```r
rel.mat.large <- GetRelMatrix("genomic", gwaa.data=data1)
```

[6]On a Linux system the environment variable `$PATH` contains a list of directories in which the excutables of tools can be found. You can type `echo $PATH` on the Linux command line to show the directories in your `PATH`.

[7]See `www.genabel.org` and `https://cran.r-project.org/web/packages/GenABEL/`.

3 Analysing a single genomic region

With the phenotype data, the covariates and the relationship matrix loaded it is time for tests of association. Since each of the methods in this package is a region-based method we will first do one more general step: reading the map data of the genotypes so we know the basepair coordinate of each genomic variant.

```
map.file <- system.file("extdata", "test.map",
                        package="RVPedigree")
mymap <- readMapFile(map.file)
```

Now is also a good time to create a variable that contains the name of the file with genotype data:

```
geno.file <- system.file("extdata", "test.ped",
                         package="RVPedigree")
```

Note: If you would like to run association tests on more than one region (or use different methods on the same region), the analyses can be speed up by pre-computing the eigenvalues and eigenvectors of the relationship matrix and pass these on as parameters. This will be shown in the ASKAT-Normalized example in §3.2.

3.1 Using ASKAT

This is the simplest way to run the ASKAT method on a single genomic region:

```
askat.result <- ASKAT.region(y=pheno, X=covar, Phi=rel.mat,
                             type="ped",
                             filename=geno.file,
                             chr=20,
                             startpos=0,
                             endpos=10000000,
                             map=mymap,
                             regionname="Gene 1")

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic

print(askat.result)

##        Score.Test   P.value N.Markers
## Gene 1   454.7332 0.7794409        10
```

Here the `type` option indicates the type of files used for the genotype data: either Plink `.bed` files (the default, `type="bed"`) or Plink `.ped` files (`type="ped"`). The option `filename` points to the name of the file containing the genotype data and `chr` specifies on which chromosome the genomic region under study is located. The `startpos` and `endpos` options (set to 0 by default) are used to indicate the start and stop basepair positions of the region. The `regionname` option allows the user to specify a name for the region on which the association test is run. The name is then used in the output of the `ASKAT.region()` function, which can be handy in case the output of multiple `ASKAT.region()` runs need to be combined.

3.2 Using ASKAT-Normalized

As mentioned earlier it makes sense to pre-compute the eigenvalues and eigenvectors of the relationship matrix when you do more than one analysis:

```
relmat.eig    <- eigen(rel.mat)
relmat.eigvec <- relmat.eig$vectors
relmat.eigval <- relmat.eig$values
```

These pre-computed variables can then be passed on to the various 'region' functions via the `U`, and `S` parameters:

```
nASKAT.result <- NormalizedASKAT.region(y=pheno, X=covar, Phi=rel.mat,
                                        type="ped",
                                        filename=geno.file,
                                        chr=20,
                                        startpos=0,
                                        endpos=10000000,
                                        map=mymap,
                                        regionname="Gene 1",
                                        U=relmat.eigvec,
                                        S=relmat.eigval)

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic

print(nASKAT.result)

##        Score.Test   P.value N.Markers
## Gene 1   4998.208 0.5201545        10
```

3.3 Using VC-C1

This implements the VC-C1 test where an asymptotic approximation to the p-value is obtained. Phased founder haplotypes are not required.

```
vcc1.result <- VCC1.region(y=pheno, X=covar, Phi=rel.mat,
                           type="ped",
                           filename=geno.file,
                           chr=20,
                           startpos=0,
                           endpos=10000000,
                           map=mymap,
                           regionname="Gene 1")

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic

print(vcc1.result)

##         Score.Test   P.value N.Markers
## Gene 1   3345.974 0.2364904        10
```

3.4 Using VC-C2

The *VC-C2* method permutes haplotypes of the founders in order to obtain empirical p-values. Thus, phased genotypes for all founders are required. The following example assumes that the data in `geno.file` are already phased with the convention that the "first alleles at all variants are assumed to lie on the same chromosome, and likewise for the second alleles".

```
vcc2.result <- VCC2.region(y=pheno, X=covar, Phi=rel.mat,
                           type="ped",
                           filename=geno.file,
                           chr=20,
                           startpos=0,
                           endpos=10000000,
                           map=mymap,
                           regionname="Gene 1")

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic

print(vcc2.result)

##         Score.Test P.value N.Markers
## Gene 1   978.4487    0.18        10
```

By default, the number of permutations in *VC-C2* method is set to 100. To set it to a higher number, use the `Nperm` option:

```
vcc2.result <- VCC2.region(y=pheno, X=covar, Phi=rel.mat,
                           type="ped",
                           filename=geno.file,
                           chr=20,
                           startpos=0,
                           endpos=10000000,
                           map=mymap,
                           regionname="Gene 1",
                           Nperm=500)

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic

print(vcc2.result)

##        Score.Test P.value N.Markers
## Gene 1   978.4487   0.142        10
```

When founder phasing is not already calculated or known, then details on how to estimate haplotypes with ShapeIt are given in section 7. After phasing founders genotypes, it is possible to take the output from ShapeIt (files with the extension `.haps`) and pass them in to `VCC2.region()` function by using the `type=` option. See the arguments of the `VCC2.region()` function in the package manual.

3.5 Using VC-C3

The *VC-C3* method also requires permutation of founder haplotypes to obtain empirical p-values, and so, again in this example, the data in `geno.file` are assumed to be phased with the convention that the "first alleles at all variants are assumed to lie on the same chromosome, and likewise for the second alleles".

```
vcc3.result <- VCC3.region(y=pheno, X=covar, Phi=rel.mat,
                           type="ped",
                           filename=geno.file,
                           chr=20,
                           startpos=0,
                           endpos=10000000,
                           map=mymap,
                           regionname="Gene 1")

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic

print(vcc3.result)

##        Score.Test   P.value N.Markers
## Gene 1   978.4487 0.2049198        10
```

Again, like *VC-C2*, *VC-C3* also uses permutations to obtain a *p*-value. The same parameter `Nperm` can be used to change the number of permutations:

```
vcc3.result <- VCC3.region(y=pheno, X=covar, Phi=rel.mat,
                           type="ped",
                           filename=geno.file,
                           chr=20,
                           startpos=0,
                           endpos=10000000,
                           map=mymap,
                           regionname="Gene 1",
                           Nperm=200)

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic

print(vcc3.result)

##        Score.Test   P.value N.Markers
## Gene 1   978.4487 0.1411562        10
```

In a similar way as in *VC-C2*, when founder phasing is unknown, one can phase founders genotypes using ShapeIt and pass the output in to `VCC3.region()` function by using the `type=` option. See again the arguments of the `VCC3.region()` function in the package manual.

4 Using RVPedigree for genome-wide analysis

If, instead of analysing a single genomic region you would like to analyse multiple regions (e.g. in a genome-wide setting), the overarching `RVPedigree()` function can be used.

First, you need to create a data frame that contains (at least) the following four columns (mind the capitals in the column names) for each region that you would like to run the association test on:

- `Name`: The name of the genomic region. This can be a gene name, for example, or simply `Region_01`.
- `Chr`: The chromosome on which the region is located.
- `StartPos`: The start position (base pair position) of the region.
- `EndPos`: The end position (base pair position) of the region.

The example data has such a region file with three regions:

```
genes.file <- system.file("extdata", "genes2.lst",
                          package="RVPedigree")
genes <- read.table(genes.file,
                    header=TRUE,
                    stringsAsFactors=FALSE)
genes

##      Gene Chr   Start    Stop
## 1 Gene_1  20 9000000 9004000
## 2 Gene_2  20 9005000 9020000
## 3 Gene_3  20 9050050 9075000

colnames(genes) <- c("Name", "Chr", "StartPos", "EndPos")
```

It is important to note that the `RVPedigree()` function has a parameter `pvalThreshold`, which is set to 0.1 by default. This means that any p-value larger than or equal to this threshold is removed from the output of the `RVPedigree()` function. This done to ensure that the output data of a scan of a large number of regions can still be reasonably handled[8].

The following runs the VC-C1 method on each of the three regions, and to make sure we capture all output the `pvalThreshold` is set to 1.0:

```
gwresults <- RVPedigree(method="VCC1",
                        y=pheno,
                        X=covar,
                        Phi=rel.mat,
                        regions=genes,
                        filename=geno.file,
                        type='ped',
                        pvalThreshold=1
                        )

## Estimating null model...
## Null model estimated.
## Starting association analysis of the 3 regions...
## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
2 loci were monomorphic
## Warning in VCC1.region(y = y, X = X, Phi = Phi, type = type, filename
= filename, :  No genotypes available in the region from 9050050 to
9075000 on chromosome 20

gwresults

##        Score.Test   P.value N.Markers
## Gene_1    1768.63 0.3195559         4
## Gene_2    1768.63 0.3195559         6
```

[8]Large data frames consume a lot of computer memory (RAM).

Note that the output contains only two lines, instead of the three expected given the number of regions analyzed. This is because regions that do not contain variants are also removed from the output (see also the warning message in the output above). Note also that the rows of the output are named by the `Name` column in the `genes` data frame.

If the `VCC3afterVCC1` option is set to TRUE (and the `method` option is set to `'VCC1'`), the VC-C3 method is run automatically on the results of the VC-C1 analysis, as shown in the following example. In this example we also show how the `Nperm` parameter can be used to set a different number of permutations (default: 100) for the calculation of the *p*-value. The `Nperm` option is only used if the VC-C2 or VC-C3 methods are used (cf. § 3.4 and § 3.5).

```
gwresults <- RVPedigree(method="VCC1",
                        y=pheno,
                        X=covar,
                        Phi=rel.mat,
                        regions=genes,
                        filename=geno.file,
                        type='ped',
                        Nperm=200,
                        pvalThreshold=0.4,
                        VCC3afterVCC1=TRUE
                        )

## Estimating null model...
## Null model estimated.
## Starting association analysis of the 3 regions...
## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
2 loci were monomorphic
## Warning in VCC1.region(y = y, X = X, Phi = Phi, type = type, filename
= filename, :  No genotypes available in the region from 9050050 to
9075000 on chromosome 20
## 'VCC3afterVCC1' option set, starting VCC3 run...
## Estimating null model...
## Null model estimated.
## Starting association analysis of the 2 regions...
## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
2 loci were monomorphic

gwresults

##        Score.Test   P.value N.Markers
## Gene_1   218.1807 0.2674097         4
## Gene_2   218.1807 0.2690388         6
```

5 Using multiple CPU cores for parallel analysis

Present-day computers have multiple cores per processor (CPU), each of which functions as a single computational unit. Consequently, by distributing parts of the analysis that are independent of each other over these CPU cores and running them in parallel, a significant speed-up can be obtained.

Since in most cases, we expect that analysis of RVPedigree will be run on servers that are shared with other users, automatic detection of the number of cores would lead to various sorts of problems. Therefore, the user has to manually specify the number of cores RVPedigree can use.

Technically, the parallelisation is done using the foreach package[9]. The doParallel package[10] is used as parallel backend to foreach.

Within the RVPedigree package distribution we have implemented two ways in which multicore systems can be used for parallel computation:

- speed up of permutations used to determine the p-values in the VC-C2 and VC-C3 methods.
- analysing multiple genomic regions simultaneously

Details of these implementations can be found in the following sections.

5.1 Parallel permutations to obtain p-values

When only one region is being analyzed, the permutations necessary to obtain p-values from either the VC-C2 or VC-C3 methods can be parallelized in a similar way. To this end the `VCC2.region()` and `VCC3.region()` functions (cf. §3.4 and §3.5) have a `Ncores` parameter. It is up to the user to specify the number of available cores (the default is `Ncores=1`).

For example, to run the example analysis from §3.4 with 2 CPU cores use:

```
vcc2.result <- VCC2.region(y=pheno, X=covar, Phi=rel.mat,
                           type="ped",
                           filename=geno.file,
                           chr=20,
                           startpos=0,
                           endpos=10000000,
                           map=mymap,
                           regionname="Gene 1",
                           Nperm=500,
                           Ncores=2)

## Warning in snpStats::read.pedfile(file = filename, snps = snps2out):
4 loci were monomorphic
```

[9]On CRAN the foreach package can be found at: https://cran.r-project.org/web/packages/foreach/

[10]On CRAN the doParallel package can be found at: https://cran.r-project.org/web/packages/doParallel/

```
print(vcc2.result)

##        Score.Test P.value N.Markers
## Gene 1   978.4487   0.146        10
```

5.2 Parallel analysis of genomic regions

A user-friendly feature of RVPedigree is the ability to analyze a chosen genomic region by specifying a table with the name (e.g. gene name), chromosome, and the start and end base positions of the region (cf. the `RVPedigree()` function in §4). Since the association analysis of a given region is independent of the analysis of other regions, the analysis of multiple regions can easily be distributed over multiple CPU cores. It is up to the user to tell the `RVPedigree()` function how many cores it is allowed to use via the `Ncores` parameter (the default is `Ncores=1`).

If the number of genomic regions is smaller than the number of CPU cores, the `RVPedigree()` function will automatically try to use the remaining cores to parallelize the permutations in case the VC-C2 or VC-C3 methods are used.

For example, if 16 CPU cores are available and five regions are to be analyzed, each region will use one core. If either the VC-C2 or the VC-C3 methods is specified, the remaining $16 - 4 = 11$ cores are split over the five regions so that each region can use $\frac{11}{5} = 2$ (integer division) extra cores to calculate the p-value.

This runs the RVPedigree example of §4 on two cores:

```
gwresults <- RVPedigree(method="VCC1",
                        y=pheno,
                        X=covar,
                        Phi=rel.mat,
                        regions=genes,
                        filename=geno.file,
                        type='ped',
                        Ncores=2,
                        pvalThreshold=1
                        )

## Estimating null model...
## Null model estimated.
## Using 2 cores to analyse the regions in parallel
## Starting association analysis of the 3 regions...

gwresults

##        Score.Test   P.value N.Markers
## Gene_1    1768.63 0.3195559         4
## Gene_2    1768.63 0.3195559         6
```

6 Specifying custom genotype weights

Based on the assumption that rare variants are more likely to be causal and have larger effect sizes, the genotypes are often weighted accordingly. By default, the weights in RVPedigree are computed as in the SKAT package[11], i.e. each variant is given a weight w_i defined by the beta distribution with parameters 1 and 25:

$$w_i = dbeta(f_i, 1, 25), \tag{1}$$

where f_i is the Minor Allele Frequency (MAF) of the variant.

The `weights` parameter of the `RVPedigree()` function and the `*.region()` functions can be used to specify one's own vector of weights.

7 Notes on using ShapeIT to estimate haplotypes

The VC-C2 and VC-C3 methods require the haplotypes of the founders to be present. If this is not the case these need to be estimated using the genotype data you do have. In this section we give some hints on how to run a tool called ShapeIT[12] that is commonly used to phase genotype data.

When running ShapeIt (or any other tool) for use with this package keep the following in mind:

- Since in most practical situations the genotype data of a cohort undergoes a centralized quality control analysis, it makes sense to do quality control of the phasing at the same time.

- ShapeIt can make use of multiple processor cores. When phasing a (large) data set, don't forget to use this option (for ShapeIt add the command line option `--thread 8`, which assumes you have 8 CPU cores).

ShapeIt accepts genotype and pedigree data in either Plink `.ped`/`.map` format or in the binary `.bed`/`.bim`/`.fam` format.

For the example below, let's assume you have your genotype and pedigree data stored in Binary Plink files called `genotypes.bed`, `genotypes.bim` and `genotypes.fam`.

It is important to note that ShapeIt phases the data per chromosome, and therefore it also requires the genotype data to be stored such that there is one file per chromosome. The following command splits an existing data set in Plink format with file name `gwasdata` into chunks of one chromosome each:

[11]The SKAT package can be found on CRAN: https://cran.r-project.org/web/packages/SKAT/.

[12]See http://www.shapeit.fr for more detailed information than can be found here. Note also that part of the information in this section was taken directly from the ShapeIt documentation.

```
for chr in $(seq 1 22); do
     plink --file gwasdata \
           --chr ${chr} \
           --recode \
           --make-bed \
           --out gwas.chr${chr};
done
```

To run ShapeIt on the first chromosome, run:

```
shapeit \
  --input-bed gwas-chr1.bed gwas-chr1.bim gwas-chr1.fam \
  --input-map genetic_map.txt \
  --output-max gwas-chr1.phased.haps gwas-chr1.phased.sample
```

References

[1] Lakhal-Chaieb L, Oualkacha K, Richards JB, Greenwood CM. *A rare variant association test in family-based designs and non-normal quantitative traits*. Stat Med. 2015 Sep 29. doi: 10.1002/sim.6750.

[2] Oualkacha K, Dastani Z, Li R, Cingolani PE, Spector TD, Hammond CJ, Richards JB, Ciampi A, Greenwood CMT. *Adjusted sequence kernel association test for rare variants controlling for cryptic and family relatedness*. Genet Epidemiol. 2013 May;37(4):366-76. doi:10.1002/gepi.21725.