Package Ravages (RAre Variant Analysis and GEnetic Simulation)

Herve Perdry and Ozvan Bocher

2021-10-11

Introduction

Ravages was developped to simulate genetic data and to perform rare variant association tests (burden tests and the variance-component test SKAT) on different types of phenotypes (Bocher et al., 2019, doi: 10.1002/gepi.22210, Bocher et al., 2020, doi:10.1038/s41431-020-00792-8) at a genome-wide scale. Ravages relies on the package Gaston developped by Herve Perdry and Claire Dandine-Roulland. Most functions are written in C++ thanks to the packages Rcpp, RcppParallel and RcppEigen.

Functions of Ravages use bed.matrix to manipulate genetic data as in the package Gaston (see documentation of this package for more details).

In this vignette, we illustrate how to perform rare variant association tests on real data on different phenotypes at a genome-wide scale. A second vignette is available showing how to simulate genetic data and how to use them for power calculation. To learn more about all options of the functions, the reader is advised to look at the manual pages.

Example of analysis using LCT data

Below is an example of an association analysis and previous steps of data filtering using the dataset `LCT.matrix` available with the package Ravages. This dataset containts data from the 1000Genome project in the locus containing the Lactase gene. In this example, we look for an association between rare variants and the european populations of 1000Genomes. The population of each individual is available in the dataframe `LCT.matrix.pop1000G`. A classical analysis by gene is performed, and an analysis using the strategy "RAVA-FIRST".

Details about the "RAVA-FIRST" strategy and each function is given right after this example.

```
# Import data in a bed matrix
x <- as.bed.matrix(x=LCT.matrix.bed, fam=LCT.matrix.fam, bim=LCT.snps)
# Add population
x@ped[,c("pop", "superpop")] <- LCT.matrix.pop1000G[,c("population", "super.population")]

# Select EUR superpopulation
x <- select.inds(x, superpop=="EUR")
x@ped$pop <- droplevels(x@ped$pop)

# Group variants within know genes by extending their positions
# 500bp upstream and downstream
# (the function uses build 37 unless told otherwise)
x <- set.genomic.region(x, flank.width=500)
```

```r
# a quick look at the result
table(x@snps$genomic.region, useNA = "ifany")
```

```
##
## R3HDM1  UBXN4    LCT   MCM6   DARS   <NA>
##   2047   1207   1454   1149    924   1295
```

```r
# Filter variants with maf in the entire sample lower than 1%
# And keep only genomic region with at least 10 SNPs
x1 <- filter.rare.variants(x, filter = "whole", maf.threshold = 0.01, min.nb.snps = 10)
table(x1@snps$genomic.region, useNA="ifany")
```

```
##
## R3HDM1  UBXN4    LCT   MCM6   DARS
##    268    172    208    163    136
```

```r
# run burden test CAST, using the 1000Genome population as "outcome"
# Null model for CAST
x1.H0.burden <- NullObject.parameters(x1@ped$pop, ref.level = "CEU",
                                      RVAT = "burden", pheno.type = "categorial")
burden(x1, NullObject = x1.H0.burden, burden = "CAST", cores = 1)
```

```
## Categorial phenotype
```

```
##             p.value is.err
## R3HDM1 1.300274e-04      0
## UBXN4  1.993354e-05      0
## LCT    4.489474e-08      0
## MCM6   3.142808e-08      0
## DARS   2.064661e-03      0
```

```r
# run SKAT, using the 1000Genome population as "outcome"
# COnstruct null model for SKAT, then run test with only a few permutations
x1.H0.SKAT <- NullObject.parameters(x1@ped$pop, RVAT = "SKAT", pheno.type = "categorial")
SKAT(x1, x1.H0.SKAT, params.sampling=list(perm.target = 10, perm.max = 500))
```

```
## Categorial phenotype
## permutations
```

```
##            stat      p.perm       p.chi2      p.value
## R3HDM1 6.445786 0.001996008 3.887497e-06 3.887497e-06
## UBXN4  2.136390 0.007984032 3.023460e-03 3.023460e-03
## LCT    3.623174 0.001996008 4.356358e-04 4.356358e-04
## MCM6   2.923228 0.003992016 4.466921e-03 4.466921e-03
## DARS   2.132967 0.068322981 9.971173e-02 6.832298e-02
```

```r
# run a similar analysis but using the RAVA-FIRST approach with WSS
# RAVA-FIRST(x1, filter = "whole", maf.threshold = 0.01, min.nb.snps = 10,
#           burden = TRUE, x1.H0.burden, SKAT = F)
```

Defining genomic regions

For rare variant association tests, the unit of analysis is not a single variant but a genomic region or 'testing unit', typically a gene. The first step of the analysis is therefore to group variants into genomic regions. This can be done using the function **set.genomic.region()** and known regions positions. It works on a bed.matrix (see Gaston) and simply adds a column "genomic.region" to the slot `x@snps` containing the genomic region (a factor) assigned to each variant. The positions of regions should be given to *regions* as a dataframe in a bed format containing the following columns: *Chr*, *Start*, *End*, *Name*. The dataframe should absolutely be ordered in the genome order, as well as the levels of *regions$Name*.

By default, any variant being outside the gene positions won't be annotated. Regions boundaries can be extended to include more variants using the argument *flank.width* corresponding to the number of base pairs upstream and downstream the region to which expand the positions. If *flank.width=Inf*, each variant will be assigned to the nearest region.

If two regions overlap, variants in the overlapping zone will be attributed to both regions, separated by a comma. By default, *split=TRUE* in **set.genomic.region()**, which means that variants attributed to multiple regions will be duplicated in the bed matrix. This is done by calling the function **bed.matrix.split.genomic.region()** which takes a bed matrix as argument (x) and duplicates variants being assigned to multiple regions separated by *split.pattern*. If *changeID=TRUE*, the id from the bedmatrix will be changed to the format chr:pos:A1:A2:genomic.region to distinguish the duplicated variants.

The files **genes.b37** and **genes.b38** available in Ravages which contain gene positions from ENSEMBL versions GRCH37 and GRCH38 can be used as *regions*.

Running the following toy example will show you the behavior of these functions.

```
# Example bed matrix with 4 variants
x.ex <- as.bed.matrix(x=matrix(0, ncol=4, nrow=10),
                      bim=data.frame(chr=1:4, id=paste("rs", 1:4, sep=""),
                                     dist = rep(0,4), pos=c(150,150,200,250),
                                     A1=rep("A", 4), A2=rep("T", 4)))

# Example genes dataframe
genes.ex <- data.frame(Chr=c(1,1,3,4), Start=c(10,110,190,220), End=c(170,180,250,260),
                       Gene_Name=factor(letters[1:4]))

# Attribute genomic regions without splitting the variants
# attributed to multiple genomic regions
x.ex <- set.genomic.region(x.ex, regions = genes.ex, split = FALSE)
x.ex@snps$genomic.region

# Split genomic regions
x.ex.split <- bed.matrix.split.genomic.region(x.ex, split.pattern = ",")
x.ex.split@snps$genomic.region
```

Rare variant selection

Frequency filter

To perform rare variant analysis, it is also important to define what is a rare variant in order to leave out common ones that are expected to have a weaker functional impact. The function **filter.rare.variants()** enables to keep only variants with a MAF (Minor Allele Frequency) below a given threshold while leaving out monomorphic variants. This function uses and returns a bed.matrix which can be filtered in three different ways:

- If *filter="whole"*, all the variants with a MAF lower than the threshold in the entire sample will be kept.
- If *filter="controls"*, all the variants with a MAF lower than the threshold in the controls group will be kept. In this situation, the controls group needs to be specified to the argument *ref.level*.
- If *filter="any"*, all the variants with a MAF lower than the threshold in any of the groups will be kept.

It is also possible to specify the minimum number of variants needed in a genomic region to keep it using the parameter *min.nb.snps*, as well as the minimum cumulative MAF using *min.cumulative.maf*.

For the *any* and *controls* filters, the group of each individual should be given as a factor to *group*.

Rare variant association tests

We have implemented two burden tests extensions (CAST and WSS, doi:10.1002/gepi.22210) and an extension of the variance-component test SKAT (doi:10.1038/s41431-020-00792-8) to perform the association tests between a region and more than two groups of individuals.

The general idea of burden tests is to compute a genetic score per individual and per genomic region and to test if its distribution differs between the different groups of individuals. To extend these tests to more than two groups of individuals, a non-ordinal multinomial regression is used. The independant variable in this regression is the genetic effect of the region represented by the genetic score. Covariates can be added in the model. In addition to the genetic scores CAST and WSS directly implemented in the package, the user can specify another genetic score for the regression.

The variance-component test SKAT looks at the dispersion of genetics effects of rare variants. A geometrical interpretation of the test was used for its extension to more than two groups of individuals. Covariates can also be included in this model.

Genetic score for burden tests

We have implemented two functions to compute CAST and WSS scores respectively. These functions return a matrix with one row per individual and one column by genomic region. They are directly called in the function **burden()** if these scores are used to perform the association tests. It is also possible to compute genetic scores in a genomic region based on a vector of weights for each variant using the function **burden.weighted.matrix()**. It is important to note that all burden functions compute a score for the rare alleles. Therefore, if the reference allele is rare, the alleles will be flipped and the reference allele will be counted in the score instead of the alternative allele. This can be avoided only in the **CAST()** score if *flip.rare.alleles = FALSE* (it is set at *TRUE* by default).

CAST

CAST is based on a binary score which has a value of one if an individual carries at least one variant in the considered genomic region, and 0 otherwise. A MAF threshold for the definition of a rare variant is therefore needed (argument *maf.threshold*). This score can be computed using the function **CAST()** as shown here on the LCT data:

```
# Calculation of the genetic score with a maf threshold of 1%
CAST.score <- CAST(x = x1, genomic.region = x1@snps$genomic.region, maf.threshold = 0.01)
head(CAST.score)
```

```
##         R3HDM1 UBXN4 LCT MCM6 DARS
## HG00096      0     0   1    1    1
## HG00097      1     0   0    0    0
```

```
## HG00099      1     0   0     0     0
## HG00100      0     1   0     0     0
## HG00101      0     1   0     0     0
## HG00102      1     0   0     0     1
```

WSS

WSS (Weighted Sum Statistic) is based on a continuous score giving the highest weights to the rarest variants:

$$WSS_j = \sum_{i=1}^{R} I_{ij} \times w_i$$

with

$$w_i = \frac{1}{\sqrt{t_i \times q_i \times (1 - q_i)}}$$

and

$$q_i = \frac{n_i + 1}{2t_i + 1}$$

Where n_i is the total number of minor alleles genotyped for variant i, t_i is the total number of alleles genotyped for variant i and I_{ij} is the number of minor alleles of variant i for the invidual j. In the original method, each variant is weighted according to its frequency in the controls group. In our version of WSS, the weights depend on allele frequencies calculated on the entire sample ; this avoids using permutations. The function **WSS()** can be used to compute the WSS score as shown on the LCT data:

```
WSS.score <- WSS(x = x1, genomic.region = x1@snps$genomic.region)
head(WSS.score)
```

```
##            R3HDM1    UBXN4       LCT    MCM6     DARS
## HG00096 0.0000000 0.000000 0.8185268 1.26932 1.418436
## HG00097 0.8185268 0.000000 0.0000000 0.00000 0.000000
## HG00099 1.0019881 0.000000 0.0000000 0.00000 0.000000
## HG00100 0.0000000 1.001988 0.0000000 0.00000 0.000000
## HG00101 0.0000000 1.001988 0.0000000 0.00000 0.000000
## HG00102 1.0019881 0.000000 0.0000000 0.00000 1.001988
```

Other genetic scores

It is also possible to compute other genetic scores based on variants weights using the function **burden.weighted.matrix()**. The weights should be given as a vector to *weights* (with the length as the number of variants). The genetic score will be compute as:

$$Score_j = \sum_{i=1}^{R} I_{ij} \times w_i$$

with w_i the weight of each variant in *weights*, and I_{ij} the number of minor alleles for individual j in variant i.
Here is an example corresponding to a genetic score with all the weights at 1, i.e. counting the number of minor alleles:

```
Sum.score <- burden.weighted.matrix(x = x1, weights = rep(1, ncol(x1)))
head(Sum.score)
```

```
##          R3HDM1 UBXN4 LCT MCM6 DARS
## HG00096      0     0   1    2    2
## HG00097      1     0   0    0    0
## HG00099      1     0   0    0    0
## HG00100      0     1   0    0    0
## HG00101      0     1   0    0    0
## HG00102      1     0   0    0    1
```

Regressions

We have extended burden tests using a non-ordinal multinomial regression model. Let consider C groups of individuals including a group of controls ($c = 0$) and $C - 1$ groups of cases with different sub-phenotypes of the disease. We can compute $C - 1$ probability ratios, one for each group of cases:

$$ln\frac{P(Y_j = c)}{P(Y_j = 0)} = \beta_{0,c} + \beta_{G,c}X_G + \beta_{k1,c}K_1 + ... + \beta_{kl,c}K_l$$

where Y_j corresponds to the phenotype of the individual j and K_l is a vector for the lth covariate with the corresponding coefficient β_{kl}. The genetic effect is represented by X_G and correspond to the genetic score (for example CAST or WSS) with $\beta_{G,c}$ the log-odds ratio associated to this burden score.

The p-value associated to the genetic effect is calculated using a likelihood ratio test comparing this model to the same model without the genetic effect (null hypothesis). If only two groups are compared, a classical logistic regression is performed.

This regression can be performed on a bed.matrix using the function **burden()** which relies on the package mlogit. Parameters under the null model can be obtained using the function **NullObject.parameters()**. To generate this null model, the phenotype (argument *pheno*) of each individual should be given as a factor, and the potential covariates to include in the model should be given as a matrix to the argument *data* (one row per individual and one column per covariate). If only a subset of covariates from *data* are to be included in the model, a R formula should be given to *formula* with these covariates, otherwise all the covariates will be included. In addition, the reference group should be given to the argument *ref.level*, i.e. all odds ratios will be computed in comparison to this group of individuals. The choice of the reference group won't affect the p-value. As the parameters needed to run the association tests depend on the type of test performed (burden tests or SKAT), and the type of phenotype (continuous or categorial), both arguments shouls be given to **NullObject.parameters()** (*RVAT* and *pheno.type* respectively).

The function **NullObject.parameters()** will return a list with the parameters to use in **burden()** to run the burden test, including the Log-Likelihood computed under the null model, the argument *data* with the covariates to include determined from the argument *formula*.

Once the null model has been created, the function **burden()** can be used to perform burden tests. To do so, the user needs to give the results from **NullObject.parameters()** to the argument *NullObject*, and needs to specify the genomic region associated to each variant (argument *genomic.region*).

The CAST or WSS genetic scores can be directly calculated in the regression (*burden="CAST"* or *burden="WSS"*). The user can also use another genetic score in the regression, which has to be specified as a matrix with one row per individual and one column per genomic region to *burden*. In this situation, no bed matrix is needed, and the result from **burden.weighted.matrix()** can be used directly.

To shorten the computation time, calculations are parallelised; the argument *cores*, set at 10 by default, controls the parallelisation.

The function **burden()** will return the p-value associated to the regression for each genomic region. If there is a convergence problem with the regression, the function will return 1 in the column *is.err*. The effect size (odds ratio for categorial phenotypes and beta value for continuous phenotypes) associated to each group of cases compared to the reference group (*NullObject$ref.level*) with its confidence interval at a given alpha threshold (argument *alpha*) can also be obtained with *get.effect.size=TRUE*.

An example of the p-value and OR calculation with its 95% confidence interval using WSS on the LCT data is shown below with or without the inclusion of covariates. The outcome here corresponds to the population from 1000Genome.

```
# Null model
x1.H0 <- NullObject.parameters(x1@ped$pop, ref.level = "CEU",
                               RVAT = "burden", pheno.type = "categorial")
# WSS
burden(x = x1, NullObject = x1.H0, burden ="WSS",
      alpha=0.05, get.effect.size=TRUE, cores = 1)

# Sex + a simulated variable as covariates
sex <- x1@ped$sex
u <- runif(nrow(x1))
covar <- cbind(sex, u)
# Null model with the covariate "sex"
x1.H0.covar <- NullObject.parameters(x1@ped$pop, ref.level = "CEU",
                                     RVAT = "burden", pheno.type = "categorial",
                                     data = covar, formula = ~ sex)

# Regression with the covariate "sex" without OR values
# Using the score matrix WSS computed previously
burden(NullObject = x1.H0.covar, burden=WSS.score, cores = 1)
```

Finally, using Ravages, it is also possible to perform burden tests with a continuous phenotype by specifying *pheno.type = "continuous"*, and by giving a numeric vector to *pheno* as showed below:

```
# Random continuous phenotype
set.seed(1) ; pheno1 <- rnorm(nrow(x1))
# Null model
x1.H0.continuous <- NullObject.parameters(pheno1, RVAT = "burden",
                                          pheno.type = "continuous")
# Test CAST
burden(x1, NullObject = x1.H0.continuous, burden = "CAST", cores = 1)
```

Functionally-informed burden tests

Finally, we offer to perform functionally-informed burden tests to take into account functional information. To do so, we propose to define genomic regions (for example a gene of interest) and to integrate sub-scores in the regression corresponding to different functional categories (for example distinguish between coding and regulatory variants). A burden test will then be applied for each genomic region but will take into account the different categories of variants. The number of freedom corresponds to: (number of groups of individuals - 1) * number of sub-scores.

To attribute variants to genomic region and subregions, the function **set.genomic.region.subregion()** needs to be used which is very similar to **set.genomic.region()**. Two dataframes (in bed format) should be included: one for the large genomic regions (regions), and one for the subregions (subregions). Two columns will be added to *x@snps*: *genomic.region* and *SubRegion*.

The function **burden.subscores()** can then be applied on this bed matrix, which works similarily as **burden()**, with the extra argument *SubRegion* corresponding to the vector with the subregions on which sub-scores should be computed. In addition, the argument *burden.function* replaces the argument *burden* in **burden()** and requires to give a function which determines how the genetic score is computed (for example *CAST* or *WSS*).

Below is an example of the two functions where subscores correspond to coding and regulatory categories in the LCT locus.

```
# *** Functionally-informed WSS analysis ***
# Attribution of variants to regions and subregions
x2 <- set.genomic.region.subregion(x, regions = genes.b37,
                                   subregions = subregions.LCT)
# Burden test
burden.subscores(x2, x1.H0.burden, cores = 1)
```

```
## Categorial phenotype

##             p.value n_subscores is.err
## R3HDM1           NA           3      1
## UBXN4  2.626565e-21           2      0
## LCT              NA           3      1
## MCM6   3.488717e-31           2      0
## DARS             NA           3      1
```

SKAT

We also extended the variance-component test SKAT using a geometric interpretation. Unlike the burden tests, the is no burden calculated in this test: the distribution of the genetic effects in the genomic region is compared to a null distribution. SKAT is based on a linear mixed model where the random effects correspond to the genetic effects.

Before running SKAT, the function **NullObject.parameters()** first needs to be called as for the burden tests, but by specifying *RVAT = "SKAT"*. As before, potential covariates could be included as a matrix to *data*. If only some of them are to be included, they should be given as a R formula to *formula*. This function will compute parameters to use the **SKAT()** function, and should be given to this function using the argument *NullObject*.

To compute the p-values, a chi-square approximation is used based on the statistics' moments. The moments can either be estimated using a sampling procedure, or be analytically computed using the method from Liu et al. 2008. The chi-square approximation can be based on the first three moments (*estimation.pvalue = "skewness"*), or on moments 1, 2 and 4 to have a more precise estimation of the tail distribution (*estimation.pvalue = "kurtosis"*). If *get.moments = "theoretical"* and *estimation.pvalue = "skewness"*, it is equivalent to the "liu" method in the SKAT package, and if *estimation.pvalue = "kurtosis"*, it corresponds to the "liu.mod" method in the SKAT package. If *debug = TRUE*, the statistics' moments will be returned in addition to the p-values.

If the sample size is lower than 2000, we recommand to use the sampling procedure. If no covariates are present, a simple permutation procedure can be used (*get.moments = "permutations"*), otherwise, a boostrap sampling should be used (*get.moments = "bootstrap"*). For those two situations, a sequential procedure is used to compute the p-values: permutated statistics are computed and each one is compared to the observed statistics. The sampling procedure stops when either *perm.target* (the number of times a permutated statistics should be greater than the observed statistics) or *perm.max* (the maximum number of permutations to perform) is reached. P-values are then computed in two different ways: if *perm.target* is reached, the p-value is computed as *perm.target* divided by the number of permutations performed to reach this value; if *perm.max* is reached before *perm.target* (that is, for pretty small p-values), p-values are computed using the chi-square approximation based on moments estimated from the permutated statistics. *perm.target* and *perm.max* should be given as a list to the argument *params.sampling* of SKAT.

If the sample size is bigger than 2000, the analytical calculation from Liu et al. can be used to compute the theoretical moments. In this situation, it is possible to parallelise the calculations using the argument *cores*, set at 10 by default.

It is possible to clearly ask for a specific method to compute the moments using *get.moments = "permutations"*, *"bootstrap"* or *"theoretical"*. By default, *get.moments = "size.based"*, and the method will depend on the sample size.

An example of the SKAT function by specifying the "permutations" or "theoretical" method is shown.

```
# Null model
x1.null <- NullObject.parameters(x1@ped$pop, RVAT = "SKAT", pheno.type = "categorial")
# Permutations because no covariates
SKAT(x1, x1.null, get.moments = "permutations", debug = TRUE,
     params.sampling = list(perm.target = 100, perm.max =5e4))
# Theoretical on 1 core
SKAT(x1, x1.null, get.moments = "theoretical", debug = TRUE, cores = 1)
```

It is also possible to perform the SKAT test on a continuous phenotype by using the argument *pheno.type = "continuous"* in **NullObject.parameters()** before the **SKAT()** function:

```
# Random continuous phenotype
set.seed(1) ; pheno1 <- rnorm(nrow(x1))
# Null Model with covariates
x1.H0.c <- NullObject.parameters(pheno1, RVAT = "SKAT", pheno.type = "continuous",
                                 data = covar)
# Run SKAT
SKAT(x1, x1.H0.c)
```

RAVA-FIRST (RAre Variant Analysis using Functionally-InfoRmed STeps)

Ravages also offers the possibility to analyse rare variants using the 'RAVA-FIRST' strategy, composed of three main steps: defining testing units, filtering rare variants, and running functionnaly informed burden tests.

Definition of testing units: CADD regions

CADD regions are non-overlapping regions defined genome-wide using the variant pathogenicity score CADD that can be used as testing units: regions are defined between variants observed at least two times in GnomAD with a high CADD score. This CADD score is not the original CADD score v1.4 oublished by Rentzch et al., but an adjusted score that take into account three types of functional categories : coding, regulatory and intergenic categories. This adjusted score enables to find the most important functional variants within each of those three categories.

To attribute adjusted CADD scores to variants, the function **adjustedCADD.annotation()** can be used. It will either annotate variants in the bed matrix using a provided file with adjusted CADD scores to *variant.scores* (with columns 'chr', 'pos', 'A1', 'A2', 'adjCADD'), or by downloading the adjusted scores from https://lysine.univ-brest.fr/RAVA-FIRST/ in Ravages repository.

To assign variants to CADD regions and to the functional categories, the function **set.CADDregions()** can be used. It will add *genomic.region* to the bed matrix with the corresponding CADD regions present in the file "CADDRegions.2021.hg19.bed.gz" and *SubRegion* with the functional area present in the file "FunctionalAreas.hg19.bed.gz", both directly downloaded from https://lysine.univ-brest.fr/RAVA-FIRST/ in the repository from the package Ravages. In addition, *adjCADD.Median* will be added to *x@snps* which correspond to the median adjusted CADD score of variants observed at least two times in GnomAD and can be used for variant filtering as explained later.

```
# Annotation of variants with adjusted CADD scores
x <- adjustedCADD.annotation(x)
# Attribution of CADD regions
x.CADDregions <- set.CADDregions(x)
```

Filtering of rare variants: region-dependant thresholds

In RAVA-FIRST, we propose a new approach to select rare variants to include in the RVAT that keeps within each CADD region only the variants with an adjusted CADD score greater than the median observed in GnomAD. To this end, the function **filter.adjustedCADD()** can be used which relies on the same arguments than **filter.rare.variants()** for the frequency parameters. Within this function, SNVs of the bed matrix will be directly annotated with the adjusted CADD scores using the file "AdjustedCADD_v1.4_202108.tsv.gz" that will be downloaded from https://lysine.univ-brest.fr/RAVA-FIRST/ into the repostitoy of the package Ravages or with a provided dataframe *variant.scores*. If the bed matrix has previously been annotated by **adjustedCADD.annotation()**, **filter.adjustedCADD()** will filter rare variants based on `x@snps$adjCADD` without annotating the variants to gain in computation time. Only the variants with a score greater than the median will be kept. It is necessary that the bed matrix is first annotated with **set.CADDregion()** to get the median of each variant from its corresponding CADD region.

```
# Keep only CADD regions with 2 variants and variants with a MAF greater than 1%
# and with an adjusted CADD score greater than the median
x.median <- filter.adjustedCADD(x.CADDregions, maf.threshold = 0.01, min.nb.snps = 2)
```

Functionally-informed burden tests

CADD regions can overlap different types of functional categories (coding, regulatory or intergenic regions). To take them into account in RAVA-FIRST, we propose to use the functionally-informed burden test **burden.subscores()** with subscores in the regression, one for each functional category, within each CADD region. There will be at most three sub-scores in the regression. **burden.subscores()** can be directly applied on the bed matrix obtained from **set.CADDregions()** or **filter.adjustedCADD()**.
SKAT() can also be applied on the bed matrix with variants grouped and filtered according to RAVA-FIRST but it will correspond to the classical SKAT analysis without the integration of functional information.

```
# Functionally-informed WSS analysis
x.burden <- burden.subscores(x.median, x1.H0.burden, burden.function = WSS)
# SKAT analysis
x.SKAT <- SKAT(x.median, x1.H0.SKAT)
```

Complete RAVA-FIRST approach and whole-genome analysis

RAVA-FIRST enables to perform RVAT in the whole genome. To facilitate the analysis, we propose the function **RAVA.FIRST()** which gathers all the previous functions and enables to directly performs the whole RAVA-FIRST strategy on a bed matrix. It simply needs the bed matrix and the NullObject (that can be obtained from **NullObject.parameters()** and should be given to *H0.burden* and/or *H0.SKAT*) and optional filtering arguments and parameters for the RVAT previously mentionned. *burden.parametrs* should be a list containing *burden.function* and *get.effect.size* if *burden = TRUE* and *SKAT.parameters* should be a list containing *weights*, *get.moments estimation.pvalue*, *param.sampling* and *debug* if *SKAT = TRUE*. If no such list is provided, the default parameters of **burden.subscores()** and **SKAT** will be used. Due to time and memory management, we advise the user to import the data and apply the different functions of annotation, filtering and association (or directly **RAVA.FIRST()**) chromosome by chromosome for large datasets as shown in the example below.

```
# *** RAVA-FIRST analysis with functionally-informed burden and SKAT
# Chromosome by chromosome
res.bychr <- vector("list", 22)
for(chr in 1:22){
  x <- read.bed.matrix(paste0(path_file, prefix_vcf, chr, ".vcf.gz"))
  res.bychr[[chr]] <- RAVA.FIRST(x, H0.burden = H0.burden,
                                 burden.function = WSS, H0.SKAT = H0.SKAT)
}
```

Data management

Data in plink format or in vcf format can be loaded in R using the functions **read.bed.matrix()** and **read.vcf()** respectively from the package gaston.

If the data for the controls and the different groups of cases are in different files, they can be loaded separately and then combined using the function **gaston::rbind()** as long as the same variants are present between the different groups of individuals.

An example is given below where the simulated data have been split according the the group of each individual, and then combined in a bed.matrix:

```
# Selection of each group of individuals
CEU <- select.inds(x1, pop=="CEU")
CEU
```

```
## A bed.matrix with 99 individuals and 947 markers.
## snps stats are set
##   There are  748  monomorphic SNPs
## ped stats are set
```

```
FIN <- select.inds(x1, pop=="FIN")
FIN
```

```
## A bed.matrix with 99 individuals and 947 markers.
## snps stats are set
##   There are  771  monomorphic SNPs
## ped stats are set
```

```
GBR <- select.inds(x1, pop=="GBR")
GBR
```

```
## A bed.matrix with 91 individuals and 947 markers.
## snps stats are set
##   There are  792  monomorphic SNPs
## ped stats are set
```

```
# Combine in one file:
CEU.FIN.GBR <- rbind(CEU, FIN, GBR)
CEU.FIN.GBR
```

```
## A bed.matrix with 289 individuals and 947 markers.
## snps stats are set
##   There are  515  monomorphic SNPs
## ped stats are set
```