# Package Ravages (RAre Variant Analysis and GEnetic Simulation)

*Herve Perdry and Ozvan Bocher*

*2018-07-20*

```
library("mlogit")
library("knitr")
require("Ravages")
```

## Introduction

Ravages can be used to simulate genetic data and to perform burden tests, a type of rare variant association tests. 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 show how to simulate genetic data and we illustrate association tests using these simulated data. To learn more about all options of the functions, the reader is advised to look at the manual pages.

## Defining genomic regions

For rare variant association tests, the unit of analysis is not a single variant but a genomic region, typically a gene. The difficulty in this type of study is therefore to define the genomic region. In this package, two methods are proposed to group variants into genomic regions. The first one, called **region.by.pos()** groups the variants based on the distance between them and on the maximum number of groups we want to make. Indeed, the distance between each adjacent variant pair is calculated and the maximum distance between two variants within a genomic region increases until we have less groups than the allowed maximum number of groups. The second one, called **region.by.gene()** uses positions of known genes to group variants into genomic regions. If the option *include.all=FALSE* is used, only variants within known genes will be assigned to a genomic region, the other ones being left out. If the option *include.all=TRUE*, each variant will be assigned to the nearest gene. The files **hg37** and **hg38** can be used to define gene positions in this function.

## Simulation of genetic data

Genetic data can be simulated using the package Ravages. The procedure is similar to the one from Suzanne Leal et al used in the programm SeqPower but unlike this software, Ravages allows the simulation of more than two groups of individuals. Using functions from our package , it is possible to compute MAF in each group of individuals based on MAF in the general population and GRR values.

### Calculation of frequencies in each group of individuals

The GRR corresponds to the genetic relative risk which reflects the increased risk of a disease for a given genotype compared to a reference genotype, here the homozygous for the reference allele. More precisely, the GRR associated to the heterozygous genotype in the group $c$ corresponds to the ratio between the penetrance

of phenotype $c$ for the heterozygous genotype and the penetrance of phenotype $c$ for the homozygous for the reference allele as follow:

$$GRR_{Aa} = \frac{P(Y = c|Aa)}{P(Y = c|AA)}$$

The frequency of each genotype in each group of cases $c$ can be calculated using Bayes theorem:

$$P(Aa|Y = c) = \frac{P(Y = c|Aa) * P(Aa)}{\sum_{Geno=AA,Aa,aa} P(Y = c|Geno) * P(Geno)} = \frac{GRR_{Aa} * P(Aa)}{P(AA) + GRR_{Aa} * P(Aa) + GRR_{aa} * P(aa)}$$

With A the reference allele, and a the alternate allele. The three genotype frequencies can then be calculated in the controls group using the rule of total probability:

$$P(Geno|Y = 1) = P(Geno) - \sum_{c=2}^{c=C} P(Geno|Y = c) * P(Y = c)$$

The function **group.mafs.GRR()** enables to perform these last calculations to compute the MAF in each group of individuals. To do so, the user needs to give *P(Y=c)* which corresponds to $baseline_c$, the prevalence of each group of cases ; and the *GRR* values. The GRR values need to ben in a matrix given by the user with one row per cases group and one column per variant. If there is no supposed link between the genetic relative risk associated to the heterozygous genotype and the genetic relative risk associated to the homozygous for the alternate allele (general model of the disease, *model* = "*general*"), the user needs to specify two GRR matrices: one for $GRR_{Aa}$ and one for $GRR_{AA}$ in a list. If *model="recessive", "multiplicative" or "dominant"*, only one GRR matrix is needed. To help the user with the construction of this GRR matrix, we implemented the function **compute.GRR.matrix()**. To use this function, the user needs to specify how the GRR needs to be calculated (argument *GRR*). The user can choose to give the same GRR to all the variants (*GRR="constant"*), its value being specified to the argument *GRR.value*. It is also possible to compute the GRR by using the formula from the publication presenting the method SKAT (*GRR="SKAT"*). Finally, the user can choose to calculate the GRR in another way depening on MAF of the variants in the general population (*GRR="variable"*); in this case, the function to compute the GRR depending on the MAF needs to be given to the argument *GRR.formula*. In the two last situations, a file containing the MAF in the general population with at least a column "maf" and a column "gene" should be given to the function (argument *file.pop.maf*). Two such files are available with this package: the file *Kryukov* containing MAF simulated under a demographic model of Kryukov and the file *GnomADgenes* containing MAF from the population NFE in GnomAD. As these files contain MAF for multiple gene, the user needs to specify which gene to choose to the argument *select.gene*. The user needs finally to specifiy by which the GRR are multiplicated between each group of cases compared to the first group of cases (number of values: number of cases groups - 1). **compute.GRR.matrix()** will return a GRR matrix in the appropriate format for the function **group.mafs.GRR()**. Examples of these two functions are show below:

```
#GRR calculated using the formula from the SKAT paper with two groups of cases,
#the second group having GRR values twice as high as the first one.

GRR.del <- compute.GRR.matrix(GRR = "SKAT", file.pop.maf = Kryukov, n.case.groups = 2,
                              GRR.multiplicative.factor=2, select.gene = "R1")
GRR.del[,1:5]
```

```
##           [,1]      [,2]     [,3]      [,4]      [,5]
## [1,]  5.037728  6.222656 12.34472  9.042877  8.133663
## [2,] 10.075455 12.445313 24.68944 18.085755 16.267327
```

```
#Calculation of genotype frequencies in the two groups of cases and the controls group
#The previous GRR matrix is used with a multiplicative model of the disease
#All variants are deleterious and the prevalence in each group of cases is 0.001

MAF.groups <- group.mafs.GRR(file.pop.maf = Kryukov, select.gene="R1", GRR = GRR.del,
                             baseline = c(0.001, 0.001), model = "multiplicative")
MAF.groups[,1:5]
```

```
##                  [,1]         [,2]         [,3]         [,4]         [,5]
## controls 9.375276e-05 2.785699e-05 5.403418e-07 3.246158e-06 5.972867e-06
## cases_1  4.784006e-04 1.762618e-04 6.912999e-06 3.011198e-05 4.969452e-05
## cases_2  9.563437e-04 3.524614e-04 1.382590e-05 6.022214e-05 9.938410e-05
```

### Simulation of genotypes

In addition to calculate the genotype frequencies in each group of individuals, it is possible to directly simulate these genotypes. This can be done using the function **random.bed.matrix.GRR()** which relies on the function **group.mafs.GRR()** explained previously. The arguments *file.pop.maf*, *select.gene*, *baseline* and *model* are the same as in the function **group.mafs.GRR()**. In **random.bed.matrix.GRR()**, the proportion of deleterious and protective variants simulated in the genomic region should be specified to *prop.del* and *prop.pro* respectively. The argument *GRR.matrix* should contain a matrix with GRR values as if all the variants were deleterious. If *model="general"*, two GRR matrices need to be given (one for the heterozygous genotype and the other for the homozygous genotype for the alternate allele) as a list to the argument *GRR.matrix*. If the user wants to simulate protective variants in addition to deleterious variants, the same type of argument as *GRR.matrix* should be given to *GRR.matrix.pro* but with GRR values as if all variants were protective. If the argument *GRR.matrix.pro* is empty and *prop.pro*>0, the GRR values for protective variants will be calculated as 1/GRR of the deleterious variants. These protective and deleterious GRR values will be then assigned to the sampled protective and deleterious variants in the simulations. The size of the different groups of individuals should be a vector specified to *size*, and the user should choose whether the causal variants will be the same between the different groups of cases with the argument *same.variant*. Finally, the number of simulations is specified with the argument *replicates*. This function will return a bed matrix with the group of each individual in the field *@ped$pheno*, the first one being considered by default as the controls group, and the replicate number corresponding to the genomic region in the field *@snps$genomic.region*. The example below shows how to simulate a group of 1,000 controls and two groups of 500 cases with 50% of deleterious variants having a GRR calculated in the previous example. The deleterious variants are different between the two groups of cases and 5 genomic regions are simulated.

```
x <- random.bed.matrix.GRR(file.pop.maf = Kryukov, size = c(1000, 500, 500),
                           baseline = c(0.001, 0.001), GRR.matrix = GRR.del,
                           prop.del = 0.5, prop.pro = 0, same.variant = FALSE,
                           genetic.model = "multiplicative", select.gene = "R1",
                           replicates = 5)
x
```

```
## A bed.matrix with 2000 individuals and 1915 markers.
## snps stats are set
##   There are  1661  monomorphic SNPs
## ped stats are set
```

```
table(x@ped$pheno)
```

```
##
##    0    1    2
## 1000  500  500
```

```
table(x@snps$genomic.region)
```

```
##
##  R1  R2  R3  R4  R5
## 383 383 383 383 383
```

## Rare variant definition

To perform rare variant analysis, it is important to define what is a rare variant in order to leave out common ones. We therefore computed the function **filter.rare.variants()** which enables to keep only variants of interest based on a given MAF threshold. This function uses and returns a bed.matrix with three filters available. If the filter *"whole"* is used, all the variants with a MAF lower than the threshold in the entire sample will be kept. If the filter *"controls"* is chosen, all the variants with a MAF lower than the threshold in the control groups will be kept. Finally, if the filter *"any"* is used, all the variants with a MAF lower than the threshold in any of the groups will be kept. Monomorphic variants are also filtered out using this function. 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*.

```
#Filter of rare variants based on the MAF in any of the groups,
#only the genomic regions with at least 5 variants are kept
x.filter <- filter.rare.variants(x, filter = "any", maf.threshold = 0.01,
                                 min.nb.snps = 5)
x.filter
```

```
## A bed.matrix with 2000 individuals and 249 markers.
## snps stats are set
## ped stats are set
```

```
table(x.filter@snps$genomic.region)
```

```
##
## R1 R2 R3 R4 R5
## 50 58 45 45 51
```

## Rare variant association tests

We have implemented the generalisation of two burden tests, a type of rare variant association tests: CAST and WSS . The general principle of burden tests is to compute a genetic score per individual and and to compare it between the different groups of individuals. For these extensions, a non-ordinal multinomial regression is used in the function **score.reg.mlogit()**. This function uses a bed.matrix and depends on the package mlogit. The independant variable in this regression is the genetic effect of the gene represented by the genetic score. The scores CAST or WSS explained later can be directly calculated in this function, but the user can specify another genetic score in the function.

### Genetic scores

### CAST

The CAST score is a binary score which has a value of one if the individual carries at least one variant in the considered genomic region, 0 otherwise. A MAF threshold for the definition of a rare variant is therefore needed. This score can be computed using the function **CAST.0()**.

### WSS

The WSS (Weighted Sum Statistic) score is a continuous score giving the highest weights to the rarest variants. It can be computed using the function **WSS.0()** as follow:

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

with

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

and

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

Where $n_i$ is the total number of minor alleles genotyped for SNP $i$, $t_i$ is the total number of alleles genotyped for SNP $i$ and $I_{ij}$ is the number of minor alleles of SNP $i$ for the invidual $j$. In the original method, each SNP is weighted according to its frequency in the controls group. In our version of WSS, the weights depend on allele frequency calculated on the entire sample.

### Regressions

We have extended the two tests CAST and WSS using non-ordinal multinomial regression models. Let consider $C$ groups of individuals including a group of controls ($c = 1$) and $C - 1$ groups of cases with different sub-phenotypes of the disease. We can compute $C - 1$ probability ratios:

$$ln\frac{P(Y_j = c)}{P(Y_j = 1)} = \beta_{0,c} + \beta_{G,c}X_G + \beta_{k1,c}K_1 + ... + \beta_{kl,c}K_l$$

Where $Y_j$ corresponds to the phenotyphe of the individual $j$ and $K_l$ is a vector for the $l$th covariate with the corresponding coefficient $\beta_{kl}$. The genetic effect is represented by $X_G$ corresponding to the genetic score 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 computed using a likelihood ratio test comparing this model to the same model without the genetic effect (null hypothesis). If only two groups are compared using this package, a classical logistic regression is performed. The p-value of theses tests can be obtained using the function **score.reg.mlogit()** which can also return the odds ratio associated to the genetic score with its confidence interval (argument *get.OR.value=TRUE*) at a given alpha threshold (argument *alpha*) in each group of cases. In this function, the user needs to specify a vector with the phenotype of each individual (argument *group*) and the gene of each variant (argument *genomic.region*). This regression can also be performed on another genetic score than CAST or WSS, which has to be specified as a matrix in the argument *other.score*. This matrix should contain one individual per row and one column per genomic region. An example of the p-value and OR calculation for CAST and WSS on the data simulated previously using a non-ordinal multinomial regression on three groups is shown below.

```
##CAST
score.reg.mlogit(x=x.filter, group=x.filter@ped$pheno,
                 genomic.region=x.filter@snps$genomic.region,
                 burden.score="CAST", maf.threshold=0.01, reflevel="0",
                 alpha=0.05, get.OR.value=TRUE)
```

```
##         p.value is.err     OR.1     OR.2 l.lower.1 l.lower.2 l.upper.1
## R1 7.034788e-15      0 3.730312 7.298827  2.072242  4.255166  6.715057
## R2 9.034146e-14      0 3.996851 5.683023  2.380117  3.465091  6.711776
## R3 2.799959e-05      0 1.788910 2.750496  1.107735  1.774652  2.888957
## R4 6.853609e-06      0 2.203704 2.451311  1.460121  1.637969  3.325965
## R5 9.674315e-12      0 2.063830 4.746177  1.229556  3.031887  3.464171
##    l.upper.2
## R1 12.519578
## R2  9.320607
## R3  4.262936
## R4  3.668522
## R5  7.429763
```

```
##WSS
score.reg.mlogit(x=x.filter, group=x.filter@ped$pheno,
                 genomic.region=x.filter@snps$genomic.region,
                 burden.score="WSS", maf.threshold=0.01, reflevel="0",
                 alpha=0.05, get.OR.value=TRUE)
```

```
##         p.value is.err     OR.1      OR.2 l.lower.1 l.lower.2 l.upper.1
## R1 4.363514e-18      0 9.989477 24.540344  3.934779 10.111951 25.360926
## R2 2.179008e-13      0 7.836952 11.848759  3.543320  5.497554 17.333413
## R3 1.503636e-07      0 3.655277  6.664041  1.687091  3.254606  7.919575
## R4 9.578373e-08      0 3.762232  6.546627  1.777963  3.250273  7.961017
## R5 9.074212e-14      0 5.565266 13.323955  2.443377  6.207249 12.675974
##    l.upper.2
## R1  59.55611
## R2  25.53738
## R3  13.64511
## R4  13.18607
## R5  28.60007
```

## Power calculation

The power of the burden tests extensions can be directly calculated on simulations using the function **power.burden()**. Many arguments are needed to run this function, corresponding to the arguments from the simulations and the arguments from the regression models. As this function calculates only the power of the tests, individual p-value and OR values are not returned. The threshold used to calculate the power should be specified in the argument *alpha*. An example of this function to estimate the power at a 0.1% significance threshold of CAST and WSS with 100 replicates using the same parameters for the simulations as previously is shown below:

```
power.burden(alpha = 0.001, WSS = TRUE, CAST = TRUE,
             ,maf.threshold = 0.01, filter = "any", min.nb.snps = 5,
             pooled.analysis = TRUE, non.pooled.analysis = TRUE, analysis.by.group = TRUE,
             file.pop.maf = Kryukov, select.gene = "R1", size = c(1000,500,500),
             baseline = c(0.001,0.001), same.variant = FALSE, GRR.matrix = GRR.del,
             genetic.model = "multiplicative", prop.del = 0.5, prop.pro = 0,
             reflevel = "0", replicates = 100)
```

```
##                                power         se nb.replicates
## CAST                            1.00 0.00000000           100
## Cases1vsControls.CAST           0.57 0.04950758           100
## Cases2vsControls.CAST           1.00 0.00000000           100
## pooled.CAST                     1.00 0.00000000           100
## WSS                             1.00 0.00000000           100
## Cases1vsControls.WSS            0.67 0.04702127           100
## Cases2vsControls.WSS            1.00 0.00000000           100
## pooled.WSS                      1.00 0.00000000           100
## other.score                       NA         NA            NA
## Cases1vsControls.other.score      NA         NA            NA
## Cases2vsControls.other.score      NA         NA            NA
## pooled.other.score                NA         NA            NA
```