HaploSim, a package for simulating haplotypes

Albart Coster, John Bastiaansen.
Albart.Coster@wur.nl, John.Bastiaansen@wur.nl

October 16, 2008

INTRODUCTION

Simulation of genotype data is of critical importance in genetic studies. Assessing accuracy of methods for estimation of genetic parameters and evaluation of strategies generally is performed using simulated datasets. Simulation software is especially of importance in the context of genetic marker data.

Software available for simulation of genotype data include SimPed (Leal et al., 2005), GWAsimulator (Li and Li), genomeSIM (Dudek et al., 2006) and HAPSIMU (Zhang et al., 2008). Programs are generally designed for specific purposes; GWAsimulator and genomeSIM are designed for case-control studies. HAPSIMU is designed to evaluate the effect of population structure in population based association studies. SimPed is a more general simulation program for generating data of a large number of marker loci in general pedigrees.

Simulations need to be realistic and an important aspect in simulation studies is evaluation of this. Important aspects are if the simulated population is in mutation-drift equilibrium, patterns of linkage disequilibria and number of heterozygous loci. Parameters from a simulated population need to be comparable to their theoretical expectations.

Posibilities for testing specific parameters is limited in most available simulation packages. Packages are generally available as compiled executable files with limited input and output possibilies. Another simulation paradigm is to provide users with specific tools or *functions* which can then be used for building specific simulation scenaria.

In this paper, we introduce the package HaploSim for R (R Development Core Team, 2007). HaploSim provides a series of simulation functions embedded in R. Integration into R allows to easily program simulation scenaria and evaluate simulation outcome. The paper is organized as follows. First, we briefly introduce HaploSim. Next, we show three examples and analyse some outcome. We conclude the paper with final remarks.

DESCRIPTION

HaploSim uses S4 class definitions (Chambers, 1998). Class haplotype exploits the biallelic nature of SNP markers by only storing genome positions of one allele of a marker locus (say the *1* allele). The 1 alleles in an object of class haplotype are represented as integers which denote their relative position on the chromosome and are stored in the `snp` slot the object. The other slots of class haplotype are `qtl`, which is a list with the additive effects of the genes on the chromosome; `hID` which represents the objects ID number (in correctly programmed simulation unique for each haplotype object); `phID0` and `phID1` which represent the ID number of the two parental haplotypes.

The following example shows object `haplo`. The object is of class haplotype and has a 1 allele at positions (46,288,409,453,525,548,788,881,887,937)'. The object's empty `qtl` slot demonstrates that

there are no gene alleles with additive effect different from 0 in this object. The object's ID number is 1 and parental haplotypes are unknown.

```
An object of class "haplotype"
Slot "snp":
 [1]  46 288 409 453 525 548 788 881 887 937

Slot "qtl":
$`103`
[1] 0.957

$`573`
[1] 0.453

$`899`
[1] 0.678


Slot "hID":
[1] 1

Slot "phID0":
[1] NaN

Slot "phID1":
[1] NaN
```

Package HaploSim provides class haploList to store objects of class haplotype. Class haploList extends the standard list class of R and includes slot `genDist` which specifies the haplotype size in Morgan and slot `nDec` which specifies the number of decimal positions of the marker locations. The combination of `genDist` and `nDec` gives the number of marker positions. In the previous example, `genDist` was 1 Morgan and `nDec` was 3. Consequently, there where 1,000 marker positions on the haplotype.

Package HaploSim utilizes the Haldane mapping function to model recombinations (Haldane, 1919). The number of recombinations in each meiosis is drawn from a poisson(`genDist`) distribution. The recombinations are subsequently positioned at random positions along the chromosome and absence of interference is assumed. Only an odd number of recombinations in a segment produce recombinant segments in the offspring haplotype. The basic function is HaploSim is `SampleHaplotype`, which is available for custom use and is used in higher level functions as `SampleHaplotypes` and `SamplePedigree`.

Currently, HaploSim does only implement additive gene action of genes, which are refered to as *Quantitative Trait Loci* (QTL). QTL are stored in slot `qtl` of class `haplotype` which is a list. The elements of the list can be accessed by index or by name. The names are integers which represent the qtl positions on the chromosome and the elements are the additive effects. To avoid computational load, HaploSim does only additive effects different from 0.

The following example shows object `haplo` again. QTL are located at positions (691,889,955)' and additive effects vary between 0.042 and 0.328.

```
An object of class "haplotype"
Slot "snp":
```

```
 [1]  46 288 409 453 525 548 788 881 887 937

Slot "qtl":
$`691`
[1] 0.246

$`889`
[1] 0.0421

$`955`
[1] 0.328


Slot "hID":
[1] 1

Slot "phID0":
[1] NaN

Slot "phID1":
[1] NaN
```

Function `AssignQTL` assigns qtl loci to the haplotypes of object of class `hList`. Arguments of `AssignQTL` include `sigma2qtl` which represents the genetic variance of each QTL and `MAF` which represents the Minor Allele Frequency above which SNP loci are candidate for becoming QTL. Setting argument `rmCausSNP = FALSE` removes *causative* marker loci from the locus panel.

Function `getAll` retrieves alleles from an object of class `hList` and returns a matrix with number of rows equal to the number of haplotypes and number of columns equal to the total number of locus positions in the object. To retrieve qtl alleles from an object of class `hList`, one should set argument `what = 'qtl'`.

DEMONSTRATION

We show three reproducible examples of implementing HaploSim. The first example simulates a random mating population for 100 generations. The second example simulates a crossbred population. The third generation simulates haplotypes through a known pedigree.

Example 1: Random Mating population

In this example, we simulate a random population for 100 generations. Chromosome length is 1 Morgan and the number of marker positions is 10,000. The base population consists of 200 haplotypes which are heterozygous at 100 marker positions with MAF equal to 0.5.

```
> nDec <- 3
> genDist <- 1
> nLoc <- 100
> pSnp <- seq(0, 1, length.out = nLoc)
> baseHaplos <- SampleHaplotypes(nHaplotypes = 200, genDist = genDist,
+     nDec = nDec, pSnp = pSnp)
```

Mutation probability is maintained at 1E-05 along the generations. The number of haplotypes is maintained at 200 and mating is random. Effective population size is 199 ($Ne = 2N - 1$, ($Ne = 2N - 1$, Crow and Kimura (1970) p99)). In the following code lines, we program the script to simulate the 100 random generations. The objects of class haplotype are stored in object `hList` of class haploList and the `hList` object in each generation simply overwrites `hList` of the previous generation.

Every thenth generation, allele frequencies are calculated with function `calc.freq` and a matrix with pattern of Linkage Disequilibrium decay is calculated with function `ldDecay`. Functions `calc.freq` and `ldDecay` are from additional package mfwSchatting which provides a series of functions to analyse output from package HaploSim. The allele frequencies and LD pattern are stored in object `output`. The names of `output` identify the generation number.

```
> hList <- SampleHaplotypes(orig = baseHaplos, prMut = 1e-05)
> output <- list()
> for (g in 2:10) {
+     hList <- SampleHaplotypes(orig = hList, prMut = 1e-05)
+     if (g%%2 == 0)
+         output[[as.character(g)]] <-
+           list(calc.freq(hList),ldDecay(hList,min(100,
+                     maxSnp = min(dim(getAll(hList))[2]))))
+ }
```

Elements of `output` were extracted and used to generate Figures and . The histograms in the panels of Figure vary between generations and indicate that the population was not in an equilibrium situation. Note peaks at extreme that the allele frequencies will have a peak at low frequency, due to the combination of drift and mutations. The number of cases at high frequencies is expected to be lower because the probability of the combination of drift and mutation is less likely.

Figure shows strength of Linkage Disequilibrium plotted against distance between markers over generations. Again, there is no evidence for an equilibrium situation. The line plotted in the panels shows the theoretical expectation of LD pattern (Sved, 1971).

Figures and indicate that an equilibrium situation was not reached after 100 generations. We simulated the population for 900 additional generations. Distribution of allele frequencies is shown in the left panel of Figure **??** and the right panel of this figure shows the Linkage Disequilibrium pattern with the theoretical expectation.

Example 2: Crossbreeding population

In this section simulate a crossbred population. The parental individuals each crossbred individual originate from two populations and consequently have different genetic background. Parental population *p0* is simulated from the haplotypes simulated in example 1 and now are assigned to object `hList0`. Haplotypes for parental population *p1* are simulated identically as in example 1 and are assigned to object `hList1`.

Parental individuals are simulated from independent populations. We use equal parameters for simulation of both populations. simulating the data as in the previous example to simulate purebred parental populations `p0` and `p1`.

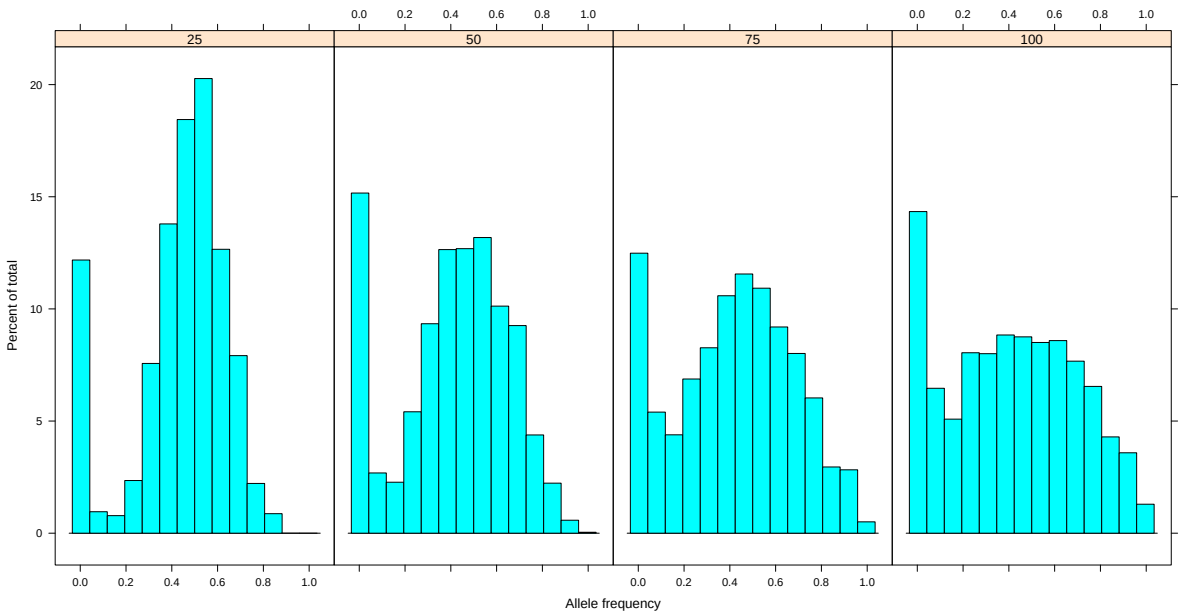

Figure 1: Distribution of allele frequency over 100 generations. Unstable distribution shows that the population is far from an equilibrium situation.

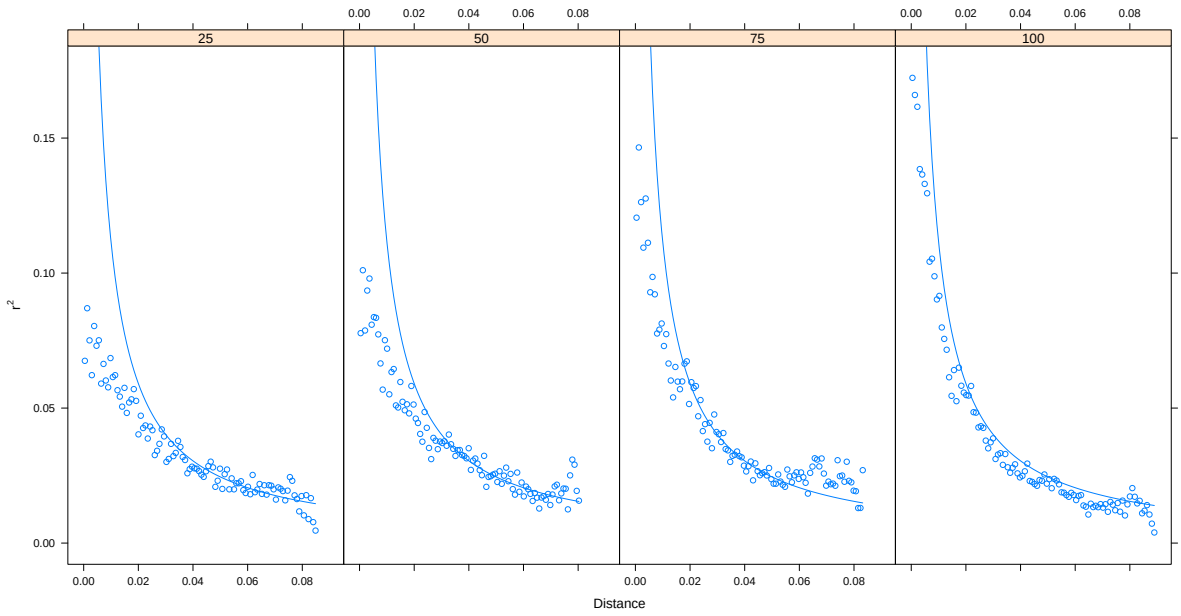

Figure 2: Linkage Disequilibrium pattern each 25th generation. Line is the theoretical Linkage Disequilibrium pattern from (Sved, 1971). Difference between observations and line give indicate that the population is not in an equilibrium situation.

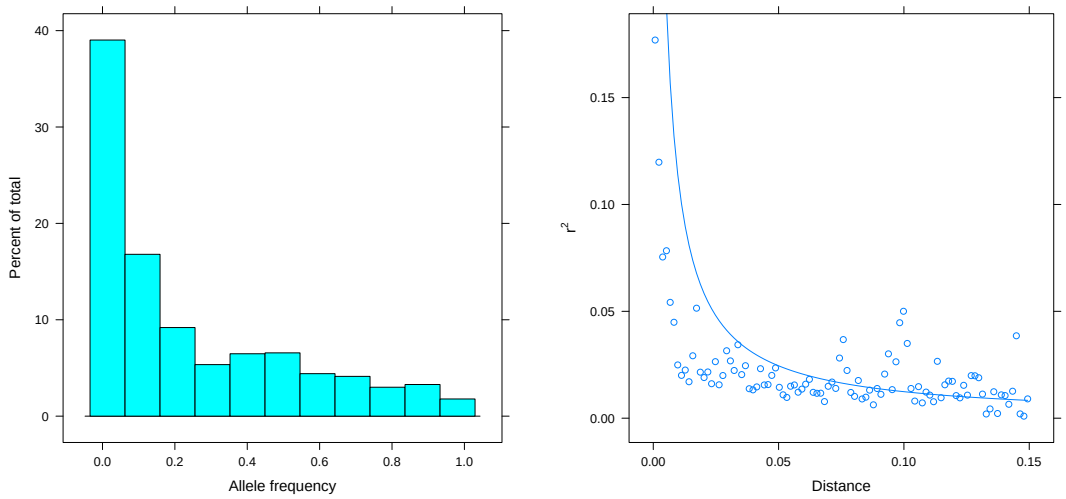

Figure 3: Histogram of allele frequencies (a) and pattern of Linkage Disequilibrium (b) after 1,000 generations.

A practical way to simulate individuals in HaploSim is to simply record which haplotype pair belongs to which individual. In this example, we simulate *p0* individuals by combining two random haplotypes from `hList0`. Two consecutive elements in vector `i0` identify the two haplotypes of an individual. Individuals in *p1* are simulated identically. Finally, we reorder `hList0` and `hList1` according to `i0` and `i1`.

```
> i0 <- sample(1:length(hList0))
> hList0 <- hList0[i0]
> i1 <- sample(1:length(hList1))
> hList1 <- hList1[i1]
```

In the previous example, we did not simulate gene loci. Here, we will assign additive effects to alleles of specific marker loci on the haplotypes in `hList0` and `hList1`. We use function `AssignQTL` for this purpose. In this example, we use equal variance for all gene loci, specified with argument `sigma2qtl = 1`. Ten percent of the markers with MAF$>$ 0.1 are selected to become QTL and consequently are removed from the marker panel of both populations.

```
> hList0 <- AssignQTL(hList0, frqtl = 0.1, sigma2qtl = 1,
+     MAF = 0.1)
> hList1 <- AssignQTL(hList1, frqtl = 0.1, sigma2qtl = 1,
+     MAF = 0.1)
```

We simulate the crossbred population by simulating haplotypes from both populations and combining them into one object of class `haploList`. We use the `gg` argument of function `SampleHaplotypes` to communicate that consequtive haplotypes belong to one individual. We set argument `QTL = TRUE` because we have QTL. Finally, we create haplotype combinations and reorder `cb` according to these combinations.

```
> cb <- SampleHaplotypes(orig = hList0, gg = 1:length(hList0),
+     nMeioses = 1, QTL = TRUE, prMut = 1e-05)
```

```
> cb <- c(cb, SampleHaplotypes(orig = hList1, gg = 1:length(hList1),
+     nMeioses = 1, QTL = TRUE, prMut = 1e-05))
> icb <- numeric(length(cb))
> icb[seq(1, length(cb), by = 2)] <- sample(1:(length(cb)/2))
> icb[seq(2, length(cb), by = 2)] <- sample((length(cb)/2) +
+     1:(length(cb)/2))
> cb <- cb[icb]
```

We might want to count the number of different haplotypes in and between populations. In a higher number of shared haplotypes between populations and this gives an idea about the genetic relation between populations. The package `mfwSchatting` provides functions for analysing haplotype data and we use function `countHaplotypes` from this package.

```
> hCount <- countHaplotypes(hList = c(hList0, hList1,
+     cb), gg = c(rep("pb0", length(hList0)), rep("pb1",
+     length(hList1)), rep("cb", length(cb))))
> hCount[upper.tri(hCount)] <- NA
```

Table 1: Haplotype count within (diagonal elements) and between populations (off-diagonal elements).

	cb	pb0	pb1
cb	200		
pb0	13	200	
pb1	14	0	199

Table 2 summarizes characteristics of the three populations. The number of marker loci in the crossbred population obviously is larger than in the two purebred populations, and the same is true for the number of QTL loci.

Table 2: Summary of simulated crossbreeding scheme. Purebred parental lines are pb0 and pb1. Crossbred offspring population is cb.

	pb0	pb1	cb
nloc	1013	974	1812
nqtl	53	46	99
vg	59.411	39.840	38.669

Example 3: Simulating through a pedigree

In this example, we apply function `SamplePedigree` to simulate haplotypes through a pedigree, obtained from pig breeding company IPG (Beuningen, Netherlands). The pedigree consists of 566 individuals over 8 generations.

We apply function `SamplePedigree` to simulate haplotypes through the pedigree. Haplotypes for individuals with unknown parents are sampled from list of *base* haplotypes `hList0`, which was simulated in Example 2. Because QTL were already assigned to marker loci of these haplotypes in

the previous example, we set argument `QTL = TRUE` to to continue simulating these QTL through the pedigree.

```
> phList <- SamplePedigree(orig = hList, ped = ped,
+     prMut = 1e-05, QTL = TRUE)
> ped <- phList$ped
> hList <- phList$hList
```

Function `SamplePedigree` returns a list. The first element of the list is the pedigree with two additional columns which identify the two haplotypes belonging to each individual. The second element is an object of class haploList, which contains the haplotypes.

Genotype value of an individual is the sum of the additive effects at its two haplotypes. In the following code, we calculate genotype values for the individuals and assign them to an additional column in the pedigree. We simulate phenotype by adding random normal deviate to the the genotype with mean 0 and variance equal to the genotype variance.

```
> hVal <- apply(getAll(hList, what = "q"), 1, sum)
> names(hVal) <- names(hList)
> mm0 <- match(ped$hID0, names(hVal))
> mm1 <- match(ped$hID1, names(hVal))
> ped$G <- hVal[mm0] + hVal[mm1]
> ped$P <- ped$G + rnorm(n = nrow(ped), sd = sd(ped$G))
```

Now we can use the simulated pedigree to perform analyses. In this example, we perform a Whole Genome Analysis to identify marker loci which are in strong Linkage Disequilibrium with QTL. We use a simple analysis to identify significant marker loci and assing the names of these loci to object `signLoci`.

```
> alleles <- getAll(hList)
> genotypes <- alleles[mm0, ] + alleles[mm1]
> signpVals <- apply(genotypes, 2, function(x) {
+     model <- lm(ped$P ~ x)
+     summary(model)$coefficients[, 4][2] < 0.05
+ })
> signLoci <- names(signpVals)[which(signpVals)]
```

The number of significant marker loci was 174 from the initial set of 1211 marker loci. In this analysis, we ignored problems due multiple testing and population structure.

FINAL REMARKS

With package HaploSim, we developped a flexible package for simulation of population genetic data. The most important advantage of HaploSim in comparison to other available simulation packages is its integration into R. This enables users to easily program their own simulation schemes and enables analysis of the results using available statistical and graphical functions in R.

R is a high level programming language which comes at the cost of computing speed. The design of HaploSim, however, uses the sparse nature of genotype data. In addition, computing time does not scale with the number of markers but with chromosome size because ffunction `SampleHaplotype` uses vectorized computing to simulate recombinations.

HaploSim available from the repository of R packages CRAN, http://cran.r-project.org/.

REFERENCES

J. M. Chambers. *Programming with Data. A Guide to the S Language*. Springer-Verlag, 1998.

J. F. Crow and M. Kimura. *AN INTRODUCTION TO POPULATION GENETICS THEORY*. Alpha Editions, 1970.

S. M. Dudek, A. A. Motsinger, D. R. Velez, S. M. Williams, and M. D. Ritchie. Data Simulation Software for Whole−Genome Association and Other Studies in Human Genetics. In *Pacific Symposium on Biocomputing*, pages 499–510, 2006.

J. B. S. Haldane. THE COMBINATION OF LINKAGE VALUES, AND THE CALCULATION OF DISTANCES BETWEEN THE LOCI OF LINKED FACTORS. *J. Genet.*, (8), 1919.

S. M. Leal, K. Yan, and Müller-Myhsokm B. Simped: A Simulation Program to Generate Haplotype and Genotype Data for Pedigree Structures. *Human Heredity*, (60):119–122, 2005.

C. Li and M. Li. GWAsimulator: a rapid whole-genome simulation program. *Bioinformatics*, 24(1): 140–142.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL `http://www.R-project.org`.

J. A. Sved. Linkage disequilibrium and homozygosity of chromosome segments in finite populations. *Theoretical Population Biology*, 2(2):125–141, 1971.

F. Zhang, J. Liu, J. Chen, and H.-W. Deng. HAPSIMU: a genetic simulation platform for population-based association studies. *BMC Bioinformatics*, 9(1):331, 2008.