Package oz

Herve Perdry and Ozvan Bocher

2018-03-13

```
library("knitr")
library(oz)
```

Introduction

Oz can be used for rare-variant association tests and genetics data simulation. Oz 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 this package use bed.matrix as in the package Gaston (see documentation of this package for more details). In this vignette, we show how to simulate genetics 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 SNP into genomic regions. The first one, called by **region.by.pos()** groups the SNPs based on the distance between them and on the maximum number of groups we want to make. Indeed, the distance between each adjacent SNP pair is calculated and the maximum distance between two SNP within a genomic region increases until we have less groups than the allowed maximum number of groups. The second one, called by **region.by.gene()** uses positions of known genes to group SNP into genomic regions. If the option *include.all=FALSE* is used, only SNP within known genes will be assigned to a genomic region, the other SNP being left out. If the option *include.all=TRUE*, each SNP will be assigned to the nearest gene.

Simulation of genetic data

Genetic data can be simulated using the package oz. The procedure is similar to the one from Suzanne Leal et al used in the programm SeqX. Using functions from oz package, it is possible to compute mafs in groups of cases based on mafs in the general population and OR values. It is also possible to simulate genotype data based on mafs in each group.

Compute OR

It is possible to generate a group of controls and one or more groups of cases with different OR values. Indeed, by giving the probability for each variant of being deleterious or protective and the corresponding OR values in each group of cases, an OR matrix can be computed. Two functions can be called depending on the desired design, both returning a matrix containing one row per group of cases and one column per variant. If the function **OR.matrix.same.variant()** is called, the same variants will be deleterious or protective in the different groups of cases. If the function **OR.matrix()** is called, different variants will be deleteirous or protective in the different groups of cases. In the first example below, 10 variants are simulated, each one having a probability of 20% of being deleterious and a probability of 10% of being protective. In the first group of cases, deleterious and protective variants have respectively OR values of 2 and 0.5. In the second group of cases, deleterious and protective variants have respectively OR values of 4 and 0.25. The same variants are deleterious, neutral and protective in the two groups of cases. In the second example, 10 variants are simulated, each one having a probability of 20% of being deleterious and a probability of 10% of being protective. In the two groups of cases, deleterious and protective variants have respectively OR values of 2 and 0.5 but different variants are deleterious, neutral and protective between the two groups of cases.

```
OR.matrix.same.variant(n.variants = 10 , OR.del = c(2,4), OR.pro = c(0.5,0.25),
                       prob.del = 0.2, prob.pro = 0.1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    1    1    1 0.50 0.50    1    1    2     1
## [2,]    1    1    1    1 0.25 0.25    1    1    4     1
```

```
OR.matrix(n.variants = 10 , OR.del = c(2,2), OR.pro = c(0.5,0.5),
          prob.del = 0.2, prob.pro = 0.1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  0.5    2    1    1    1    2    1    1  0.5   0.5
## [2,]  0.5    1    2    1    1    1    2    1  1.0   2.0
```

Compute mafs

It is possible using the function **group.mafs()** to simulate with a second degree equation maf values in groups of individuals based on mafs in the general population and OR values. The output matrix will have one row per group of individuals (the first one being the control group) and one column by variant. Output from **OR.matrix()** and **OR.matrix.same.variant()** can be used as input for this function. An example is presented below using the data from Kryukov et al. available in the package oz. In this example, 10 variants in two groups of cases are simulated with the same variants being deleterious, neutral and protective. In the first group of cases, deleterious variants have OR=2 and protective variants have OR=0.5 whereas in the second one, deleterious variants have OR=4 and protective variants have OR=0.25.

```
OR <- OR.matrix.same.variant(n.variants = length(Kryukov$maf[Kryukov$unit=="R1"]) ,
                             OR.del = c(2,4), OR.pro = c(0.5,0.25),
                             prob.del = 0.2, prob.pro = 0.1)
MAF <- group.mafs(pop.maf = Kryukov$maf[Kryukov$unit=="R1"], OR = OR,
                  baseline = c(0.001,0.001))
MAF[,1:5]
```

```
##              [,1]      [,2]    [,3]     [,4]         [,5]
## controls 9.5e-05 2.833e-05 5.6e-07 3.33e-06 6.085658e-06
## cases_1  9.5e-05 2.833e-05 5.6e-07 3.33e-06 1.217124e-05
## cases_2  9.5e-05 2.833e-05 5.6e-07 3.33e-06 2.434219e-05
```

Compute genotypes

Finally, we can simulate genotypes based on maf values, group sizes, OR values and a study design using the function **random.bed.matrix()**. If the function *OR.matrix* is called as the argument *OR.function*, different variants will be deleterious and protective in the different groups of cases. However, if the function *OR.matrix.same.variant* is chosen, the same variants will be deleterious and protective in the different groups of cases. Moreover, the argument *replicates* asks for the number of replicates corresponding to the number of genomic regions that will be generated. The argument *OR.pars* needs to be a list containing at least OR values for deleterious variants in each group and the probabilities for a variant of being deleterious or protective. **random.bed.matrix()** will return a bed.matrix with the group of each individual in the field *@ped$pheno*, and the replicate number corresponding to the genomic region in the field *@snps$genomic.region*.

Output from previous simulation functions can be used as inputs for **random.bed.matrix()** as showed in the examples. By default, the first group of individual will be the group of controls.

```
#Simulation of genotypes with 10 replicates for 400 controls and two groups of 200 cases
#with the same variants being deleterious or protective but with different OR values
my.pars <- list(OR.del = c(2, 4), prob.del = 0.2, prob.pro = 0.05)
x <- random.bed.matrix(pop.maf = Kryukov$maf[Kryukov$unit=="R1"], size = c(400, 200, 200),
                       baseline = c(0.001, 0.001), replicates = 10, OR.pars = my.pars,
                       OR.function = OR.matrix.same.variant)
x
```

```
## A bed.matrix with 800 individuals and 3830 markers.
## snps stats are set
##   There are  3680  monomorphic SNPs
## ped stats are set
```

```
#Simulation of genotypes with 10 replicates for 400 controls and two groups of 200 cases
#with the different variants being deleterious or protective but with the same OR values
my.pars <- list(OR.del = c(2, 2), prob.del = 0.2, prob.pro = 0.05)
x <- random.bed.matrix(pop.maf = Kryukov$maf[Kryukov$unit=="R1"], size = c(400, 200, 200),
                       baseline = c(0.001, 0.001), replicates = 10, OR.pars = my.pars,
                       OR.function = OR.matrix)
x
```

```
## A bed.matrix with 800 individuals and 3830 markers.
## snps stats are set
##   There are  3677  monomorphic SNPs
## ped stats are set
```

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 a function **filter.rare.variants()** which enables to keep only SNP of interest based on a given maf threshold. This function uses and returns a bed.matrix with three filters are available. If the filter *"whole"* is used, all the SNPs with a maf lower than the threshold in the entire sample will be kept. If the filter *"controls"* is chosen, all the SNPs with a maf lower than the threshold in the control groups will be kept. Finally, if the filter *"any"* is used, all the SNP with a maf lower than the threshold in any of the groups will be kept. Monomorphic SNP are also filtered out using this function. All the genomic regions having less SNPs than the parameter *min.nb.snps* will also be removed.

```
my.pars <- list(OR.del=c(2, 4), prob.del=0.2, prob.pro=0.05)
#Simulation of genotypes with 100 replicates for 400 controls and two groups of 200 cases
#with the the same variants being deleterious or protective but with different OR values
x <- random.bed.matrix(pop.maf = Kryukov$maf[Kryukov$unit=="R1"], size = c(400, 200, 200),
                       baseline = c(0.001, 0.001), 100, OR.pars = my.pars,
                       OR.function = OR.matrix.same.variant)
x
```

```
## A bed.matrix with 800 individuals and 38300 markers.
## snps stats are set
##   There are  36756  monomorphic SNPs
## ped stats are set
```

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

```
## A bed.matrix with 800 individuals and 1436 markers.
## snps stats are set
## ped stats are set
```

Rare variant association tests

We have implemented the generalisation of three rare variant association tests: CAST, WSS and C.alpha. We also implemented a new test, Beta-M. All the functions use bed.matrix. All the examples will use the bed.matrix x which was simulated using previous simulation commands and Kryukov's maf in the general population. x contains a group of 400 controls and two groups of 200 cases. The same variants are deleterious, neutral and protective in the two groups of cases, with deleterious variants having OR=2 and OR=4 and protective variants having OR=0.5 and OR=0.25 in the two groups of cases respectively. 5 replicates corresponding to 5 genomic regions have been simulated under this scenario. Finally, only the variants having a maf lower than 1% in any of the three groups will be kept for the analysis.

```
my.pars <- list(OR.del = c(2, 4), prob.del = 0.2, prob.pro = 0.05)
#Simulation of genotypes with 5 replicates for 400 controls and two groups of 200 cases
#with the the the same variants being deleterious or protective but different OR values
x <- random.bed.matrix(pop.maf = Kryukov$maf[Kryukov$unit=="R1"], size = c(400, 200, 200),
                       baseline = c(0.001, 0.001), replicates = 5, OR.pars = my.pars,
                       OR.function = OR.matrix.same.variant)
#Keep only variants with MAF<1% in one of the three groups
x <- filter.rare.variants(x, filter = "any", maf.threshold = 0.01)
x
```

```
## A bed.matrix with 800 individuals and 75 markers.
## snps stats are set
## ped stats are set
```

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

```
##
## R1 R2 R3 R4 R5
## 13 19 15 19  9
```

Burden tests

CAST

The statistic for CAST which computes a binary score based on the presence or absence of at least one allele in the genomic region can be calculated with the function **CAST()**. The p-value is calculated using a Chi-square with Monte-Carlo simulations when the expected counts are lower than five and with an asymptotic p-value otherwise. When all the cases are put in a single group, we have the classical CAST test. Examples using the simulated data x are showed below.

```
#Compute the CAST score on 3 groups
CAST(x, group = x@ped$pheno, genomic.region = x@snps$genomic.region )
```

```
##    genomic.region       stat   p.value
## R1             R1 4.53190221 0.1037313
## R2             R2 2.98383755 0.2249406
## R3             R3 0.29607698 0.8623979
## R4              R4 0.05525625 0.9727500
## R5              R5 2.76923077 0.2928371
```

```
#Compute the CAST score by considering all the cases as one group
CAST(x, group = ifelse(x@ped$pheno==0, 0, 1), genomic.region = x@snps$genomic.region )
```

```
##    genomic.region     stat   p.value
## R1              R1 1.947923 0.1628109
## R2              R2 2.237878 0.1346657
## R3              R3 0.000000 1.0000000
## R4              R4 0.000000 1.0000000
## R5              R5 1.282051 0.2575180
```

WSS

The weighted Sum Statistic (WSS) and its p-value can be calculated using the function **WSS()**. The score is calculated 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

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

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 and scores between the two groups are compared with a rank test. In our version of WSS, the weights depend on allele frequency calculated on the entire sample. Therefore, a Kruskall-Wallis test is used to compare the different groups.

```
#Compute the WSS score on three groups
WSS(x, group = x@ped$pheno, genomic.region = x@snps$genomic.region )
```

```
##    genomic.region       stat     p.value
## R1              R1 4.68705613 0.095988387
## R2              R2 2.98358691 0.224968822
## R3              R3 0.30888379 0.856893280
## R4              R4 0.06150501 0.969715541
## R5              R5 9.65935397 0.007989101
```

```
#Compute the WSS score by considering all the cases as one group
WSS(x, group = ifelse(x@ped$pheno==0, 0, 1), genomic.region = x@snps$genomic.region )
```

```
##    genomic.region        stat    p.value
## R1              R1 2.604096198 0.10658791
## R2              R2 2.731321079 0.09839786
## R3              R3 0.001333928 0.97086536
## R4              R4 0.003140829 0.95530747
## R5              R5 9.131533813 0.00251241
```

Variance-component tests

C-alpha

The C-alpha statistic and its p-value can be calculated using the function **C.ALPHA()**. The score for each genomic region is calculated as follow:

$$C_\alpha = \sum_{i=1}^{R}\sum_{c=1}^{C}[(n_{ic} - n_i * \alpha_c)^2 - (n_i * \alpha_c * (1 - \alpha_c))]$$

where R is the number of SNP in the genomic region, C is the number of groups to compare, α_c represents the proportion of group c in the population, n_{ic} and n_i the number of minor alleles for the SNP i in the group c and in the entire population respectively. Permutations are then performed to compute the p-value which is calculated by the number of times the observed statistic is exceeded plus the number of times a permutated statistic is equal to the observed one divided by the total number of permutations.

```
#Compute the C-alpha score on 3 groups
C.ALPHA(x, group = x@ped$pheno, genomic.region = x@snps$genomic.region,
        which.snps = rep(TRUE, ncol(x)), target = 10, B.max = 1e6)
```

```
##   genomic.region  stat nb.geq nb.eq nb.perms     p.value
## 1             R1 -1.00     10     0       21 0.500000000
## 2             R2 -4.25     10     1       20 0.500000000
## 3             R3 -0.75     10     1       17 0.583333333
## 4             R4 27.00     10     2      792 0.012610340
## 5             R5 59.00     10     0     6143 0.001790365
```

```
#Compute the C-alpha score by considering all the cases as one group
C.ALPHA(x, group = ifelse(x@ped$pheno==0, 0, 1), genomic.region = x@snps$genomic.region,
        which.snps = rep(TRUE, ncol(x)), target = 10, B.max = 1e6)
```

```
##   genomic.region stat nb.geq nb.eq nb.perms     p.value
## 1             R1    1     10     3       26 0.351851852
## 2             R2   -1     10     1       20 0.500000000
## 3             R3    3     10     3       24 0.380000000
## 4             R4   29     10     5      408 0.020782396
## 5             R5   82     10     0    10386 0.001059016
```

Beta-M

We developed a new statistic, Beta-M, which is a variance-component test and which can be calculated using the function **Beta.M()**. In this model, we suppose that each p_{ic}, the maf of SNP i in group c, is drawn in a Beta distribution with expected value π_i and variance:$\pi_i * (1 - \pi_i) * \frac{\phi}{1-\phi}$. Where π_i represents the average frequency of the rare variant for SNP i and ϕ the dispersion parameter which is the same for all the SNP in one considered genomic region. The null hypothesis corresponding to $\phi = 0$ and $p_{ic} = \pi_i$ for each group can be tested with the score:

$$\beta_M = \sum_{i=1}^{R}[\frac{\sum_{c=1}^{C} n_{ic}^2}{\sum_{c=1}^{C} n_{ic}} + \frac{\sum_{c=1}^{C} m_{ic}^2}{\sum_{c=1}^{C} m_{ic}}]$$

With n_{ic} and m_{ic} being respectively the number of rare and frequent alleles of SNP i in group c. Permutations are then performed to compute the p-value which is calculated by the number of times the observed statistic is exceeded plus the number of times a permutated statistic is equal to the observed one divided by the total number of permutations.

```
#Compute the Beta-M score on three groups
Beta.M(x, group = x@ped$pheno, genomic.region = x@snps$genomic.region,
       which.snps = rep(TRUE, ncol(x)), target = 10, B.max = 1e6)
```

```
##   genomic.region      stat nb.geq nb.eq nb.perms     p.value
## 1             R1  7809.935     10     0      182 0.060109290
## 2             R2 11411.496     10     0       13 0.785714286
## 3              R3  9010.225     10     0       37 0.289473684
## 4              R4 11415.074     10     0      627 0.017515924
## 5              R5  5409.548     10     0     1949 0.005641026
```

```
#Compute the Beta-M score by considering all the cases as one group
Beta.M(x, group = ifelse(x@ped$pheno==0, 0, 1), genomic.region = x@snps$genomic.region,
       which.snps = rep(TRUE, ncol(x)), target = 10, B.max = 1e6)
```

```
##   genomic.region      stat nb.geq nb.eq nb.perms     p.value
## 1              R1 10407.109     10     2       43 0.227272727
## 2              R2 15210.311     10     1       37 0.276315789
## 3              R3 12008.296     10     2       53 0.185185185
## 4              R4 15211.864     10     0      168 0.065088757
## 5              R5  7209.957     10     2     8400 0.001190334
```

Jaccard similarity index

It is possible to calculate the Jaccard similarity index by using a bed.matrix and the function **Jaccard()**. The maf threshold indicating the MAF used for the definition of a rare variant should be specified with the argument *maf.threshold*. An example of this function is illustrated bellow on a simulated data set.

```
#Selection of the first genomic region of x
x1 <- select.snps(x, x@snps$genomic.region == "R1")
J <- Jaccard(x1, maf.threshold=0.01)
J[1:5,1:5] ; dim(J)
```

```
##      A001 A002 A003 A004 A005
## A001    0    0    0    0    0
## A002    0    0    0    0    0
## A003    0    0    0    0    0
## A004    0    0    0    0    0
## A005    0    0    0    0    0

## [1] 800 800
```