Package Ravages (RAre Variant Analysis and GEnetic Simulation)

Herve Perdry and Ozvan Bocher

2020-06-12

Introduction

Ravages was developped to simulate genetic data and to perform rare variant association tests (burden tests and the variance-component test SKAT) on more than two groups of individuals (Bocher et al., 2019, Genetic Epidemiology). 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. 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 available with the package Ravages. This dataset containts data from the 1000Genome project in the genomic region 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. Details about each function is given right after this example.

```
#Importation of 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
x <- set.genomic.region(x, flank.width=500)
table(x@snps$genomic.region, useNA = "ifany")
```

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

```
# Group variants within known genes using their exact positions
x <- set.genomic.region(x, flank.width=0)
table(x@snps$genomic.region, useNA = "ifany")
```

```
##
## R3HDM1  UBXN4    LCT   MCM6   DARS   <NA>
##   2038   1175   1429   1123    913   1398
```

```
# 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
##    267    167    202    159    134
```

```
# run burden test CAST, using the 1000Genome population as "outcome"
burden.mlogit(x1, group=x1@ped$pop, burden = "CAST", ref.level = "CEU")
```

```
##             p.value is.err
## R3HDM1 1.300274e-04      0
## UBXN4  4.096613e-05      0
## LCT    3.810119e-09      0
## MCM6   1.202259e-07      0
## DARS   2.036275e-03      0
```

```
# run SKAT, using the 1000Genome population as "outcome"
#Null model
x1.H0 <- SKAT.NullObject(x1@ped$pop)
SKAT(x1, x1.H0, params.sampling=list(perm.target = 100, perm.max = 5e4))
```

```
## permutations
```

```
##            stat       p.perm       p.chi2      p.value
## R3HDM1 6.438772 0.0000199996 5.993162e-06 5.993162e-06
## UBXN4  2.101316 0.0062469075 6.753168e-03 6.246907e-03
## LCT    3.483270 0.0007199856 8.819400e-04 8.819400e-04
## MCM6   2.873867 0.0078813890 6.246948e-03 7.881389e-03
## DARS   2.093018 0.0548019533 9.262642e-02 5.480195e-02
```

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 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 gene positions. It works on a bed.matrix (see Gaston) and simply adds a column "genomic.region" to the slot x@snps containing the gene assigned to each variant as a factor. Gene positions should be given to *genes* as a dataframe containing the following columns: *Chr*, *Start*, *End*, *Gene_Name*. The dataframe should absolutely be ordered in the genome order, as well as the levels of *genes$Gene_Name*. By default, any variant being outside the gene positions won't be annotated. Gene positions can be extended to annotate more variants using the argument *flank.width* corresponding to the number of base pairs upstream and downstream the gene. If *flank.width=Inf*, each variant will be assigned to the nearest gene. If two genes overlap, variants in the overlapping zone will be attributed to both genes, separated by a comma.
The files **genes.b37** and **genes.b38** available in Ravages which contain gene positions from ENSEMBL versions hg19 and hg38 can be used for *genes* to define gene positions.

Multiple genomic regions for a variant

If variants are annotated to multiple genes, they can be split in a new bed matrix using the function **bed.matrix.split.genomic.region()**. This function takes a bed matrix as argument (x) and duplicates variants being assigned to multiple genes 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. An example is shown below with an example bedmatrix and an example genes dataframe:

```
#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=letters[1:4])

#Attribute genomic regions
x.ex <- set.genomic.region(x.ex, genes = genes.ex)

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

Rare variant definition

To perform rare variant analysis, it is also 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 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*.

Rare variant association tests

We have implemented two burden tests extensions (CAST and WSS) and an extension of the variance-component test SKAT to perform the association tests between a gene 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 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. 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.

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.mlogit()** 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()**.

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 as an argument to *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. 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* having the same size 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 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 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 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, the user needs to specify a vector with the phenotype of each individual (argument *group*) and 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 individual per row and one genomic region per column to *burden*. In this situation, no bed matrix is needed, and the result from **burden.weighted.matrix()** can be used directly.
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.
Potential covariates can also be included in the regression as a matrix with one row per individual and one column per covariate 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, otherwise all the covariates will be included.
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 to each group of cases compared to the reference group (*ref.level*) 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 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.

```
# WSS
burden.mlogit(x=x1, group=x1@ped$pop, burden="WSS", ref.level="CEU", alpha=0.05,
              get.OR.value=TRUE)
```

```
##             p.value is.err   OR.TSI    OR.FIN    OR.GBR   OR.IBS l.lower.TSI
## R3HDM1 4.876886e-08      0 1.804428 1.0189455 0.8398775 1.313583    1.360068
## UBXN4  8.341120e-07      0 1.511287 0.4875926 0.5403705 1.033206    1.042657
## LCT    9.459153e-11      0 2.883109 1.4384475 0.9698497 2.344099    1.881794
## MCM6   7.339135e-10      0 3.666278 2.1369606 1.1022770 3.006620    2.149127
## DARS   6.797271e-04      0 1.677731 0.9977584 0.8258296 1.457758    1.076988
##        l.lower.FIN l.lower.GBR l.lower.IBS l.upper.TSI l.upper.FIN l.upper.GBR
## R3HDM1   0.7323019   0.5813563   0.9760668    2.393967   1.4177895   1.2133596
## UBXN4    0.2882958   0.3212035   0.6919826    2.190546   0.8246618   0.9090818
## LCT      0.8967677   0.5715946   1.5278099    4.417231   2.3073213   1.6455868
## MCM6     1.2032805   0.5687063   1.7592628    6.254444   3.7951258   2.1364537
## DARS     0.5798540   0.4551123   0.9320409    2.613568   1.7168491   1.4985194
##        l.upper.IBS
## R3HDM1    1.767809
## UBXN4     1.542691
## LCT       3.596520
## MCM6      5.138383
## DARS      2.280005
```

```
#Sex + a simulated variable as covariates
sex <- x1@ped$sex
u <- runif(nrow(x1))
covar <- cbind(sex, u)

#Regression with the covariate "sex" without OR values
burden.mlogit(x=x1, group=x1@ped$pop, burden="WSS", ref.level="CEU", data=covar, formula = ~ sex)
```

```
##             p.value is.err
## R3HDM1 4.643284e-08      0
## UBXN4  1.119938e-06      0
## LCT    9.428589e-11      0
## MCM6   6.008329e-10      0
## DARS   7.277848e-04      0
```

```
#WSS using directly the score matrix computed previously
burden.mlogit(burden=WSS.score, group=x1@ped$pop, ref.level="CEU", data=covar, formula = ~ sex)
```

```
##             p.value is.err
## R3HDM1 4.643284e-08      0
## UBXN4  1.119938e-06      0
## LCT    9.428589e-11      0
## MCM6   6.008329e-10      0
## DARS   7.277848e-04      0
```

Finally, using Ravages, it is also possible to perform burden tests with a continuous phenotype with the function **burden_continuous()** as showed below:

```
#Random continuous phenotype
set.seed(1) ; pheno1 <- rnorm(nrow(x1))
```

```
#Test CAST
burden.continuous(x1, pheno = pheno1, burden = "CAST")
```

```
##          p.value is.err
## R3HDM1 0.1233003      0
## UBXN4  0.1231611      0
## LCT    0.8944493      0
## MCM6   0.9223583      0
## DARS   0.6315334      0
```

```
#Test WSS with covariates
burden.continuous(x1, pheno = pheno1, burden = "WSS",
                  data = covar, formula = ~ sex)
```

```
##          p.value is.err
## R3HDM1 0.4213002      0
## UBXN4  0.1835285      0
## LCT    0.2631375      0
## MCM6   0.8666831      0
## DARS   0.9100517      0
```

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.
To run the SKAT association test, the function **SKAT.NullObject()** first need to be called. The group of each individual should be given as a factor to *group*, and potential covariates should be included as a data.frame to *data*. If only some of them are to be include, the 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 <- SKAT.NullObject(x1@ped$pop)
#Permutations
SKAT(x1, x1.null, get.moments = "permutations", debug = TRUE,
     params.sampling = list(perm.target = 100, perm.max =5e4))
```

```
## permutations
```

```
##            stat nb.geq nb.eq nb.perms       p.perm     mean     sigma  skewness
## R3HDM1 6.438772      1     0    50000 0.0000399992 3.477943 0.3533629 1.0488135
## UBXN4  2.101316    100     0    14958 0.0067517882 1.810003 0.1027244 0.4377892
## LCT    3.483270     23     0    50000 0.0004799904 2.652682 0.1973176 0.5603823
## MCM6   2.873867    100     0    14047 0.0071896355 2.207896 0.2017613 0.7274034
## DARS   2.093018    100     0     2064 0.0489104116 1.746181 0.2265837 1.0093902
##        kurtosis       p.chi2      p.value
## R3HDM1 5.357322 7.078508e-06 7.078508e-06
## UBXN4  3.406305 7.726638e-03 6.751788e-03
## LCT    3.674358 7.446930e-04 7.446930e-04
## MCM6   4.154554 5.685501e-03 7.189636e-03
## DARS   4.934438 7.937551e-02 4.891041e-02
```

```
#Theoretical on 1 core
SKAT(x1, x1.null, get.moments = "theoretical", debug = TRUE, cores = 1)
```

```
## theoretical
```

```
##            stat      p.value     mean     sigma  skewness kurtosis
## R3HDM1 6.438772 8.598915e-06 3.471514 0.4036203 0.8569925 4.435626
## UBXN4  2.101316 3.064362e-02 1.804261 0.1477397 0.2917564 3.181367
## LCT    3.483270 3.757820e-03 2.647597 0.2542708 0.4894252 3.545943
## MCM6   2.873867 1.646845e-02 2.205730 0.2588984 0.6761229 3.974047
## DARS   2.093018 1.236751e-01 1.744053 0.2999021 1.0108490 4.839478
```

It is also possible to perform the SKAT test on a continuous phenotype by using the function **SKAT.NullObject.continuous()** before the **SKAT()** function:

```
#Random continuous phenotype
set.seed(1) ; pheno1 <- rnorm(nrow(x1))
#Null Model
x1.H0.c <- SKAT.NullObject.continuous(pheno1)
#Run SKAT
SKAT(x1, x1.H0.c)
```

```
## continuous phenotype
```

```
##            stat    p.value
## R3HDM1 491.9476 0.23733367
## UBXN4  326.2307 0.01730003
## LCT    313.2552 0.59521320
## MCM6   303.5711 0.29075513
## DARS   176.4387 0.68310396
```

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 929 markers.
## snps stats are set
##   There are  732  monomorphic SNPs
## ped stats are set
```

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

```
## A bed.matrix with 99 individuals and 929 markers.
## snps stats are set
##   There are  755  monomorphic SNPs
## ped stats are set
```

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

```
## A bed.matrix with 91 individuals and 929 markers.
## snps stats are set
##   There are  778  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 929 markers.
## snps stats are set
##   There are  502  monomorphic SNPs
## ped stats are set
```