Gaston

Version 1.0

Hervé Perdry, Claire Dandine-Roulland

October 1, 2015

Introduction

Gaston offers functions for efficient manipulation of large genotype (SNP) matrices, and state-of-the-art implementation of algorithms to fit Linear Mixed Models, that can be used to compute heritability estimates or to perform association tests.

Thanks to the packages `Rcpp`, `RcppParallel`, `RcppEigen`, Gaston functions are mainly written in C++.

Many functions are multithreaded; the number of threads can be setted through `RcppParallel` function `setThreadOptions`. It is advised to try several values for the number of threads, as using too many threads might be conterproductive due to an important overhead.

Some functions have a `verbose` argument, which controls the function verbosity. To mute all functions at once you can use `options(gaston.verbose = FALSE)`.

We are going to illustrate this package on the data sets `AGT`, `LCT`, and `TTN` included in `gaston` (see the corresponding manual pages for a description of these data sets). Gaston also includes some example files in the `extdata` folder.

Not all options of the functions are described here, but rather their basic usage. The reader is advised to look at the manual pages for details.

Note that the package name is written `gaston` when dealing with R commands, but Gaston (with a capital) in human language.

1 Genotype matrices

An S4 class for genotype matrices is defined, named `bed.matrix`. Each row correspond to an individual, and each column to a SNP.

1.1 Reading bed.matrices from files

Bed.matrices be read from files using `read.bed.matrix`. The function `read.vcf` reads VCF files; it relies on the package `WhopGenome`.

Gaston includes example files that can be used for illustration:

```
> x <- read.bed.matrix( system.file("extdata", "LCT.bed", package="gaston") )

Reading /tmp/RtmpBY1RVF/Rinst7a59167e4ec1/gaston/extdata/LCT.rds
Reading /tmp/RtmpBY1RVF/Rinst7a59167e4ec1/gaston/extdata/LCT.bed

> x

A bed.matrix with 503 individuals and 607 markers.
```

The folder `extdata/` contains files `LCT.bed`, `LCT.rds`, `LCT.bim` and `LCT.fam`. The `.bed`, `.bim` and `.fam` files follow the PLINK specifications. The `.rds` file is a R data file; if it is present, the `.bim` and `.fam` files are ignored. You can ignore the `.rds` file using option `rds = NULL`:

```
> x <- read.bed.matrix( system.file("extdata", "LCT.bed", package="gaston"), rds = NULL )

Reading /tmp/RtmpBY1RVF/Rinst7a59167e4ec1/gaston/extdata/LCT.fam
Reading /tmp/RtmpBY1RVF/Rinst7a59167e4ec1/gaston/extdata/LCT.bim
Reading /tmp/RtmpBY1RVF/Rinst7a59167e4ec1/gaston/extdata/LCT.bed

> x

A bed.matrix with 503 individuals and 607 markers.
```

A bed.matrix can be saved using `write.bed.matrix`. The default behavior is to write `.bed`, `.bim`, `.fam` and `.rds` files; see the manual page for more details.

1.2 Conversion from and to R objects

A numerical matrix `x` containing genotype counts (0, 1, 2 or `NA`) can be transformed in a bed.matrix with `as(x, "bed.matrix")`. The resulting object will lack individual and SNP informations (if the rownames and colnames of `x` are set, they will be used as SNP and individual ids respectively).

Conversely, a numerical matrix can be retrieved from a bed.matrix using `as.matrix`.

The function `as.bed.matrix` allows to provide data frames corresponding to the `.fam` and `.bim` files. They should have colnames `famid`, `id`, `father`, `mother`, `sex`, `pheno`, and `chr`, `id`, `dist`, `pos`, `A1`, `A2` respectively. This function is widely used in the examples included in manual pages.

```
> data(TTN)
> x <- as.bed.matrix(TTN.gen, TTN.fam, TTN.bim)
> x

A bed.matrix with 503 individuals and 733 markers.
```

1.3 The insides of a bed.matrix

In first approach, a bed.matrix behaves as a "read-only" matrix containing only 0, 1, 2 and NAs, unless the genotypes are standardized (use `standardize<-`). They are stored in a compact form, each genotype being coded on 2 bits (hence 4 genotypes per byte).

Bed.matrices are implemented using S4 classes and methods. Let's have a look on the slots names of the bed.matrix `x` created above using the dataset `LCT`.

```
> slotNames(x)

[1] "ped"                 "snps"                "bed"
[4] "p"                   "mu"                  "sigma"
[7] "standardize_p"       "standardize_mu_sigma"
```

The slot `x@bed` is an external pointer, which indicates where the genetic data are stored in memory. It will be used by the C++ functions called by Gaston.

```
> x@bed

<pointer: 0x2cee640>
```

Let's look at the contents of the slots `x@ped` and `x@snps`. The other slots will be commented later.

The slot `x@ped` gives informations on the individuals. It corresponds to the contents of a `.fam` file, or to the first 6 columns of a `.ped` file (known as linkage format).

```
> dim(x@ped)

[1] 503   6

> head(x@ped)

    famid      id father mother sex pheno
1 HG00096 HG00096      0      0   0    NA
2 HG00097 HG00097      0      0   0    NA
3 HG00099 HG00099      0      0   0    NA
4 HG00100 HG00100      0      0   0    NA
5 HG00101 HG00101      0      0   0    NA
6 HG00102 HG00102      0      0   0    NA
```

The slot `x@snps` gives informations on the SNPs. It corresponds to the contents of a `.bim` file.

```
> dim(x@snps)

[1] 733   6
```

```
> head(x@snps)
```

```
  chr         id dist       pos A1 A2
1   2  rs7571247    0 179200322  C  T
2   2  rs3813253    0 179200714  G  A
3   2  rs6760059    0 179200947  T  C
4   2 rs16866263    0 179201048  T  G
5   2 rs77946091    0 179201380  A  G
6   2 rs77711640    0 179201557  A  G
```

1.4 Adding basic statistics to a bed.matrix

Some basic descriptive statistics can be added to a bed.matrix with `set.stats`. By default this function adds columns to the `ped` and `snps` slots:

```
> x <- set.stats(x)
```

```
ped stats and snps stats have been set.
'p' has been set.
'mu' and 'sigma' have been set.
```

```
> head(x@ped)
```

```
    famid      id father mother sex pheno  N0  N1  N2 NAs  callrate
1 HG00096 HG00096      0      0   0    NA 128  82 523   0 1.0000000
2 HG00097 HG00097      0      0   0    NA 109  81 543   0 1.0000000
3 HG00099 HG00099      0      0   0    NA  75 154 503   1 0.9986357
4 HG00100 HG00100      0      0   0    NA 148  86 499   0 1.0000000
5 HG00101 HG00101      0      0   0    NA  18 394 320   1 0.9986357
6 HG00102 HG00102      0      0   0    NA  50 180 503   0 1.0000000
```

```
> head(x@snps)
```

```
  chr         id dist       pos A1 A2 N0  N1  N2 NAs callrate        maf        hz
1   2  rs7571247    0 179200322  C  T  5  88 410   0        1 0.09741551 0.1749503
2   2  rs3813253    0 179200714  G  A 24 187 292   0        1 0.23359841 0.3717694
3   2  rs6760059    0 179200947  T  C 11 139 353   0        1 0.16003976 0.2763419
4   2 rs16866263    0 179201048  T  G  2  53 448   0        1 0.05666004 0.1053678
5   2 rs77946091    0 179201380  A  G  2  53 448   0        1 0.05666004 0.1053678
6   2 rs77711640    0 179201557  A  G  2  54 447   0        1 0.05765408 0.1073559
```

The columns `N0`, `N1`, `N2` and `NAs` contain the number of genotypes 0, 1, 2 and `NA`, for each invidual or each SNP according to the data frame in which they appear. The callrate is the proportion of non-missing genotypes. In the `snps` slot, `maf` is the minor allele frequency, and `hz` it the proportion of heterozygotes.

The default is too also also update the slots `x@p`, `x@mu` and `x@sigma`:

```
> str(x@p)
```

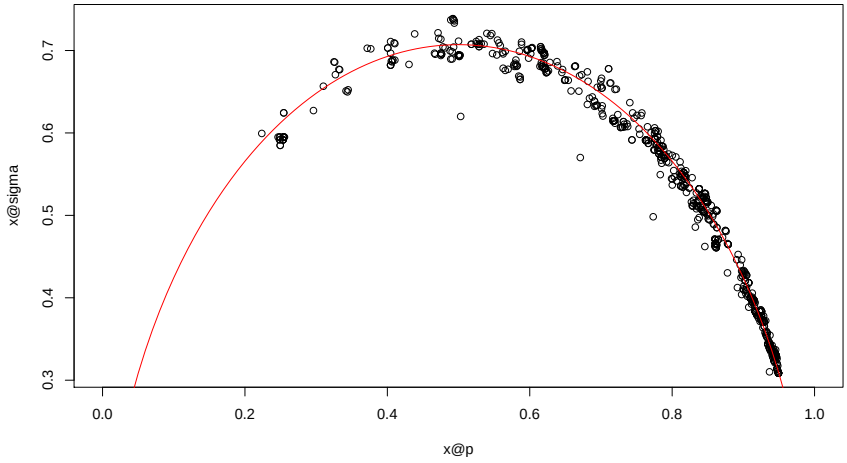
```
 num [1:733] 0.903 0.766 0.84 0.943 0.943 ...
```

```
> str(x@mu)
```

```
 num [1:733] 1.81 1.53 1.68 1.89 1.89 ...
> str(x@sigma)
 num [1:733] 0.421 0.587 0.512 0.33 0.33 ...
```

`p` contains the alternate allele frequency; `mu` is equal to `2*p` and is the expected value of the genotype (coded in 0, 1, 2); and `sigma` is the genotype standard error. If the Hardy-Weinberg equilibrium holds, `sigma` should be close to `sqrt(2*p(1-p))`. This is illustrated on the figure below.

```
> plot(x@p, x@sigma, xlim=c(0,1))
> t <- seq(0,1,length=101);
> lines(t, sqrt(2*t*(1-t)), col="red")
```



There are options to `set.stats` which alters its behavior, allowing to update only some slots.

Hardy-Weinberg Equilibrium can be tested for all SNPs simply by `x <- set.hwe(x)`. This command adds to the data frame `x@snps` a column `hwe`, containing the p-value of the test (both χ^2 and "exact" tests are available, see the man page).

Note that when writting a bed.matrix `x` to disk with `write.bed.matrix`, the `.rds` file contains all the slots of `x`, and thus contains the additionnal variables added by `set.stats` or `set.hwe`, which are note saved in the `.bim` and `.fam` files.

1.5 Subsetting bed.matrices

It is possible to subset bed.matrices just as base matrices, writing e.g. `x[1:100,]` to extract the first 100 individuals, or `x[1:100,10:19]` to extract the SNPs 10 to 19 for these 100 individuals:

```
> x[1:100,]
```

```
A bed.matrix with 100 individuals and 733 markers.
> x[1:100,10:19]
A bed.matrix with 100 individuals and 10 markers.
```

The use of logical vectors for subsetting is allowed too. The following code extracts the SNPs with minor allele frequency > 0.1:

```
> x[,x@snps$maf > 0.1]
A bed.matrix with 503 individuals and 501 markers.
```

For this kind of selection, Gaston provides the functions `select.inds` and `select.snps`, which have a nicer syntax. Hereafter we use the same condition on the minor allele frequency, and we introduce also a condition on the Hardy-Weinberg Equilibrium p-value:

```
> x <- set.hwe(x)
Computing basic HW chi-square p-values
> select.snps(x, maf > 0.1 & hwe > 1e-3)
A bed.matrix with 503 individuals and 497 markers.
```

The conditions in `select.snps` can use any of the variables defined in the data frame `x@snps`, as well as variables defined in the user session.

The function `select.inds` is similar, using variables of the data frame `x@ped`. One can for example select individuals with callrate greater than 95% with `select.inds(x, callrate > 0.5)`.

These functions should make basic Quality Control easy.

Note: Subsetting will erase some or all statistics added by `set.stats`, except the `p`, `mu` and `sigma` slots. A solution is to write `set.stats( x[1:100,10:19] )`.

2 Standardized matrices

Gaston allows to "standardize" a genotype matrix, replacing each genotype X_{ij} (i is the individual index, j is the SNP index) by

$$\frac{X_{ij} - \mu_j}{\sigma_j} \tag{1}$$

where $\mu_j = 2p_j$ is the mean of the 0,1,2-coded genotype (p_j being the alternate allele frequency), and σ_j is either its standard error, or the expected standard error under Hardy-Weinberg Equilibrium, that is $\sqrt{2p_j(1-p_j)}$.

Consider the TTN data set. The unscaled matrix contains 0,1,2 values:

```
> x <- set.stats(as.bed.matrix(TTN.gen, TTN.fam, TTN.bim))
```

```
ped stats and snps stats have been set.
'p' has been set.
'mu' and 'sigma' have been set.
```

```
> X <- as.matrix(x)
> X[1:5,1:4]
```

```
        rs7571247 rs3813253 rs6760059 rs16866263
HG00096         2         2         2          2
HG00097         2         2         2          2
HG00099         1         2         2          2
HG00100         2         0         0          2
HG00101         2         2         2          2
```

To scale it using the standard error, use `standardize(x) <- "mu_sigma"` (or `standardize(x) <- "mu"`, as the function uses `match.arg`):

```
> standardize(x) <- "mu"
> as.matrix( x[1:5, 1:4] )
```

```
         rs7571247  rs3813253  rs6760059 rs16866263
HG00096  0.4629595  0.7953656  0.6254622  0.3437939
HG00097  0.4629595  0.7953656  0.6254622  0.3437939
HG00099 -1.9132512  0.7953656  0.6254622  0.3437939
HG00100  0.4629595 -2.6094762 -3.2827052  0.3437939
HG00101  0.4629595  0.7953656  0.6254622  0.3437939
```

The result is similar to what would be obtained from the base function `scale`:

```
> scale(X)[1:5,1:4]
```

```
         rs7571247  rs3813253  rs6760059 rs16866263
HG00096  0.4629595  0.7953656  0.6254622  0.3437939
HG00097  0.4629595  0.7953656  0.6254622  0.3437939
HG00099 -1.9132512  0.7953656  0.6254622  0.3437939
HG00100  0.4629595 -2.6094762 -3.2827052  0.3437939
HG00101  0.4629595  0.7953656  0.6254622  0.3437939
```

To use $\sqrt{2p_j(1-p_j)}$ instead of the standard error, use `standardize(x) <- "p"`; a similar result can again be obtained from `scale`, with the right options:

```
> standardize(x) <- "p"
> as.matrix( x[1:5, 1:4] )
         rs7571247  rs3813253  rs6760059 rs16866263
HG00096  0.4646063  0.7807675  0.6173047  0.3465926
HG00097  0.4646063  0.7807675  0.6173047  0.3465926
HG00099 -1.9200567  0.7807675  0.6173047  0.3465926
HG00100  0.4646063 -2.5615820 -3.2398911  0.3465926
HG00101  0.4646063  0.7807675  0.6173047  0.3465926
> scale(X, scale = sqrt(2*x@p*(1-x@p)))[1:5,1:4]
         rs7571247  rs3813253  rs6760059 rs16866263
HG00096  0.4646063  0.7807675  0.6173047  0.3465926
HG00097  0.4646063  0.7807675  0.6173047  0.3465926
HG00099 -1.9200567  0.7807675  0.6173047  0.3465926
HG00100  0.4646063 -2.5615820 -3.2398911  0.3465926
HG00101  0.4646063  0.7807675  0.6173047  0.3465926
```

To go back to the 0,1,2-coded genotypes, use `standardize(x) <- "none"`.

Note: In standardized matrices, the `NA` values are replaced by zeroes, which amount to impute the missing genotypes by the mean genotype.

2.1 Matrix multiplication

Standardized matrices can be multiplied with numeric vectors or matrices with the operator `%*%`.

This feature can be used e.g. to simulate quantitative phenotypes with a genetic component. The following code simulates a phenotype with an effect of the SNP #350:

```
> y <- x %*% c(rep(0,350),0.25,rep(0,ncol(x)-351)) + rnorm(nrow(x), sd = 1)
```

2.2 Genetic Relationship Matrix and Principal Component Analysis

If X_s is a standardized $n \times q$ genotype matrix, a Genetic Relationship Matrix (GRM) of the individuals can be computed as

$$GRM = \frac{1}{q-1} X_s X_s'$$

where q is the number of SNPs. This computation is done by the function `GRM`.

The Principal Component Analysis (PCA) of large genomic data sets is used to retrieve population stratification. It can be obtained from the eigen decomposition of the GRM. To illustrate this, we included in the `extdata` folder a dataset extracted from the 1000 genomes project. This data set comprehend the 503 individuals of european ascent, with 10025 SNPs on the chromosome 2. These SNPs have been selected with

the function `LD.thin` so that they have very low LD with each other ($r^2 < 0.05$). We added in the data frame `x@ped` a variable `population` which is a factor with levels `CEU`, `FIN`, `GBR`, `IBS` and `TSI`.

We can load this data set as follows:

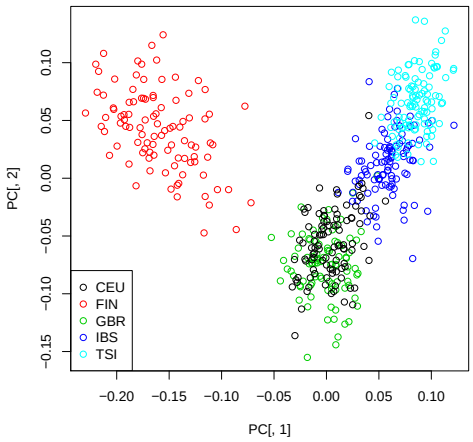
```
> x <- read.bed.matrix( system.file("extdata", "chr2.bed", package="gaston") )
Reading /tmp/RtmpBY1RVF/Rinst7a59167e4ec1/gaston/extdata/chr2.rds
Reading /tmp/RtmpBY1RVF/Rinst7a59167e4ec1/gaston/extdata/chr2.bed
> x
A bed.matrix with 503 individuals and 10025 markers.
> head(x@ped)
    famid      id father mother sex pheno population
1 HG00096 HG00096      0      0   0    -9        GBR
2 HG00097 HG00097      0      0   0    -9        GBR
3 HG00099 HG00099      0      0   0    -9        GBR
4 HG00100 HG00100      0      0   0    -9        GBR
5 HG00101 HG00101      0      0   0    -9        GBR
6 HG00102 HG00102      0      0   0    -9        GBR
> table(x@ped$population)
CEU FIN GBR IBS TSI
 99  99  91 107 107
```

We compute the Genetic Relationship Matrix, and its eigen decomposition :

```
> standardize(x) <- 'p'
> K <- GRM(x)
> eiK <- eigen(K)
> # deal with a small negative eigen value
> eiK$values[ eiK$values < 0 ] <- 0
```

The eigenvectors are normalized. The Principal Components (PC) can be computed by multiplying them by the square root of the associated eigenvalues. Here we plot the projection of the 503 individuals on the first two PCs, with colors corresponding to their population.

```
> PC <- sweep(eiK$vectors, 2, sqrt(eiK$values), "*")
> plot(PC[,1], PC[,2], col=x@ped$population)
> legend("bottomleft", pch = 1, legend = levels(x@ped$population), col = 1:5)
```



As K can be written

$$K = \left(\frac{1}{\sqrt{q-1}} X_s\right)\left(\frac{1}{\sqrt{q-1}} X_s\right)',$$

the PCs are the left singular vectors of $\frac{1}{\sqrt{q-1}} X_s$. The vectors of loadings are the right singlar vectors of this matrix. The vector of loadings corresponding to a PC u is the vector v with unit norm, such that $u = \frac{1}{\sqrt{q-1}} X_s v$.

They can be retrieved with the function `bed.loadings`:

```
> # one can use PC[,1:2] instead of eiK$vectors[,1:2] as well
> L <- bed.loadings(x, eiK$vectors[,1:2])
> dim(L)
```

```
[1] 10025     2
```

```
> head(L)
```

```
                    [,1]          [,2]
rs113106463 -0.0059549229  0.0064476377
rs13390778  -0.0099678231 -0.0125721018
rs75011129  -0.0001122798 -0.0001181864
rs4637157   -0.0151318030 -0.0148076831
rs62116661  -0.0002977572  0.0048268335
rs10170011   0.0185198427 -0.0019553369
```

We verify that the loadings have unit norm:

```
> colSums(L**2)
```

```
[1] 1 1
```

And that the PCs are retrieved by right multiplying X_s by the loadings:

```
> head( (x %*% L) / sqrt(ncol(x)-1) )
                [,1]        [,2]
HG00096  0.021410876 -0.12418402
HG00097 -0.014005024 -0.07378043
HG00099  0.001384440 -0.09708209
HG00100  0.020319167 -0.08920825
HG00101  0.002596089 -0.08925961
HG00102  0.010155044 -0.08968426
> head( PC[,1:2] )
             [,1]        [,2]
[1,]  0.021410876 -0.12418402
[2,] -0.014005024 -0.07378043
[3,]  0.001384440 -0.09708209
[4,]  0.020319167 -0.08920825
[5,]  0.002596089 -0.08925961
[6,]  0.010155044 -0.08968426
```

2.3 Linkage Disequilibrium

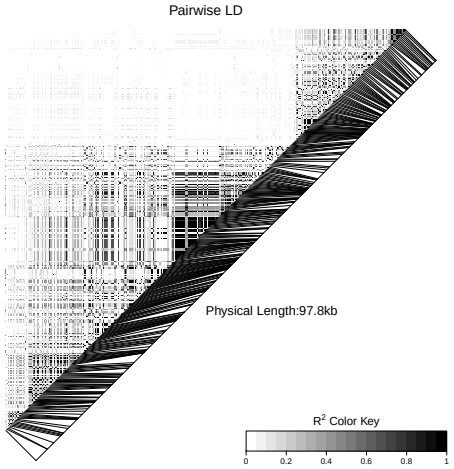
Doing the crossproduct in the reverse order produces a moment estimate of the Linkage Disequilibrium (LD):

$$LD = \frac{1}{n-1} X_s' X_s$$

where n is the number of individuals. This computation is done by the function `LD` (usually, only parts of the whole LD matrix is computed). The fonction `LD` can compute a square symmetric matrix giving the LD for a given region, measured by r^2 (the default), r or D. It can also compute the LD between two different regions.

We suggest to use `LDheatmap` (from the package of the same name) to plot square symmetric matrices of LD. Gaston may offer alternatives in future releases.

```
> data(AGT)
> x <- as.bed.matrix(AGT.gen, AGT.fam, AGT.bim)
> x <- set.stats(x)
ped stats and snps stats have been set.
'p' has been set.
'mu' and 'sigma' have been set.
> ld.x <- LD(x, c(1,ncol(x)))
> colormap <- grey.colors(20,0,1,1)
> LDheatmap(ld.x, genetic.distances=x@snps$pos, color = colormap)
```



This method is also used by `LD.thin` to extract a set of SNPs in low linkage disequilibrium (it is often recommended to perform this operation before computing the GRM). We illustrate this function on the AGT data set:

```
> y <- LD.thin(x, threshold = 0.4, max.dist = 500e3)
> y
```

```
A bed.matrix with 503 individuals and 48 markers.
```

The argument `max.dist = 500e3` is to specify that the LD won't be computed for SNPs more than 500 kb appart. We verify that there is no SNP pair with LD $r^2 > 0.4$ (note that the LD matrix has ones on the diagonal):

```
> ld.y <- LD( y, lim = c(1, ncol(y)) )
> sum( ld.y > 0.4 )
```

```
[1] 48
```

3 Linear Mixed Models

Linear Mixed Models are usually written under the form

$$Y = X\beta + Z_1 u_1 + \cdots + Z_k u_k + \varepsilon$$

where $Y \in \mathbb{R}^n$ is the response vector, and $X \in \mathbb{R}^{n\times p}, Z_1 \in \mathbb{R}^{n\times q_1}, \ldots, Z_k \in \mathbb{R}^{n\times q_k}$ are design matrices. The vector $\beta \in \mathbb{R}^p$ is the vector of fixed effects; the random effects are drawn in normal distributions, $u_1 \sim \mathcal{N}(0, \tau_1 I_{q_1}), \ldots, u_k \sim \mathcal{N}(0, \tau_k I_{q_k})$, as the residual error $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_n)$.

Here we will use the equivalent form

$$Y = X\beta + \omega_1 + \ldots + \omega_k + \varepsilon$$

where $K_i = Z_i Z_i'$ $(i = 1, \ldots, k)$, the random terms are $\omega_i \sim N(0, \tau_i K_i)$ for $i \in 1, \ldots, k$ and $\varepsilon \sim N(0, \sigma^2 I_n)$.

Note that using the R function `model.matrix` can help you to rewrite a model under this form.

Gaston provides two functions for estimating the model parameters when the model is written in the second form. We are going to illustrate these functions on simulated data.

3.1 Data simulation

We will use the above notations. First generate some (random) design matrices:

```
> set.seed(1)
> n <- 100
> q1 <- 20
> Z1 <- matrix( rnorm(n*q1), nrow = n )
> X <- cbind(1, 5*runif(n))
```

Then the vector of random effects u_1 and a vector y under the model $Y = X(1,2)' + Zu_1 + \varepsilon$ with $u_1 \sim \mathcal{N}(0, 2I_{q_1})$ and $\varepsilon \sim \mathcal{N}(0, 3I_n)$:

```
> u1 <- rnorm(q1, sd = sqrt(2))
> y <- X %*% c(1,2) + Z1 %*% u1 + rnorm(n, sd = sqrt(3))
```

To illustrate the case where there are several random effects vectors, we generate an other matrix Z_2, the corresponding vector of random effects u_2, and a vector y_2 under the model $Y = X(1,2)' + Zu_1 + Z_2 u_2 + \varepsilon$.

```
> q2 <- 10
> Z2 <- matrix( rnorm(n*q2), nrow = n )
> u2 <- rnorm(q2, sd = 1)
> y2 <- X %*% c(1,2) + Z1 %*% u1 + Z2 %*% u2 + rnorm(n, sd = sqrt(3))
```

3.2 Model fitting with the AIREML algorithm

`lmm.aireml` is a function for linear mixed models parameter estimation and BLUP computations.

3.2.1 One random effects vector

Let's start with the simple case (only one random effects vector). As `lmm.aireml` uses the second form of the linear mixed model, and we give it the matrix $K_1 = Z_1 Z_1'$.

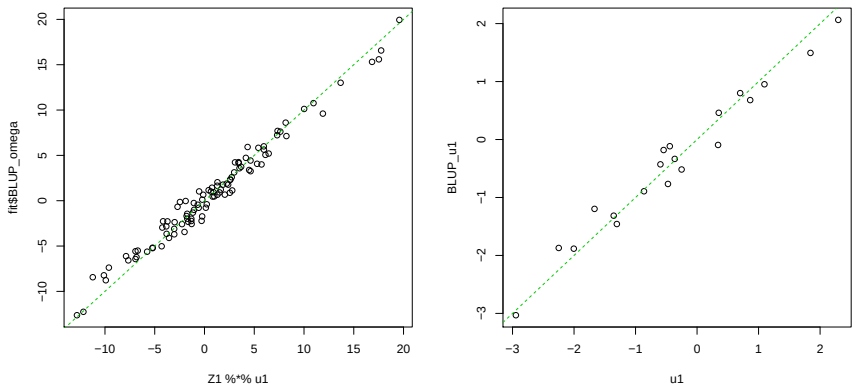
```
> K1 <- tcrossprod(Z1)
> fit <- lmm.aireml(y, X, K = K1, verbose = FALSE)
> str(fit)
```

```
List of 11
 $ sigma2    : num 3.23
 $ tau       : num 1.65
 $ logL      : num -150
 $ logL0     : num -231
 $ niter     : int 29
 $ norm_grad : num 1.15e-07
 $ Py        : num [1:100] 0.531 -0.243 0.116 0.34 -0.304 ...
 $ BLUP_omega: num [1:100] -0.443 -12.628 -1.73 5.059 5.836 ...
 $ BLUP_beta : num [1:2] 1.4 1.79
 $ varbeta   : num [1:2, 1:2] 0.134 -0.0387 -0.0387 0.016
 $ varXbeta  : num 7.6
```

The result is a list giving all kind of information. Here we see that τ is estimated to 1.65 and σ^2 to 3.23. The Best Linear Unbiased Predictor (BLUP) for β is in the component `BLUP_beta` and here it is $(1.4, 1.79)$.

The component `BLUP_omega` holds the BLUP for $\omega_1 = Zu_1$. The BLUP for u_1 can be retrieved by the formula $\widehat{u_1} = \tau_1 Z_1' Py$, the value of Py being in the component `BLUP_Py`. The plots below compare the true values of ω_1 and u_1 with their BLUPs.

```
> par(mfrow = c(1, 2))
> plot(Z1 %*% u1, fit$BLUP_omega); abline(0, 1, lty = 2, col = 3)
> BLUP_u1 <- fit$tau * t(Z1) %*% fit$Py
> plot(u1, BLUP_u1); abline(0, 1, lty = 2, col = 3)
```



3.2.2 Several random effects vector

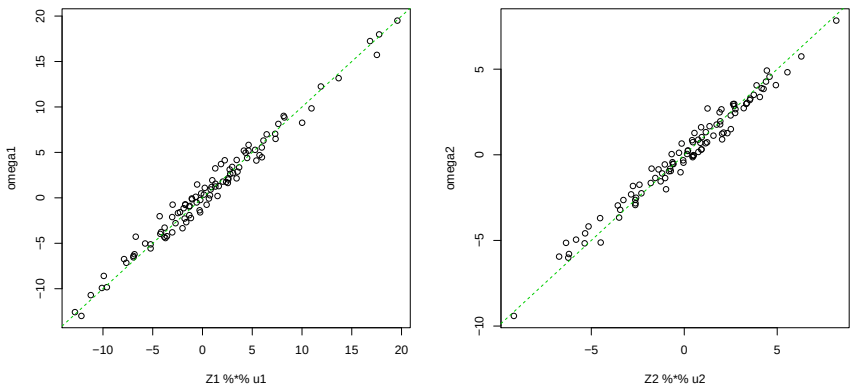
It is sufficient to give to `lmm.aireml` a list with the two matrices K_1 and K_2:

```
> K2 <- tcrossprod(Z2)
> fit2 <- lmm.aireml(y2, X, K = list(K1, K2), verbose = FALSE)
> str(fit2)
```

```
List of 10
 $ sigma2    : num 3.04
 $ tau       : num [1:2] 1.81 0.821
 $ logL      : num -164
 $ logL0     : num -244
 $ niter     : int 32
 $ norm_grad : num 6.86e-06
 $ Py        : num [1:100] -0.353 -0.199 0.879 0.472 0.968 ...
 $ BLUP_omega: num [1:100] 0.342 -15.406 6.239 4.559 4.361 ...
 $ BLUP_beta : num [1:2] 1.1 1.98
 $ varXbeta  : num 9.33
```

The component `tau` now holds the two values τ_1 and τ_2. Note that there is only one `BLUP_omega` vector. It corresponds to the BLUP for $\omega_1 + \omega_2$. The BLUPs for each ω_i can be retrieved using $\widehat{\omega_i} = \tau_i K_i P y$:

```
> par(mfrow = c(1, 2))
> omega1 <- fit2$tau[1] * K1 %*% fit2$Py
> omega2 <- fit2$tau[2] * K2 %*% fit2$Py
> plot(Z1 %*% u1, omega1); abline(0, 1, lty = 2, col = 3)
> plot(Z2 %*% u2, omega2); abline(0, 1, lty = 2, col = 3)
```



The BLUPs for u_1 and u_2 can as above be retrieved using $\widehat{u_i} = \tau_i Z_i' P y$.

3.3 Model fitting with the diagonalization trick

In the case where there is only one vector of random effects, the diagonalization trick uses the eigen decomposition of K_1 to speed up the computation of the model restricted likelihood, which allows to use a generic algorithm to maximize it. Of course, the eigen decomposition of K_1 needs to be computed beforehand, which induces a small overhead. We fit the first model again:

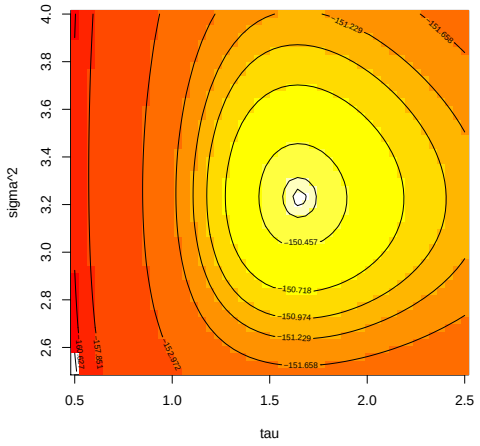
```
> eiK1 <- eigen(K1)
> fit.d <- lmm.diago(y, X, eiK1)
> str(fit.d)
```

```
List of 9
 $ sigma2    : num 3.23
 $ tau       : num 1.65
 $ Py        : num [1:100] 0.531 -0.243 0.116 0.34 -0.304 ...
 $ BLUP_omega: num [1:100] -0.443 -12.627 -1.729 5.059 5.835 ...
 $ BLUP_beta : num [1:2] 1.4 1.79
 $ varbeta   : num [1:2, 1:2] 0.134 -0.0387 -0.0387 0.016
 $ Xbeta     : num [1:100] 3.09 5.93 1.65 5.85 9.89 ...
 $ varXbeta  : num 7.6
 $ p         : int 0
```

You can check that the result is similar to the result obtained earlier with `lmm.aireml`.

The likelihood computation with the diagonalization trick is fast enough to plot the likelihood:

```
> TAU <- seq(0.5,2.5,length=50)
> S2 <- seq(2.5,4,length=50)
> lik <- lmm.diago.likelihood(tau = TAU, s2 = S2, Y = y, X = X, eigenK = eiK1)
> lik.contour(TAU, S2, lik, heat = TRUE, xlab = "tau", ylab = "sigma^2")
```



3.4 Genomic heritability estimation with Gaston

Heritability estimates based on Genetic Relationship Matrices (GRM) are computed under the following model:

$$Y = \beta + \omega + \varepsilon$$

where $\omega \sim \mathcal{N}(0, \tau K)$ and $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_n)$, K being a GRM computed on whole genome data (e.g. by the function `GRM`). The heritability is $h^2 = \frac{\tau}{\tau+\sigma^2}$.

Note that as $K = \frac{1}{q-1} X_s X_s'$ with X_s a standardized genotype matrix, letting $Z = (q-1)^{-\frac{1}{2}} X_s$, this model is equivalent to

$$Y = \beta + Zu + \varepsilon$$

with $u \sim \mathcal{N}(0, \tau I_q)$.

The function `random.pm` generates random positive matrices with eigenvalues roughly similar to those typically observed when using whole genome data. It outputs a list with a member `K`: a positive matrix, and a member `eigen`: its eigen decomposition (as the base function `eigen` would output it).

The function `lmm.simu` can be used simulate data under the linear mixed model. It uses the eigen decomposition of K. Hereafter we use $\tau = 1$, $\sigma^2 = 2$, hence a 33.3% heritability.

```
> set.seed(1)
> n <- 2000
> R <- random.pm(n)
> y <- 2 + lmm.simu(tau = 1, sigma2 = 2, eigenK = R$eigen)$y
```

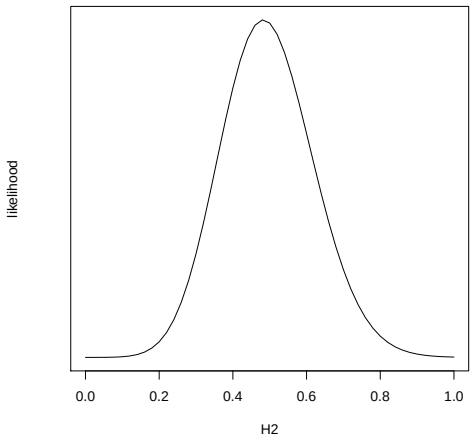
We can use `lmm.diago` to estimates the parameters of the model.

```
> fit <- lmm.diago(y, eigenK = R$eigen)
```

We have $\widehat{\tau} = 1.439$ and $\widehat{\sigma^2} = 1.54$, hence h^2 is estimated to 48.2%.

The function `lmm.diago.likelihood` allows to compute a profile likelihood with parameter $h^2 = \frac{\tau}{\tau+\sigma^2}$. It can be useful to get confidence intervals. Here, we simply plot it:

```
> H2 <- seq(0,1,length=51)
> lik <- lmm.diago.likelihood(h2 = H2, Y = y, eigenK = R$eigen)
> plot(H2, exp(lik$likelihood-max(lik$likelihood)), type="l", yaxt="n", ylab="likelihood")
```



It is often advised to include the first few (10 or 20) Principal Components (PC) in the model as fixed effects, to account for a possible population stratification. The function `lmm.diago` has an argument `p` for the number of PCs to incoporate in the model with fixed effects. We simulate a trait with a large effect of the two first PCs:

```
> PC <- sweep(R$eigen$vectors, 2, sqrt(R$eigen$values), "*")
> y1 <- 2 + PC[,1:2] %*% c(5,5) + lmm.simu(tau = 1, sigma2 = 2, eigenK = R$eigen)$y
```

Here are the heritability estimates with $p = 0$ (the default) and $p = 10$.

```
> fit0 <- lmm.diago(y1, eigenK = R$eigen)
> fit0$tau/(fit0$tau+fit0$sigma2)
[1] 0.6156099
> fit10 <- lmm.diago(y1, eigenK = R$eigen, p = 10)
> fit10$tau/(fit10$tau+fit10$sigma2)
[1] 0.3145036
```

As expected, the estimate is inflated when no PCs are incorporated in the model.

4 Association test

The function `association.test` performs a test of association between SNPs and a quantitative trait Y under the following model

$$Y = (X|PC)\alpha + G\beta + \omega + \varepsilon$$

where X is the matrix of covariables with fixed effects (including a column of ones for the intercept), G is the vector of genotypes at the SNP under consideration, and $\omega \sim \mathcal{N}(0, \tau K)$ where K is a Genetic Relationship Matrix (computed on the whole genome). A few PCs can be included in the model with fixed effects (using parameter `p` as above for `lmm.diago`).

Two tests are proposed for $H_0 : \beta = 0$: a Wald test on $\widehat{\beta}$, or a Likelihood Ratio Test.

We illustrate this function on a simple example, build on the AGT data set:

```
> data(AGT)
> x <- set.stats(as.bed.matrix(AGT.gen, AGT.fam, AGT.bim))

ped stats and snps stats have been set.
'p' has been set.
'mu' and 'sigma' have been set.

> standardize(x) <- 'mu'
```

As the whole genome is not available, we generate a random positive matrix to play the role of the GRM:

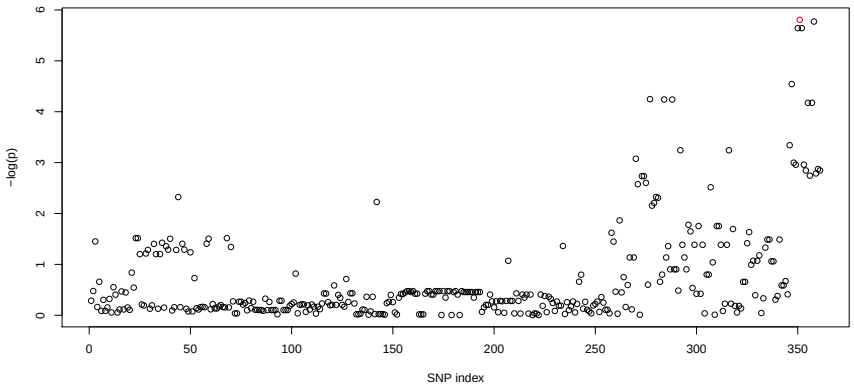
```
> set.seed(1)
> R <- random.pm(nrow(x))
```

And we simulate a phenotype with an effect of the SNP #351, and a polygenic component:

```
> y <- 2 + x %*% c(rep(0,350),0.25,rep(0,ncol(x)-351)) +
      lmm.simu(tau = 0.3, sigma2 = 1, eigenK=R$eigen)$y
```

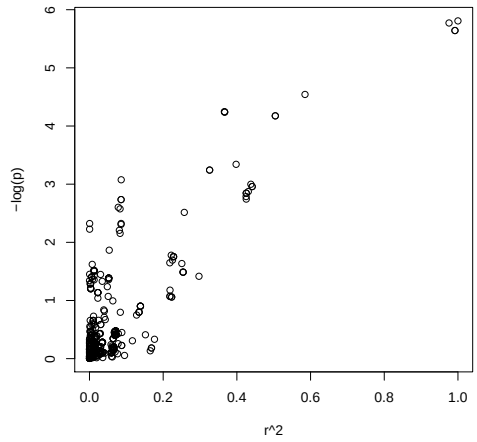
The following code performs the association test and displays the result under the form of a (mini) Manhattan plot:

```
> t <- association.test(x, y, eigenK = R$eigen)
> plot(-log10(t$p), xlab="SNP index", ylab = "-log(p)",
       col = c(rep(1,350),2,rep(1,ncol(x)-351)))
```



We colored the point corresponding to the SNP #351 in red. We see that there are other SNPs with low association p-values: these are the SNPs in LD with SNP #351. We can confirm this by plotting the p-values (again on log scale) as a function of the LD (measured by r^2):

```
> lds <- LD(x, 351, c(1,ncol(x)))
> plot(lds, -log10(t$p), xlab="r^2", ylab="-log(p)")
```



5 What you can hope for in later releases

A few things that may be added to the package in the near future:

- Reading `.ped` files
- Maximum Likelihood Linkage Disequilibrium estimation
- LMM models fitting written in equationnal form
- A faster version of `association.test`
- More functions and models for association testing
- Anything you asked the maintainer for