Package Ravages (RAre Variant Analysis and GEnetic Simulation)

Herve Perdry and Ozvan Bocher

2018-10-10

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

Introduction

Ravages was developped to simulate genetic data and to perform burden tests (a type of rare variant association tests) on more than two groups of individuals. 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. The function **set.genomic.region.by.gene()** can be used to group variant according to given gene positions. It works on a bed.matrix and simply adds a column "genomic.region" to the slot x@snps containing the gene assigned to each variant. If *include.all=FALSE*, only variants within known genes will be assigned to a genomic region, the other ones being left out. If *include.all=TRUE*, each variant will be assigned to the nearest gene. The files **genes.b37** and **genes.b38** available in Ravages which contain gene positions from ENSEMBL versions GRCh37 and GRCh38 can be used to define gene positions.

Simulation of genetic data

Genetic data for two or more groups of individuals can be simulated using the package Ravages.

Calculation of frequencies in each group of individuals

The first step to simulate genetic data is to compute genotypic frequencies in each group of individuals based on frequencies in the general population and on genetic relative risk (GRR) values. GRR correspond to the increased risk of a disease for a given genotype compared to a reference genotype, here the homozygous reference genotype. 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 reference genotype as follow:

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

With A the reference allele, and a the alternate allele. The frequency of each genotype in each group of cases c can be calculated using Bayes theorem:

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

P(AA), P(Aa) and P(aa) correspond to the genotypic probabilities in the general population. The three genotypic 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 **genotypic.freq()** performs these calculations to obtain the three genotypic frequencies in the different groups of individuals. To do so, the user needs to give *P(Y=c)*, the prevalence of each group of cases (argument *baseline*); and the GRR values. GRR values need to be in a matrix with one row per cases group and one column per variant. If there is no supposed link between the GRR associated to the heterozygous genotype and the GRR associated to the homozygous alternate genotype (general model of the disease, *genetic.model = "general"*), the user needs to specify two GRR matrices: one for GRR_{Aa} (argument *GRR*) and one for GRR_{aa} (argument *GRR.2*). If *genetic.model="recessive"*, *"multiplicative"* or *"dominant"*, only one GRR matrix is needed. **genotypic.freq()** will return a list with three matrices, one for each genotype containing the genotypic frequencies, with one row per group of individuals and one column per variant.
To help the user with the construction of the GRR matrix for the argument *GRR* of **genotypic.freq()**, we implemented the function **compute.GRR.matrix()**.
To use this function, the user needs to specify how the GRR should 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 with its own function depending on MAF in the general population (*GRR="variable"*), this function being specified to the argument *GRR.function*.

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 argument *file.pop.maf*. Two such files are available in Ravages: 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 genes, the user needs to specify which gene to choose to simulate the data with the argument *select.gene*. If this argument is empty, only the first gene will be kept in the analysis.
Finally, the multiplicative factor of the GRR between each group of cases compared to the first group of cases needs to be specified to the argument *GRR.multiplicative.factor* (number of values: number of cases groups - 1).
compute.GRR.matrix() will return a GRR matrix in the appropriate format for the function **genotypic.freq()**. Examples of these two functions are shown below:

```r
#GRR calculated using the formula from the paper presenting SKAT,
#with values in the second group of cases 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

geno.freq.groups <- genotypic.freq(file.pop.maf = Kryukov, select.gene="R1",
                                   GRR = GRR.del, baseline = c(0.001, 0.001),
                                   genetic.model = "multiplicative")
str(geno.freq.groups)
```

```
## List of 3
##  $ freq.homo.ref: num [1:3, 1:383] 1 0.999 0.998 1 1 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "controls" "cases_1" "cases_2"
##   .. ..$ : NULL
##  $ freq.het     : num [1:3, 1:383] 1.87e-04 9.56e-04 1.91e-03 5.57e-05 3.52e-04 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "controls" "cases_1" "cases_2"
##   .. ..$ : NULL
##  $ freq.homo.alt: num [1:3, 1:383] 7.90e-09 2.29e-07 9.15e-07 6.49e-10 3.11e-08 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "controls" "cases_1" "cases_2"
##   .. ..$ : NULL
```

```
geno.freq.groups$freq.homo.alt[,1:5]
```

```
##                  [,1]         [,2]         [,3]         [,4]         [,5]
## controls 7.897334e-09 6.485888e-10 7.480447e-14 6.568600e-12 2.503543e-11
## cases_1  2.288672e-07 3.106821e-08 4.778956e-11 9.067311e-10 2.469545e-09
## cases_2  9.145933e-07 1.242291e-07 1.911556e-10 3.626706e-09 9.877199e-09
```

It is also possible to calculate the MAF in each group of individuals as follow:

```
#MAF calculation for the five first variants
geno.freq.groups$freq.homo.alt[,1:5] + 0.5*geno.freq.groups$freq.het[,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 genotypic 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 **genotypic.freq()** explained previously. The arguments *file.pop.maf*, *select.gene*, *baseline* and *genetic.model* are the same as in the function **genotypic.freq()**.
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 *genetic.model="general"*, two GRR matrices need to be given as a list to the argument *GRR.matrix* (one for the heterozygous genotype and the other for the homozygous alternate genotype).
If the user wants to simulate protective variants in addition to deleterious variants, a similar argument to *GRR.matrix* should be given to *GRR.matrix.pro* 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 then be assigned

to the sampled protective and deleterious variants in the simulations, the non-causal variants having GRR values of 1.
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*. Using the argument *fixed.variant.prop*, the user needs also to choose if the argument *prop.del* (or *prop.pro*) corresponds to the final proportion of deleterious (or protective) variants, i.e. *fixed.variant.prop=TRUE* or to the probability associated to each variant of being deletrious (or protective), i.e. *fixed.variant.prop=FALSE*.
Finally, the number of genomic regions simulated is specified with the argument *replicates*.
random.bed.matrix.GRR() 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 GRR values from 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,
                           fixed.variant.prop = TRUE, replicates = 5,
                           genetic.model = "multiplicative", select.gene = "R1")
x
```

```
## A bed.matrix with 2000 individuals and 1915 markers.
## snps stats are set
##   There are  1651  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. The function **filter.rare.variants()** enables to keep only variants of interest based on a given MAF threshold. This function uses and returns a bed.matrix which can be filtered in three different ways:

- If *filter="whole"* is used, all the variants with a MAF lower than the threshold in the entire sample will be kept.
- If *filter="controls"* is chosen, all the variants with a MAF lower than the threshold in the controls group will be kept.
- If *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*.

```r
#Filter of rare variants to keep only variants with a MAF below 1% in any of the groups,
# and genomic regions with at least 5 variants
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 259 markers.
## snps stats are set
## ped stats are set
```

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

```
##
## R1 R2 R3 R4 R5
## 54 51 55 54 45
```

Rare variant association tests

We have implemented two rare variant burden association tests extensions: CAST and WSS. The general idea of burden tests is to compute a genetic score per individual and per gernomic region and and to test if it 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 gene represented by the genetic score and potential 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.

Genetic score

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, 0 otherwise. A MAF threshold for the definition of a rare variant is therefore needed. This score can be computed using the function **CAST()** as shown here on the data simulated previously:

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

```
##       R1 R2 R3 R4 R5
## A0001  0  0  0  0  0
## A0002  0  0  0  0  0
## A0003  0  0  0  0  0
## A0004  0  0  0  0  0
## A0005  0  0  0  0  0
## A0006  0  0  0  0  0
```

No statistical test is performed using this function but the user can calculate the classical chi-square associated to the CAST score as follow:

```r
#Chi-square test associated to the CAST score:
apply(CAST.score, 2, function (z) chisq.test(z)$p.value)
```

```
## Warning in chisq.test(z): Chi-squared approximation may be incorrect
## Warning in chisq.test(z): Chi-squared approximation may be incorrect

## Warning in chisq.test(z): Chi-squared approximation may be incorrect

## Warning in chisq.test(z): Chi-squared approximation may be incorrect

## Warning in chisq.test(z): Chi-squared approximation may be incorrect

##        R1        R2        R3        R4        R5
## 0.9768466 0.9880100 0.9749779 0.9662113 0.9449913
```

WSS

WSS (Weighted Sum Statistic) is based on a continuous score giving the highest weights to the rarest variants 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 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 frequency calculated on the entire sample. The function **WSS()** can be used to compute the WSS score as shown on the data simulated previously:

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

```
##       R1 R2 R3 R4 R5
## A0001  0  0  0  0  0
## A0002  0  0  0  0  0
## A0003  0  0  0  0  0
## A0004  0  0  0  0  0
## A0005  0  0  0  0  0
## A0006  0  0  0  0  0
```

Regressions

We have extended 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, one for each group of cases:

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

Where Y_j corresponds to the phenotyphe of the individual j and K_l is a vector for the lth covariate with the corresponding coefficient β_{kl}. The genetic effect is represented by X_G and correspond 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 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.mlogit()** which relies on the package mlogit. To do so, he user needs to specify a vector with the phenotype of each individual (argument *group*) and the gene of each variant (argument *genomic.region*). The CAST or WSS genetic score can directly be calculated in the regression (*burden="CAST"* or *"WSS"*). The user can also use another genetic score in the regression, which has to be specified as a matrix with one individual per row and one column per genomic region to *burden*. Potential covariates could also be included in the regression as a matrix with one row per individual and one covariate per column to the argument *data*. 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. **burden.mlogit()** 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 odds ratio associated in each group of cases with its confidence interval at a given alpha threshold (argument *alpha*) can also be obtained if *get.OR.value=TRUE*.
An example of the p-value and OR calculation with its 95% confidence interval for CAST and WSS on the data simulated previously using a non-ordinal multinomial regression on three groups is shown below.

```
##CAST
burden.mlogit(x=x.filter, group=x.filter@ped$pheno,
                 genomic.region=x.filter@snps$genomic.region,
                 burden="CAST", maf.threshold=0.01, ref.level="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 6.480024e-08      0 2.204361 3.511566  1.361829  2.256869  3.568147
## R2 8.036396e-13      0 2.011649 4.638184  1.238900  3.048718  3.266392
## R3 7.463593e-08      0 2.630435 3.435969  1.631069  2.177869  4.242119
## R4 5.713792e-11      0 1.630251 4.492782  0.942597  2.860805  2.819570
## R5 1.215756e-06      0 2.647175 3.435338  1.564906  2.080430  4.477928
##    l.upper.2
## R1  5.463807
## R2  7.056328
## R3  5.420842
## R4  7.055739
## R5  5.672648
```

```
##WSS
burden.mlogit(x=x.filter, group=x.filter@ped$pheno,
                 genomic.region=x.filter@snps$genomic.region,
                 burden="WSS", maf.threshold=0.01, ref.level="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 4.495747e-11      0 4.238710  8.655432  2.039304  4.396855  8.810195
## R2 1.126538e-11      0 3.905611  9.729603  1.766069  4.736096  8.637145
## R3 5.104523e-09      0 3.324871  6.726671  1.615672  3.483903  6.842212
## R4 1.929572e-15      0 3.492455 12.706652  1.569532  6.323535  7.771262
## R5 1.983725e-09      0 5.130469  8.474057  2.313065  3.979235 11.379583
##    l.upper.2
## R1  17.03866
## R2  19.98802
## R3  12.98776
## R4  25.53303
## R5  18.04609
```

```
##Simulation of covariates
covar <- data.frame( sex = c(sample(0:1, 1000, TRUE, c(0.2,0.8)),
                             sample(0:1, 1000, TRUE, c(0.8,0.2))),
                     u = runif(1000))

##Regression with the covariate "sex"
burden.mlogit(x=x.filter, group=x.filter@ped$pheno,
                 genomic.region=x.filter@snps$genomic.region,
                 burden="WSS", maf.threshold=0.01, ref.level="0",
                 alpha=0.05, get.OR.value=TRUE,
                 data=covar, formula = ~ sex)
```

```
##          p.value is.err     OR.1      OR.2 l.lower.1 l.lower.2 l.upper.1
## R1 5.806748e-07      0 3.298476  6.832176  1.473464  3.240738  7.383923
## R2 1.705741e-07      0 3.006242  7.567799  1.245054  3.378153  7.258715
## R3 4.726683e-07      0 3.458237  7.004377  1.544366  3.328101  7.743895
## R4 4.252773e-12      0 3.522002 12.848670  1.451028  5.830770  8.548765
## R5 1.404545e-05      0 4.232736  6.996840  1.699183  2.922245 10.543923
##    l.upper.2
## R1  14.40370
## R2  16.95352
## R3  14.74153
## R4  28.31330
## R5  16.75279
```

Power calculation

The power of burden tests extensions can be directly calculated on simulations previously explained 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. In addition, the significance threshold used to calculate the power should be specified in the argument *alpha*. As this function calculates only the power of the tests, individual p-value and OR values are not returned.
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, fixed.variant.prop=TRUE,
             GRR.matrix = GRR.del, genetic.model = "multiplicative", prop.del = 0.5,
             prop.pro = 0, ref.level = "0", replicates = 100)
```

```
##                              power         se nb.replicates
## CAST                          1.00 0.00000000           100
## Cases1vsControls.CAST         0.60 0.04898979           100
## Cases2vsControls.CAST         1.00 0.00000000           100
## pooled.CAST                   1.00 0.00000000           100
## WSS                           1.00 0.00000000           100
## Cases1vsControls.WSS          0.68 0.04664762           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
```