R Package hypred: Simulation of Genomic Data in Applied Genetics

Frank Technow

package version 0.5

1 Introduction

The package **hypred** is intended for simulating high density genomic data. Its focus is on genomic applications in applied genetics, such as genomic prediction and selection [a], but it should be useful in a related fields as well.

The function `hypredGenome` is used to define the genome parameters: number and length of chromosomes, and number of SNP loci. It also simulates a genetic map. This functions returns an object that is of class `hypredGenome`. The function `hypredNewMap` is used to modify the genetic map, and the function `hypredNewQTL` to assign QTL. The function `hypredFounder` will create two homozygous founder lines according to the object returned by `hypredGenome`. The meiosis is simulated by the function `hypredRecombine`. This function takes two haploid chromosome sets (as found in a sex cell) and returns a recombined haploid chromosome set (i.e. the gamete). The meiosis is simulated according to the count-location model, simulation of mutations is possible as well. The functions `hypredTruePerformance` and `hypredTpGenomeSpecific` return the true genotypic values of individuals given their genotypes. `hypredTruePerformance` uses the QTL effects found in the `hypredGenome` object, `hypredTpGenomeSpecific` allows to specify new and genome specific effects. Various types of design matrices (to be used in modeling algorithms) are created from the genotypes by the function `hypredCode`. All functions mentioned above are generic, with methods defined for the `hypredGenome` object.

2 Basic concepts, possibilities and limitations

2.1 Genome

Chromosomes Only diploid genomes are supported. An arbitrary number of chromosomes, with individual lengths, can be simulated. All distances are genetic distances with unit Morgan (M).

Marker loci An arbitrary number of marker loci can be assigned to the chromosomes (and with arbitrary I mean 1, 2, 3, $\ldots \gg$ 10,000). However, the number of marker loci needs to be equal for each chromosome. The markers are treated as bi-allelic single nucleotide polymorphisms (SNP) with alleles 1 and 0. The genetic map of the markers can be generated randomly (via a uniform distribution over the chromosome) or explicitly set by the user [1].

QTL Quantitative trait loci (QTL) can be assigned to marker loci. The QTL alleles are treated as identical apart from one causative SNP (the one the QTL is assigned to). These causative SNP can be unobserved (i.e. not included into the set of markers available for analysis) or observed (i.e. perfect markers). The QTL can have additive effects (a) or additive and dominance effects (d). The effects are defined according to the Falconer scale [b], where a is half the difference between the homozygous genotypes (**0-0** and **1-1**) and d is the deviation of the heterozygous genotypes (**0-1** and **1-0**) from the mean of the two homozygous genotypes. Genotypic values are expressed as deviations from the "lower" homozygot (**0-0**), so that the genotypic value of the genotype (**1-1**) would be $2a$, for example.

The number of QTL with additive effect only, the number of QTL with additive and dominance effect and the number of perfect markers, must be the same for all chromosomes. If the number of QTL needs to be different per chromosome, a workaround is to assign effects of $a = d = 0$ to some of the QTLs.

Only QTL for one trait can be considered at once. However, multiple traits can be simulated by modifying the numbers, positions and effects of QTL.

[1] The meiosis engine provided by function `hypredRecombine` can handle markers with multiple alleles, coded as integers (1,2,3,...). This feature is not yet officially supported

Meiosis The meiosis is simulated according to the count-location model that assumes no interference and agrees with the Haldane mapping function.[c][2] Here the number of crossing over on a random meiotic product of a chromosome is Poisson distributed with parameter $\lambda = L$ (L being the length of the chromosome in Morgan). The locations of the crossing over follow a uniform distribution over the interval $(0, L)$.

Mutations The occurrence of mutations, μ denoting the mutation rate, is simulated according to a simple count-location model as well. The number of observed SNP mutations in the gamete follows a Poisson distribution with parameter $\lambda = \mu$ times the number of loci on a chromosome times the number of chromosomes. Then each SNP-loci has an equal chance of being mutated, irrespective on whether mutations occurred in its vicinity. A mutation event always changes a **1** allele to a **0** allele and vice versa (i.e. **0** to **1**; **1** to **0**). During one meiosis, a given SNP-loci can only be mutated twice if the number of mutations exceeds the total number of SNP. This should rarely happen and is included only for computational reasons. μ can be different for marker loci and QTL.

Since all the simulated loci are treated as SNP, μ should be interpreted as μ per *base-pair* per meiosis.

2.2 Working concepts

This section concerns mainly the central, workhorse function `hypredRecombine`, which simulates the recombination.

Low level approach `hypredRecombine` is as low level as it can get. It just takes two haploid chromosome sets (as can be found in a sex cell) and returns a single gamete. To create populations, loops have to be constructed in which the function is called (examples follow). Hence, basic to intermediate proficiency in R is required to perform more complex simulations. *This is a low level package on purpose.* The low level nature gives the user full flexibility to do whatever he wants, and, I confess, not including functions like "`hypredCreateNCIII_designBC`" makes life easier for me.

[2] This fact was used to validate the algorithm because observed recombination frequencies could be compared with expected frequencies as calculated by the inverse of the Haldane mapping function from the map distances.

In a sense, the function `hypredRecombine` follows some of the UNIX "Software tools" programming principles. Namely the "do one thing well" principle and that the input and output data should be the same. The idea is that this function is used as a building block in larger programs for simulating very complex populations.

Sex cell - gamete approach This is of course just the consequence of the low level thing. As mentioned, function `hypredRecombine` just takes two haploid chromosome sets (as can be found in a sex cell) and returns a single gamete. So when trying to create a population, one has to think and work in terms of sex cells and gametes. This means that one also needs to have some basic knowledge of population genetics, because one needs to know how to get from a sex cell or single gamete to some complex population.

low level data A gamete, i.e. the in- and output of `hypredRecombine` and most other functions, is simply an one row integer matrix that represents the genotype as elements of 1 and 0 (corresponding to the **1** and **0** alleles). An advantage of this is that the data is easily accessible, designing a gamete with a special genotype, for example, just involves creating a matrix accordingly. A disadvantage is that there is more responsibility at the user side to assure that the data is valid. However, the functions include many validity checks and using only unmodified data produced by functions from the package should guaranty validity.

The accessibility of the in- and output data and of the `hypredGenome` object allows the users to quickly write their own extractor and modifier functions respectively to adapt the ones provided. Hence, intermediate to advanced users of R should find it very simple to include some epistatic interactions into function `hypredTruePerformance`, for example.

computing efficiency The computations for simulating the meiosis are performed by a C routine that is called through `hypredRecombine`. Still, having to extensively use loops in R might be somewhat inefficient, mainly in terms of computing time. However, my experience shows that even for simulation of high volume data ($>$ 50,000 loci, many hundreds of individuals and hundreds of generations of random mating), the computations are reasonably fast, even on a computer that is mediocre at best.

And, of course, the C source code is available to everyone. So when computing time is really an absolutely critical factor, one could just use the underlying C function (which is used in basically the same way as the calling R function), to write a customized C routine.

3 Demonstration

In this last part, usage of the package will be demonstrated. This is done by a session in which the following szenario is simulated:

1. 250 F_2 individuals from two founder lines are created
2. this population is then random mated for 50 generations
3. after this, the best individual is selected according to its phenotype ($h^2 = 0.5$)
4. a DH population is then generated from this selected individual
5. this DH population is the training population for some genomic selection algorithm, hence the final step is to generate the phenotypes and a design matrix

The simulated genome is defined as follows:

- two chromosomes, of length $1M$ each
- 100 loci per chromosome, of these, five are QTL (three with purely additive effect, two with dominance effect as well). One of the QTL on a chromosome is a perfect (observed) marker. Hence there are 96 markers per chromosome.

The simulated data is not quite "high density", the package can do allot more than 200 loci/marker. However, this is supposed to be an introduction and demonstration, so I want to keep the amount of data that is created low. The interested user can easily run the example with, say, 5000 loci, without experiencing a serious hike in computing time and RAM requirement.

3.1 Genome definition

After loading the package,

```
R> library(hypred)
```

the first step in a simulation will usually be the definition of the the genome parameters and the creation of the `hypredGenome` object. This is done with the function `hypredGenome`[3]. The argument `num.chr` gives the number of chromosomes, `len.chr`, their lengths in Morgan and `num.snp` the number of SNP loci *per chromosome*.

```
R> genome <- hypredGenome(num.chr = 2,
                          len.chr = c(1.0, 1.0),
                          num.snp = 100)
```

This creates an object of class `hypredGenome` which serves as a storage container for all the information about the genome that is needed by the other functions. Please refer to the help page of the object for details on its expected content. There are methods for the functions `summary` and `show` that will both print the same short information about the genome (`summary` also invisibly returns a list with the displayed parameters).

```
R> summary(genome)

The defined genome is characterized as follows:


2 chromosomes of lengths  1 1  M
100 SNP markers plus 0 perfect SNP markers per chrom.
0 QTL with additive effect per chrom.
Of these, 0 also have (has) a dominance effect
```

The function `hypredGenome` by default simulates a genetic map of the loci on the chromosome, where the positions of the loci are uniformly distributed.

```
R> slot(genome, "pos.snp")[1:5] ## pos. of first 5

[1] 0.02226214 0.02417047 0.02745109 0.03510093 0.06256874
```

This map can be modified with the function `hypredNewMap` which takes only one argument appart from the object[4]. This argument is `new.map` and takes a

[3] Notice that all function, that are intended to be called by the user, are prefixed with "`hypred`" to avoid naming clashes with other packages. Notice further that the argument names usually follow the pattern: `aspect-of-thing-of-interesst` then `thing-of-interesst`. Whenever reasonable, the arguments are in small case, only the first three letters of the word are used, words are separated by a dot and articles and prepositions are not used. So for example, the argument for the "number of chromosomes" is `num.chr`.

[4] The `hypredGenome` object is usually the first argument in every function

numeric vector that gives the map position of all the loci. In this case there are 100 loci per chromosome, so there need to be 200 elements in the vector. The elements `[1:100]` give the positions of the loci on chromosome 1, elements `[101:200]` the positions of the loci on chromosome 2. On the new map, the loci are arranged evenly and in equal distance.

```
R> map <- rep(seq(0, 0.99, by = 0.01 ), 2)
R> genome <- hypredNewMap(genome,
                          new.map = map)
R>

R> slot(genome, "pos.snp")[1:5] ## pos. of first 5

[1] 0.00 0.01 0.02 0.03 0.04
```

There is a convenience function, called `hypredSNPdist`, that returns the genetic distance (`d`) and the expected recombination frequency (`r`) between two loci on a chromosome. The arguments are self explaining.

```
R> hypredSNPdist(genome,
                 chromosome = 1,
                 SNP1 = 1,
                 SNP2 = 3)

        [,1]
d 0.02000000
r 0.01960528
```

Until now, there are no QTL assigned and all loci are markers. Assignment of QTL is done with the function `hypredNewQTL`. Apart from the object, there are five additional arguments to it. Arguments `new.id.add`, `new.id.dom` and `new.id.per.mar` give the IDs of the loci which are assigned QTL with additive effects, QTL with dominance effects and perfect markers. Since all QTL must have an additive effect, `new.id.add` actually gives the IDs of all QTL and `new.id.dom` must be a subset of it. The ID of a loci is its index in the vector of all loci in the object. If, as in this example, the object holds 200 loci, then the ID of the locus with lowest map postion of chromosome 1 is 1, the next has ID 2 and so on. The ID of the locus with lowest map position of chromosome 2 is 101. Note that the number of QTL, QTL with dominance effect and the number of perfect markers must be the same on each chromosome.

The five QTL per chromosome will be assigned to the 1st, 20th, 40th, 60th and 80th locus on each chromosome.

```
R> qtl.ids <- c(1, 20, 40, 60, 80,  ## chr. 1
              101, 120, 140, 160, 180)  ## chr. 2
```

The third of these is the one that has a dominance effect as well.

```
R> qtl.dom.ids <- c(40,  ## chr. 1
                  140)  ## chr. 2
```

The second will be the perfect marker

```
R> per.mar.ids <- c(20,  ## chr. 1
                  120)  ## chr. 2
```

The remaining two arguments to `hypredNewQTL` are `new.eff.add` and `new.eff.dom`. They give the additive and dominance effects. These are assigned to the QTL according to the corresponding element in the vectors given to `new.id.add` respectively `new.id.dom`. So the second additive effect in the vector given to `new.eff.add` would be assigned to the QTL with ID 20. For simplicity, the additive effects are 1 and the dominance effects are 0.5.

```
R> genome <- hypredNewQTL(genome,
                        new.id.add = qtl.ids,
                        new.id.dom = qtl.dom.ids,
                        new.id.per.mar = per.mar.ids,
                        new.eff.add =  rep(1, 10),
                        new.eff.dom = c(0.5, 0.5)
                        )
```

After assigning QTL, the printed summary changes accordingly, and now includes summary statistics of the effects (not very usefull in this example, though).

```
R> summary(genome)

The defined genome is characterized as follows:

2 chromosomes of lengths  1 1  M
95 SNP markers plus 1 perfect SNP markers per chrom.
```

```
5 QTL with additive effect per chrom.
Of these, 1 also have (has) a dominance effect

Summary of distribution of additive effects:

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1


Summary of distribution of dominance effects:

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.5     0.5     0.5     0.5     0.5     0.5
```

The definition of the genome is thereby complete. Note that the map, as well as the QTL can be modified later on at any point.

3.2 F_2 base population

The gametes of two founder lines that differ at each locus can be created by the function `hypredFounder`. The only additional argument to it is `prob.snp` which gives the probability that founder line 1 has the **1** allele at any locus. For simplicity, this is set to 1, so that line 1 has all the **1** alleles and line 2 all the **0** alleles. The output is a two row matrix, with the gamete of line 1 in row 1 and the gamete of line 2 in row 2. Because the lines are assumed to be absolutely homozygous, each can have only one distinct gamete.

```
R> founder1 <- hypredFounder(genome,
                             prob.snp = 1)[1, ]
R> founder2 <- hypredFounder(genome,
                            prob.snp = 1)[2, ]
```

As mentioned in the beginning, genotypes are stored as one row matrices:

```
R> founder1[1:5] ## first 5 loci

1 2 3 4 5
1 1 1 1 1

R> founder2[1:5] ## first 5 loci
```

```
1 2 3 4 5
0 0 0 0 0
```

These matrices could have been created directly, `hypredFounder` is just a convenience function.

To recombine these parental genomes, the workhorse function `hypredRecombine` is used. This is the central function of the package, see its help page, as well as the descriptions in section 2 for details. The *two haploid parental chromosome sets in a sex cell* that are to be recombined, are given as one row matrices via the arguments `genomeA` and `genomeB`. Which set is given to which doesn't matter. Because the two founder lines are homozygous, it follows that this is the sex cell of a F_1 individual.

The mutation rate for both SNP marker and QTL is set to $2.5 \cdot 10^{-5}$, haplotype blocks are not used.

```
R> set.seed(114)## to see a recomb. event
R> new.gamete <- hypredRecombine(genome,
                                 genomeA = founder1,
                                 genomeB = founder2,
                                 mutate = TRUE,
                                 mutation.rate.snp = 2.5 * 10^-5,
                                 mutation.rate.qtl = 2.5 * 10^-5,
                                 block = FALSE)
```

The output is again a one row matrix with the recombined gamete.

```
R> new.gamete[1:20] ## first 20 loci

 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
```

To create a F_2 population with size $N = 250$, 500 such gametes have to be produced. This is best done in a loop. For this a matrix is pre-allocated that has as many rows as gametes and as many columns as loci.

```
R> N <- 250 ## number of individuals
R> F2 <- matrix(nrow = N * 2,
                ncol = 200)
```

Then `hypredRecombine` is called within the loop and the gametes are assigned to subsequent rows.

```
R> for(i in 1:(N*2))
   {
     F2[i, ] <- hypredRecombine(genome,
                                genomeA = founder1,
                                genomeB = founder2,
                                mutate = TRUE,
                                mutation.rate.snp = 2.5 * 10^-5,
                                mutation.rate.qtl = 2.5 * 10^-5,
                                block = FALSE)
   }
```

The rows 1 and 2 represent the two chromosome sets of F_2 individual 1, the rows 3 and 4 represent the two chromosome sets of F_2 individual 2 and so on.

3.3 Random mating

Random mating is done according to the following algorithm:

1. create two matrices with dimension $[2N, l]$ (l is the no. loci) and call them `random.mate.pop` and `random.mate.pop.temp`
2. store the genotypes of all the individuals in `random.mate.pop`
3. create two recombined gametes from each individual in `random.mate.pop` and store them in `random.mate.pop.temp` at the same position as the parental chromosome sets in `random.mate.pop`
4. randomly permutate the rows of `random.mate.pop.temp`, thats one generation of random mating
5. repeat 2 to 4 for a particular number of generations

This algorithm is implemented below with two nested loops (outer for generations, inner for individuals).

```
R> G <- 50 ## number of generations
R> random.mate.pop <- F2 ## identical to the F2 at start
R> random.mate.pop.temp <- matrix(nrow = N*2, ncol = 200)

R> for(generation in 1 : G) ## loop over generations
   {
     gameteIndex1 <- 1 ## indexing variables
```

```
  gameteIndex2 <- 2
  for(indiv in 1 : N) ## loop over individuals
    {
      ## gamete 1
      random.mate.pop.temp[gameteIndex1,] <-
        hypredRecombine(genome,
                        genomeA = random.mate.pop[gameteIndex1,],
                        genomeB = random.mate.pop[gameteIndex2,],
                        mutate = TRUE,
                        mutation.rate.snp = 2.5 * 10^(-5),
                        mutation.rate.qtl = 2.5 * 10^(-5),
                        block = FALSE)
      ## gamete 2
      random.mate.pop.temp[gameteIndex2,] <-
        hypredRecombine(genome,
                        genomeA = random.mate.pop[gameteIndex1,],
                        genomeB = random.mate.pop[gameteIndex2,],
                        mutate = TRUE,
                        mutation.rate.snp = 2.5 * 10^(-5),
                        mutation.rate.qtl = 2.5 * 10^(-5),
                        block = FALSE)

      ## increment to next individual
      gameteIndex1 <- gameteIndex1 + 2
      gameteIndex2 <- gameteIndex2 + 2
    } ## end for N

  ## permutate
  random.mate.pop <- random.mate.pop.temp[sample(1 : (N * 2)), ]
} ## end for G
```

The matrix `random.mate.pop` now contains the random mated population.

3.4 Selection

The genotypic values of the individuals can be obtained by the function `hypredTruePerformance`. This function takes two further arguments: `genotypes`, to which the matrix that stores the genotypes is given and `DH` which is `TRUE` or `FALSE`, de-

pending on whether the matrix contains DH individuals or not. This argument needs to be set to `FALSE` in this case, the implications of `DH = TRUE` will be discussed later on.

```
R> g.value <- hypredTruePerformance(genome,
                                    random.mate.pop,
                                    DH = FALSE)
R> summary(g.value)
```

```
       V1
 Min.   : 3.00
 1st Qu.: 9.00
 Median :10.50
 Mean   :10.58
 3rd Qu.:12.00
 Max.   :18.00
```

Note that the expected value (assuming that all QTL are in linkage equilibrium (LE), p denoting the frequency of the **1** allele and q the frequency of the **0** allele) would be 10.601 [b]

$$\sum_l 2p(a + dq)$$

which is in good agreement with the observed mean of 10.584[5].

The observed variance of the genotypic values is 4.862, which is as well in good agreement[6] with the expectation under perfect LE, which would be 4.747 [b].

$$\sum_l 2pq\big(a + d(q - p)\big)^2 + (2pqd)^2$$

In order to obtain $h^2 \approx 0.5$, a normaly distributed random variable, with mean 0 and the same variance is added to arrive at the phenotypic values.

```
R> var.env <- var(g.value)
R> phen.value <- g.value + rnorm(250, 0,  sqrt(var.env))
```

[5] The computations for the expectations are included into the .Rnw file, but are not echoed. They are of course available in the source.

[6] The good agreement between observed and expected values implies that 50 generations of random mating were enough to dissipate the coupling phase linkage and bring the QTL into LE.

Individual number 42 has the highest phenotypic value and thus is selected. The two rows (chromosome sets) in `random.mate.pop` that correspond to it are 83 and 84. A new matrix (called `selected`) is created that stores only the genotype of this individual.

```
R> index.individual <- which(phen.value == max(phen.value))
R> index.row1 <- index.individual * 2 - 1
R> index.row2 <- index.individual * 2
R> selected <- random.mate.pop[c(index.row1, index.row2), ]
```

3.5 DH population

The DH population is produced similarily to the F_2, just that only 1 gamete is produced per individual.

```
R> DH <- matrix(nrow = N,
              ncol = 200)

R> for(i in 1:N)
   {
     DH[i, ] <- hypredRecombine(genome,
                                genomeA = selected[1, ],
                                genomeB = selected[2, ],
                                mutate = TRUE,
                                mutation.rate.snp = 2.5 * 10^-5,
                                mutation.rate.qtl = 2.5 * 10^-5,
                                block = FALSE)
   }
```

Note that it is not necessary to explicitely double the gametes. The function `hypredTruePerformance`, as well as the function `hypredCode`, which creates design matrices, both have an argument named `DH`. Setting this argument to `TRUE`, makes the function assume that the matrix given to the argument `genotypes` holds DH individuals with one row representing one individual. This feature should be used as much as possible, because the matrix then needs to be only half as big, which in turn saves memory! The mean genotypic value of the DH population is 18.056. The genotypic value of the of the individual from which the DH population was derived was 18. This can easily be verified by looking at its QTL-genotype

```
R> selected[, qtl.ids] ## all QTL
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    0    1    1    1     1
[2,]    1    1    1    1    1    1    0    1    1     1
```

```
R> selected[, qtl.dom.ids] ## QTL with dominance
```

```
     [,1] [,2]
[1,]    1    1
[2,]    1    1
```

```
R> ## effect a = 1 times the number of 1 alleles
R> sum(selected[, qtl.ids] == 1) +
   ## 0.5 if the dominance QTL have genotype 01 or 10 (hence sum = 1)
   (sum(selected[, qtl.dom.ids[1]]) == 1)*0.5 +
   (sum(selected[, qtl.dom.ids[2]]) == 1)*0.5
```

```
[1] 18
```

It follows also that the expected mean of the DH population (assuming QTL in LE) must be 18, which is in good agreement with the observed value.

Phenotypic values of the DH population are generated in the same way as for the random mating population. Note that the same environmental variance is used, so the heritability will be different.

```
R> g.value.DH <- hypredTruePerformance(genome,
                                       DH,
                                       DH = TRUE)
R> phen.value.DH <- g.value.DH + rnorm(250, 0,  sqrt(var.env))
```

The last demonstrated step is the creation of a design matrix. This is done by the function `hypredCode`. Its additional arguments are: `genotypes`, `DH` and `type`. The first two have the same meaning as for `hypredTruePerformance`. The argument `type` takes a character string that gives the type of design matrix to be created. Currently four different types can be created, see `?hypredCode` for details. The one used in this example is `type = "012"` which means that element X_{ij} (i indexing the individual, j the locus) will be 0 if the locus is **0-0**, 1 if the locus is **1-0** or **0-1** and 2 if the locus is **1-1**, i.e. coded is the number of **1** alleles at a locus. This type of design matrix is suitable to estimate substitution effects.

```
R> design.DH <- hypredCode(genome,
                           genotypes = DH,
                           DH = TRUE,
                           type = "012")
```

Note that QTL loci that are not designated as perfect markers, are not included in the design matrices. For example, locus 1 was a QTL, but not a perfect marker, hence it is removed (the first row of the output are the column names of the matrix, which correspond to the loci IDs):

```
R> design.DH[1, 1:5]
```

```
2 3 4 5 6
2 0 0 0 0
```

The QTL assigned to locus 20, was a perfect marker and hence is included:

```
R> design.DH[1, 17:21]
```

```
18 19 20 21 22
 2  2  2  0  0
```

With the phenotypes, genotypic values and the design matrix available, one can proceed with the actual analysis of this data or with deploying some genomic selection/prediction method. However, this is out of the scope of this vignette and package.

References

[a] Meuwissen, T. H. E., B. J. Hayes, and M. E. Goddard: Prediction of Total Genetic Value Using Genome-Wide Dense Marker Maps. *Genetics 157*(4), 1819-1829.

[b] D. S. Falconer: *Introduction to Quantitative Genetics*. Oliver and Boyd., London, 1967.

[c] Wu, R., M. Chang-Xing, G. Casella: *Statistical Genetics of Quantitative Traits*. Springer, New York, 2007.