MateR Usage

Javier Fernández-González

2025/09/10 (yyyy/mm/dd)

Contents

Introduction

This package allows to perform **optimal genomic mating** for optimal cross selection in a sigle trait or **multi-trait** framework. It is highly versatile, supporting breeding schemes used in **self-pollinated, clonal and hybrid crops**. It can consider the number of selfing cycles performed before selection, it can account for dominance effects and it can be used to breed to maximize the specific combining ability with a given set of testers. It supports **diploid, allopolyploid and autotetraploid** species, as well as the use of double haploids.

In genomic mating, there is a set of parental lines available and we want to answer the question of how to optimally combine them to obtain the best possible offspring. To that end, two criteria have to be balanced:

1. **Usefulness criterion**. This value combines the **family average** (the average genotypic value of all individuals generated by a cross) and the **within-family variance**. The more variance a family has, the more diverse it is and the higher the likelihood of producing exceptionally high-fitness individuals. Furthermore, a large number of offspring generated for a family allows for a better exploration of its diversity, although there are diminishing returns as this number increases. Selected families should ideally present a **high average and a high variance**. Moreover, the **number of offspring** generated per family should be **larger for high-variance families** than for low-variance ones, as in the former there are more potential gains from exploring its diversity. This can be done by repeating the crosses that generate high-variance families more times than the low-variance crosses.
2. **Across-family diversity**. Selecting diverse parents is key for keeping the genetic diversity of the population, which is essential for long-term gain. There is a trade-off between maximizing diversity and usefulness, as high diversity requires selecting a balanced mix of high performance and low performance parents. MateR package maximizes usefulness with the constraint that the diversity cannot drop below an user-defined threshold.

MateR **requires license keys**, but they will be provided for **free to public bodies**, such as, Universities, and non-profit organizations. You can get license keys by contacting javier.fgonzalez@upm.es or j.isidro@upm.es.

Please **make sure to have TrainSel installed before attempting to install MateR**. In the following sections, we will showcase the usage of MateR for different breeding schemes.

```r
#Load the library
library(MateR)
```

```
##

## ***********************************************************

## TrainSel package is required

## Available in https://github.com/TheRocinante-lab/TrainSel

## To obtain license keys, please contact

## javier.fgonzalez@upm.es or j.isidro@upm.es.

## Free license keys will be provided to public bodies,

## such as, Universities, and non-profit organizations.

## ***********************************************************
```

```r
set.seed(1234)
```

```r
#To get license keys, please contact javier.fgonzalez@upm.es or j.isidro@upm.es
#They will be provided for free to public bodies, such as, Universities, and non-profit organizations
username <- NULL #type your username here.
password <- NULL #type your password here
username_TrainSel <- NULL #type your TrainSel username here.
password_TrainSel <- NULL #type your TrainSel password here
```

Self-Pollinated Crop

In a **self-pollinated crop**, there is typically a set of elite, **fully homozygous parental lines**. These parents are crossed among themselves to generate new variability. The resulting **F1 offspring are then repeatedly selfed** to increase their homozygosity. At some point during the selfing process, field trials are performed and the best genotypes are selected. MateR allows to find the best mating plan for **maximizing the usefulness of the offspring in the generation in which they are evaluated, i.e., after several cycles of selfing.**

First, we will load some example data:

```r
#Load the example data
data(ExampleDataDiploid)

#1) There are 100 parental lines
length(Parents)
```

```
## [1] 100
```

```r
Parents[1:5] #names of the parental lines
```

```
## [1] "g1_1" "g1_2" "g1_3" "g1_4" "g1_5"
```

```r
#2) Genotypic information of the parents
#marker matrix counting the number of times the alternative allele is present in each locus
dim(Markers) #100 parental lines, 1000 Markers
```

```
## [1]  100 1000
```

```r
Markers[1:5,1:5]
```

```
##      SNP1 SNP2 SNP3 SNP4 SNP5
## g1_1    0    2    0    2    0
## g1_2    0    0    0    0    0
## g1_3    0    0    0    0    0
## g1_4    0    0    0    0    0
## g1_5    0    0    0    0    0
```

```r
sum(Markers == 1) #no heterozygous position. Fully inbred parental lines
```

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

```r
#3) Additive marker effects for two traits, yield (YLD) and maturity (MAT):
markereffects$YLD[1:5]
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
##  0.2288611 -1.0132430 -0.7770513 -0.4573172 -1.2113599
```

```r
markereffects$MAT[1:5]
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
##  1.6079669  1.1523572 -0.2692306  0.2806039  1.4864272
```

```r
#4) Coefficients for a multi-trait selection index
coefficients
```

```
##        YLD        MAT
##  0.9594875 -0.2817511
```

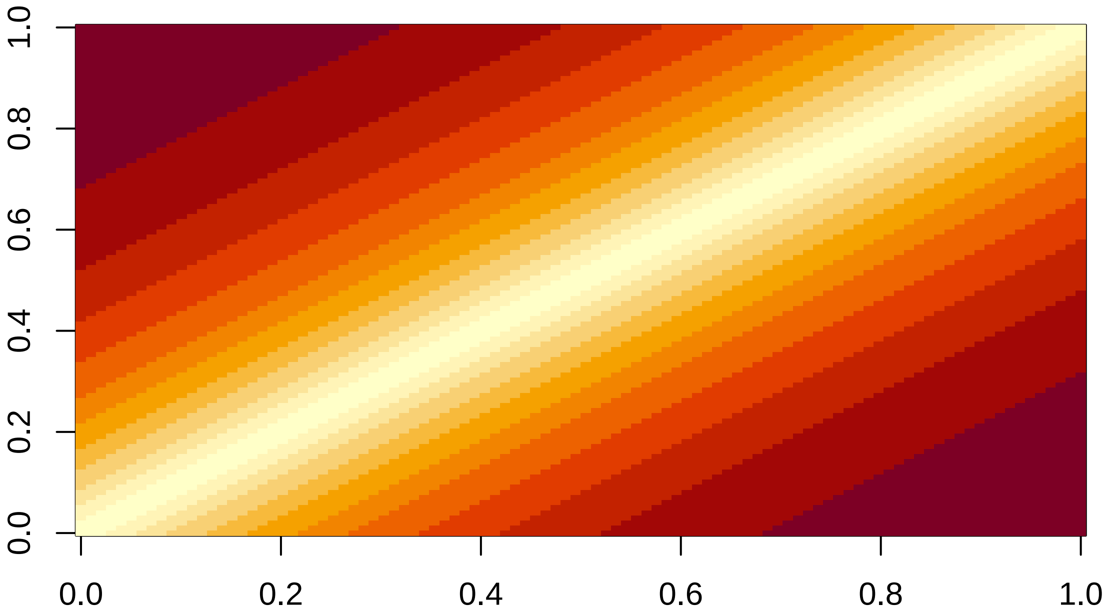
```r
#5) List of matrices of frequencies of recombination per each chromosome
str(Chromosomes) #list indicating which markers belong to each chromosome
```

```
## List of 8
##  $ : chr [1:100] "SNP1" "SNP2" "SNP3" "SNP4" ...
##  $ : chr [1:150] "SNP101" "SNP102" "SNP103" "SNP104" ...
##  $ : chr [1:150] "SNP251" "SNP252" "SNP253" "SNP254" ...
##  $ : chr [1:100] "SNP401" "SNP402" "SNP403" "SNP404" ...
##  $ : chr [1:150] "SNP501" "SNP502" "SNP503" "SNP504" ...
##  $ : chr [1:150] "SNP651" "SNP652" "SNP653" "SNP654" ...
##  $ : chr [1:100] "SNP801" "SNP802" "SNP803" "SNP804" ...
##  $ : chr [1:100] "SNP901" "SNP902" "SNP903" "SNP904" ...
```

```
#assume all chromosomes have a length of 150 cM
ChromosomeCentimorgans <- rep(150, length(Chromosomes))
#Assume Haldane map function
c_list <- create_c_list(Chromosomes = Chromosomes,
                        ChromosomeCentimorgans = ChromosomeCentimorgans,
                        model = "Haldane")
c_list[[1]][1:5,1:5] #0 --> recombination never happens; 0.5 --> independent segregation
```

```
##            SNP1       SNP2       SNP3       SNP4       SNP5
## SNP1 0.00000000 0.01492425 0.02940303 0.04344964 0.05707698
## SNP2 0.01492425 0.00000000 0.01492425 0.02940303 0.04344964
## SNP3 0.02940303 0.01492425 0.00000000 0.01492425 0.02940303
## SNP4 0.04344964 0.02940303 0.01492425 0.00000000 0.01492425
## SNP5 0.05707698 0.04344964 0.02940303 0.01492425 0.00000000
```

```
image(c_list[[1]]) #Visualize one chromosome
```



```
#6) Haplotype data (optional)
H_parents[[1]][,1:5] #which allele is present in each chromosome
```

```
##           SNP1 SNP2 SNP3 SNP4 SNP5
## DH_gamete    0    1    0    1    0
## DH_gamete    0    1    0    1    0
```

Regarding the matrices of recombination frequencies `c_list`, it is important to note that, ideally, this information should be extracted from a linkage map for maximum accuracy. However, if the map is not available, a rough approximation can be made with the `create_c_list()` function. Nevertheless, `create_c_list()` relies on a few simplistic assumptions and it will incur in some error. For more details, please type `?create_c_list` in R.

Using this data, we can find the optimal mating plan for making crosses among the **100 parental lines**. As an example, we will assume the following parameters:

1. We have enough resources in our breeding program to **make 80 crosses**.
2. To ensure that enough diversity is retained, we want to ensure that no more than 5% of additive standard deviation is lost every selection cycle.
3. We will make the field trials to select the best **offspring in the F4** generation, i.e., after **3 cycles of selfing** from the original F1 offspring. As the final objective is obtaining fully inbred lines with no heterozigosity, we will focus on improving the additive breeding values of the individuals. **Dominance effects are not of interest** in this scheme as they are not heritable.
4. From **each cross** between two parental lines, we will obtain **20 different individuals in the F4**.
5. In the F4 generation, we will **select the best 5 individuals from each family** to continue the selfing process. The best individuals are selected through phenotypic selection. Thus, **selection accuracy** will be the **square root of heritability**, specifically narrow sense heritability as only additive effects are of interest in this scheme.
6. We will consider that broad sense heritability is 0.3 for all traits and narrow sense heritability is 0.2. Heritability is very small due to the low replication of the field trials in which the best 5 individuals from each family would be selected (due to low seed availability).

We can create some parameters that reflect the information above:

```r
#TrainSel hyperparameters. We will use the demo version to be able to run this
#quicker. However, the demo version is very limited and will not reach
#an optimal solution. It is convenient for testing but good results are not guaranteed.
control = TrainSel::SetControlDefault(size="demo",
                                      verbose = F)

# #Recommended parameters (slower but converge to a much better solution):
# control = TrainSel::SetControlDefault(size="large",
#                                       complexity = "high_complexity",
#                                       verbose = F)

#limit diversity loss:
#limit of diversity loss = losing 5% of additive standard deviation
PropSD_limit <- 0.05
Number_of_crosses <- 80 #We are limited to less than 100 by the demo version!
Selfing_cycles <- 3 #3 cycles of selfing: we will perform selection in the F4
F4_individuals_per_cross <- 20
F4_selected_per_family <- 5
h2 <- 0.2
H2 <- 0.3
```

Single Trait Optimization, Maximize Yield.

We will start with a single-trait optimization. MateR by default maximizes a trait, making yield maximization easy:

```r
#get the marker effects  for the desired trait
markereffectsYLD <- list(YLD = markereffects$YLD)
```

```
#Perform optimization to maximize yield with the desired paremeters:
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffectsYLD, #marker effects for desired trait
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list,
                       coefficients = NULL,
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_1/g1_42    g1_1   g1_42                 6       9.666677
## 2  g1_3/g2_16    g1_3   g2_16                 5      15.958627
## 3 g1_14/g1_42   g1_14   g1_42                 5      26.039619
## 4 g1_14/g1_43   g1_14   g1_43                 4      21.481458
## 5 g1_14/g2_22   g1_14   g2_22                 8       6.298680
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             18.27085   27.93753
## 2                             17.63513   33.59375
## 3                             18.16316   44.20278
## 4                             16.69554   38.17700
## 5                             21.61432   27.91300
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04893595
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##               [,1]
## [1,] 5.911078e-09
```

One important thing to note about the `GenomicMatingMT()` function, is that it has two arguments for available parental lines, `Parents1` and `Parents2`. If these two arguments are different, the function will consider that the only crosses allowed are the ones involving one parent from `Parents1` and another parent from `Parents2`. In this case, we have set **`Parents1=Parents2=Parents`**. This means that **all crosses from a full diallel** of the lines in `Parents` **are allowed.**

The parameter `PropSD` is used to control how much diversity should be mantained. It is directly interpretable as the expected proportion of additive standard deviation lost in each selection cycle. As additive standard deviation is directly proportional to genetic gain, `PropSD` can be seen as the reduction in the rate of genetic gain per selection cycle due to dwindling diversity, allowing us to optimize its value. In case you are more familiar to the use of inbreeding rate to control for the diversity of the mating plan, MateR also supports it (`deltaF` parameter):

```r
inbreedingRate <- 0.05 #desired limit of 5% of inbreeding per generation:

#Perform optimization to maximize yield with the desired paremeters:
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffectsYLD, #marker effects for desired trait
                       n = Selfing_cycles,
                       ##########################################################
                       deltaF = inbreedingRate, #use inbreeding rate instead of PropSD
                       ##########################################################
                       size = Number_of_crosses,
                       c_list = c_list,
                       coefficients = NULL,
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
## Warning in GenomicMatingMT(Parents1 = Parents, Parents2 = Parents, Markers =
## Markers, : As inbreeding rate tracks IBD similarity, we recommend working with
## a numerator relationship matrix instead of a genomic relationship matrix. You
## can provide it in the "G" argument. Computing inbreeding with the genomic
## relationships can work well if pedigree is not available, although it becomes
## difficult to interpret its value.
```

```r
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_2/g1_43    g1_2   g1_43                 5      0.1345081
## 2  g1_3/g2_22    g1_3   g2_22                10     -5.0072292
## 3 g1_14/g1_15   g1_14   g1_15                 1     20.1182211
## 4 g1_14/g1_41   g1_14   g1_41                 2     18.6582398
## 5 g1_14/g1_42   g1_14   g1_42                 1     26.0396187
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             17.46082   17.59532
## 2                             22.22919   17.22196
## 3                              10.97779   31.09601
## 4                              13.49045   32.14869
## 5                              11.19269   37.23231
```

```
#the mating plan is below the desired threshold of 5% inbreeding rate:
out$deltaF
```

```
##             [,1]
## [1,] 0.04861049
```

```
#We can also see diversity loss that corresponds to this inbreeding rate
out$PropSD
```

```
## [1] 0.04023673
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##               [,1]
## [1,] 9.220117e-10
```

The main problem of working with inbreeding rate is that calculating inbreeding from the genomic relationship matrix incurs in some error. It should instead be computed from a numerator relationship matrix, which is not always available. Therefore, we recommend using proportion of standard deviation lost to control for inbreeding in the population.

Single Trait Optimization, Maximize Yield, Constraint number of families

If a specific number of families is desired, it is possible to do it with the `n_families` parameter. For instance, if we want to have 150 individuals in the F4 after selection and we select 5 individuals per family, we need to have exactly 30 different families. We can do it as follows:

```
n_families <- 30 #specifiy desired number of families

#Perform optimization to maximize yield with the desired paremeters:
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffectsYLD, #marker effects for desired trait
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list,
                       coefficients = NULL,
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       ##########################################################
                       n_families = n_families, #specifiy desired number of families
                       ##########################################################
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
```

```
                        Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##       Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1 g1_1/g1_15    g1_1   g1_15                 3       3.745279
## 2 g1_1/g1_43    g1_1   g1_43                 2       5.108516
## 3 g1_2/g1_42    g1_2   g1_42                 1       4.692669
## 4 g1_3/g1_43    g1_3   g1_43                 2      10.175549
## 5 g1_3/g2_33    g1_3   g2_33                 2       4.652830
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             15.72780   19.47308
## 2                             12.58160   17.69012
## 3                             10.78873   15.48140
## 4                             13.25833   23.43387
## 5                             15.02543   19.67826
```

```
length(unique(out$OptimalMatingScheme$Family)) #There are exactly 30 families as desired
```

```
## [1] 30
```

```
#diversity loss :
out$PropSD
```

```
## [1] 0.04988595
```

```
#The value above is close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 4.421234e-08
```

Please note that adding more constraints may reduce the ability of the algorithm to converge to an optimal solution. We advise adding a constraint for the number of families only if it is really needed.

Single Trait Optimization, Maximize Yield, Constraint number of times each parent is crossed

Sometimes, due to practical reasons, a parent can only be crossed a limited number of times. This is specially evident in animal breeding, where females can often be crossed only once per cycle. The `Parental_limits` argument allows to introduce this constraint. Here, we will show an example in which the first 20 parents can participate in up to 4 crosses while the remaining 80 parents can only participate in 2 each:

```
Parents_limits <- data.frame(parents = Parents,
                             #minimum number of times each parent has to be used.
                             #We set it at zero (we don't force any parent to be chosen)
                             minimum = rep(0,length(Parents)),
                             maximum = c(rep(4,20), #maximum 4 crosses for first 20 parents
                                         rep(2,80))) #maximum 2 crosses for last 80 parents
head(Parents_limits)
```

```
##   parents minimum maximum
## 1    g1_1       0       4
## 2    g1_2       0       4
## 3    g1_3       0       4
## 4    g1_4       0       4
## 5    g1_5       0       4
## 6    g1_6       0       4
```

```
#Perform optimization to maximize yield with the desired paremeters:
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       ##########################################################
                       Parents_limits = Parents_limits,
                       ##########################################################
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffectsYLD, #marker effects for desired trait
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list,
                       coefficients = NULL,
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##       Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_1/g1_8    g1_1    g1_8                 1     -22.780117
## 2 g1_1/g1_16    g1_1   g1_16                 1     -16.493995
## 3 g1_1/g1_24    g1_1   g1_24                 1      -8.088138
## 4 g1_1/g2_37    g1_1   g2_37                 1     -18.862824
## 5  g1_2/g1_5    g1_2    g1_5                 1     -37.991893
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             10.72873 -12.051382
## 2                             10.80382  -5.690175
## 3                             10.49290   2.404760
## 4                             11.86262  -7.000208
## 5                             11.06468 -26.927216
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.01271398
```

```
#The value above is not close to the desired limit because parental constraints
#force a mating plan with lower diversity loss
```

```
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 6.29874e-05
```

```
#check that no parent is crossed more times than the limit
ParentsCount <- c()
scheme <- out$OptimalMatingScheme
for (parent in Parents_limits$parents) {
  tmp <- max(0, sum(scheme$Number_Of_Crosses[scheme$Parent1 == parent]))
  tmp <- tmp + max(0, sum(scheme$Number_Of_Crosses[scheme$Parent2 == parent]))

  ParentsCount <- c(ParentsCount, tmp)
}
names(ParentsCount) <- Parents_limits$parents

sum(ParentsCount > Parents_limits$maximum) #no parent is crossed more times than the limit
```

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

Please note that adding more constraints may reduce the ability of the algorithm to converge to an optimal solution. Simultaneously adding the constraints of `n_families` and `Parental_limits` is allowed but not advised, as it is possible that no solution exists that meets both simultaneously while keeping across-family diversity above the desired threshold.

Single Trait Optimization, Minimize Maturity

Continuing with single-trait optimization, we will minimize maturity. As **MateR by default maximizes a trait**, we have to indicate that we want **to minimize** it. This can be easily done by **creating a coefficient (weight) of minus one for the trait**. This will result on the optimization maximizing the opposite of maturity, which is the same as minimizing it:

```
#get the marker effects  for the desired trait
markereffectsMAT <- list(MAT = markereffects$MAT)
Coefficient <- c(MAT=-1) #give a weight of minus one to maturity to minimize it

#Perform optimization to maximize yield with the desired parameters
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffectsMAT,#marker effects for desired trait
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list,
                       #########################################################
                       coefficients = Coefficient, #negative coefficient: minimize trait
                       #########################################################
```

```
                        offspring_per_cross = F4_individuals_per_cross,
                        within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                        control = control,
                        n_selected_per_family=F4_selected_per_family,
                        Username=username, #you can get one by contacting us
                        Password=password, #you can get one by contacting us
                        Username_TrainSel=username_TrainSel, #you can get one by contacting us
                        Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1 g1_19/g2_24   g1_19   g2_24                10       22.33016
## 2 g1_22/g2_27   g1_22   g2_27                 8       33.28431
## 3 g1_27/g2_17   g1_27   g2_17                 8       30.34137
## 4 g1_34/g2_28   g1_34   g2_28                11       23.81420
## 5 g1_38/g2_14   g1_38   g2_14                 7       38.33003
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             21.53661   43.86677
## 2                             22.30284   55.58715
## 3                             21.34696   51.68833
## 4                             21.93577   45.74997
## 5                             22.91514   61.24517
```

```
#The expected breeding values are the opposite of the values in the table:
ExpectedMaturityValues <- -out$OptimalMatingScheme$Usefulness
ExpectedMaturityValues[1:5] #Very small values as desired, maturity has been minimized
```

```
## [1] -43.86677 -55.58715 -51.68833 -45.74997 -61.24517
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04961914
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 7.654338e-08
```

Multi-Trait Optimization

A more realistic scenario involves **selecting the offspring for maximum yield and minimum maturity simultaneously**. As a result, we want to be able to consider both traits when performing the genomic mating. To that end, we need to work within the framework of **index selection**: we have some coefficients that correspond to the weight of each trait. We will maximize the index scores, which are simply the weighted sum of the traits.

Getting the best values for the coefficients can be difficult. One possible way of getting them is the gain() function from StageWise package. Details on how to use it are available in Vignette3 of StageWise (https://github.com/jendelman/StageWise). Here, we will just use the coefficients available in the example data.

```
#Multi-trait coefficients:
#Positive value for yield: we want to maximize it
#Negative value for maturity: we want to minimize it
#Weight for yield has a larger magnitude than the one for maturity. This means
#that yield has more economical importance than maturity.
coefficients
```

```
##        YLD        MAT
##  0.9594875 -0.2817511
```

```
markereffects$YLD[1:5] #additive marker effects for yield
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
##  0.2288611 -1.0132430 -0.7770513 -0.4573172 -1.2113599
```

```
markereffects$MAT[1:5] #additive marker effects for maturity
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
##  1.6079669  1.1523572 -0.2692306  0.2806039  1.4864272
```

```
#both traits are correlated, making multi-trait analysis more important
cor(markereffects$YLD, markereffects$MAT)
```

```
## [1] 0.2663142
```

```
#Perform optimization to maximize the index scores (multi-trait):
start <- Sys.time()
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffects, #marker effects for both traits
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list,
                       #########################################################
                       coefficients = coefficients, #coefficients for index scores
                       #########################################################
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
end <- Sys.time()
timeLD <- end-start #save this for later
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_4/g1_42    g1_4   g1_42                 4       3.401262
## 2  g1_7/g2_16    g1_7   g2_16                 5      10.444912
## 3 g1_14/g1_42   g1_14   g1_42                 3      17.395103
## 4 g1_14/g2_37   g1_14   g2_37                12      -7.337887
## 5 g1_15/g1_41   g1_15   g1_41                 6       9.012014
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             16.30168   19.70295
## 2                             16.29564   26.74055
## 3                             14.89557   32.29067
## 4                             20.10284   12.76496
## 5                             16.28691   25.29892
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04990137
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 2.193463e-09
```

```
#save this for later:
Family_values_LD <- out$FamilyValues
```

It is important to note that the **usefulness values in the output** correspond to the **multi-trait index scores**.

Using Haplotype Data

Computing **within-family variance** requires taking into account the **linkage disequilibrium (LD)** of the parental lines. This is controlled in the `GenomicMatingMT()` function through the `LD` argument, which defaults to `LD="Approx"`. This results in the assumption that the **LD pattern of any parental line is well represented by the LD computed on the overall parental population**. `LD="Approx"` combines very fast computational **speed** with good **performance**, but it can incur in some error. For instance, if the parental lines present **strong population structure**, the phasing patterns in the haplotypes may be different in each subpopulation. Therefore, the LD of a random parent would be well represented by the overall LD patterns of its subpopulation, but not necessarily by the LD patterns in the entire population. Alternatively, MateR allows to accurately calculate the **LD patterns of any specific parental line from its haplotype matrix** (setting `LD="Full"`). This is completely **robust to the population structure** in the parental lines and makes very consistent and **highly accurate predictions**. However, it presents three main **disadvantages**: 1) **haplotype data has to be available** (which is trivial for fully homozygous parental lines but requires phasing information othewise), 2) it is more **computationally intensive** and 3) it is less robust to high error in the estimation of marker effects (which is often the case in empirical datasets). Next, we will show how to use the haplotypes:

Compute haplotype data internally

This option is only available **if the parental lines are fully homozygous**. If this is the case, we simply need to set `LD="Full"` and phasing information will be computed by MateR:

```r
sum(Markers == 1) #no heterozygous positions in parental lines --> fully homozygous
```

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

```r
LD="Full" #Use haplotypes to fully account for linkage disequilibrium
```

```r
#Perform optimization to maximize yield with the desired parameters:
start <- Sys.time()
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       c_list = c_list,
                       #########################################################
                       LD=LD, #Fully account for linkage disequilibrium
                       #########################################################
                       markereffects = markereffectsYLD, #marker effects for desired trait
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       coefficients = NULL,
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
end <- Sys.time()
timeHaplotypes <- end-start #save this for later
```

```r
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_3/g1_42    g1_3   g1_42                 7       14.73371
## 2 g1_14/g1_15   g1_14   g1_15                 6       20.11822
## 3 g1_14/g1_46   g1_14   g1_46                 3       18.35724
## 4 g1_14/g2_16   g1_14   g2_16                 3       27.26454
## 5 g1_14/g2_33   g1_14   g2_33                 4       15.95874
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             15.69967   30.43338
## 2                             18.51464   38.63287
## 3                             14.85021   33.20745
## 4                             15.81184   43.07637
## 5                             21.58595   37.54469
```

```r
#diversity loss:
out$PropSD
```

```
## [1] 0.04957359
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##               [,1]
## [1,] 2.797272e-08
```

```
#save this for later:
Family_values_Haplotypes <- out$FamilyValues
```

This approach allows to make extremely accurate predictions but it is slower. In is possible to **parallelize** it by indicating the desired number of cores to use in the argument `ncores` of the `GenomicMatingMT()` function. However, it is important to note that **there is some of overhead involved** in the parallelization, which can reduce its efficiency in some scenarios.

Explicitly provide haplotype data

MateR also allows to **manually provide phasing information** for parental lines through the `H_parents` argument of `GenomicMatingMT()` function. This is **only needed if the parental lines are not fully homozygous**. An example of this will be provided for the autotetraploid, clonally propagated crop (see the **"Using Haplotypes for Non-Homozygous Parents"** section).

Disregard linkege disequilibrium

Above, we have shown how to compute within-family variance with haplotypes or without haplotypes using the LD patterns of the entire parental population. Both approaches perform well and it is recommended to use one of them. Nevertheless, if a simpler approach is desired, MateR supports a final alternative. **Within-family variance can be calculated assuming no LD** (independent loci). This is rather **simplistic** and generally has **substantially more error** that the other approaches. However, it has the advantages of **not needing** to know any information about the **recombination frequencies** in the genome and it is the **fastest methodology**.

```
#Example for Haplotype data unavailable:
LD="Ind" #do not use linkage disequilibrium of the parents as an input for computing
#within-family variance. This is only recommended for strong population structure!

#Perform optimization to maximize yield with the desired parameters:
start <- Sys.time()
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       ##########################################################
                       c_list = NULL, #No need for recombination frequencies
                       LD=LD, #LD="Ind" --> disregard LD
                       ##########################################################
                       markereffects = markereffectsYLD, #marker effects for desired trait
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       coefficients = NULL,
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
end <- Sys.time()
timeNoLD <- end-start #save this for later
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_3/g1_14    g1_3   g1_14                 6      17.221662
## 2  g1_4/g1_46    g1_4   g1_46                 8       5.850617
## 3 g1_11/g1_43   g1_11   g1_43                 5       8.190844
## 4 g1_14/g1_15   g1_14   g1_15                 4      20.118221
## 5 g1_14/g2_16   g1_14   g2_16                 4      27.264536
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             18.54724   35.76890
## 2                             18.14119   23.99181
## 3                             17.15151   25.34235
## 4                             16.94338   37.06160
## 5                             18.73736   46.00189
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04957852
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 7.830494e-09
```

```
#save this for later:
Family_values_no_LD <- out$FamilyValues
```

Comparison of Linkage Disequilibrium Approaches

Let's now compare the three LD approaches available in MateR in terms of computational time and performance:

```
#computational time:
timeHaplotypes #time required when using haplotypes to compute LD
```

```
## Time difference of 38.27851 secs
```

```
timeLD #time required when assuming that LD of a parent is equal to LD in the population
```

```
## Time difference of 20.32122 secs
```

```
timeNoLD #time required when disregarding LD
```

```
## Time difference of 17.85283 secs
```

As expected, using **haplotypes is slower**, but its time requirements are in the same order of magnitude as its alternatives. For a performance comparison, please refer to the MateR paper.

Hyperparameter tuning

Until now, we have assumed that the desired **parameters** such as the **size of the mating plan** and the **limit for proportion of standard deviation lost** are known. However, if these values are not known with certainty, it may be useful to perform some testing on how they would impact the overall mating plan.

The optimization pipeline can be divided into **two computationally intensive steps**:

1. Compute the average value of each family and **within-family variance**. This step is independent from the optimization hyperparameters and it is often the computational bottleneck when using haplotype data.

2. Perform **optimization** to maximize usefulness.

When doing repeated tests, it is advised to **perform the first step only once** and then **repeatedly run the second step** with different hyperparameters to perform the tuning. The first step can be manually calculated using the `calculateFamilyValues()` function or it can be **extracted from the output of a previous optimization**. We will use the latter option, as it is generally easier. We will show how to test different values of the `PropSD` limit:

```
#extract family mean and sd from the previous optimization
FamilyValues <- Family_values_Haplotypes
FamilyValues$mean[1:5]
```

```
## $`g1_1/g1_2`
## [1] -9.19232
##
## $`g1_1/g1_3`
## [1] 0.8487202
##
## $`g1_1/g1_4`
## [1] -0.3519923
##
## $`g1_1/g1_5`
## [1] -33.01788
##
## $`g1_1/g1_6`
## [1] -9.099269
```

```
FamilyValues$sd[1:5]
```

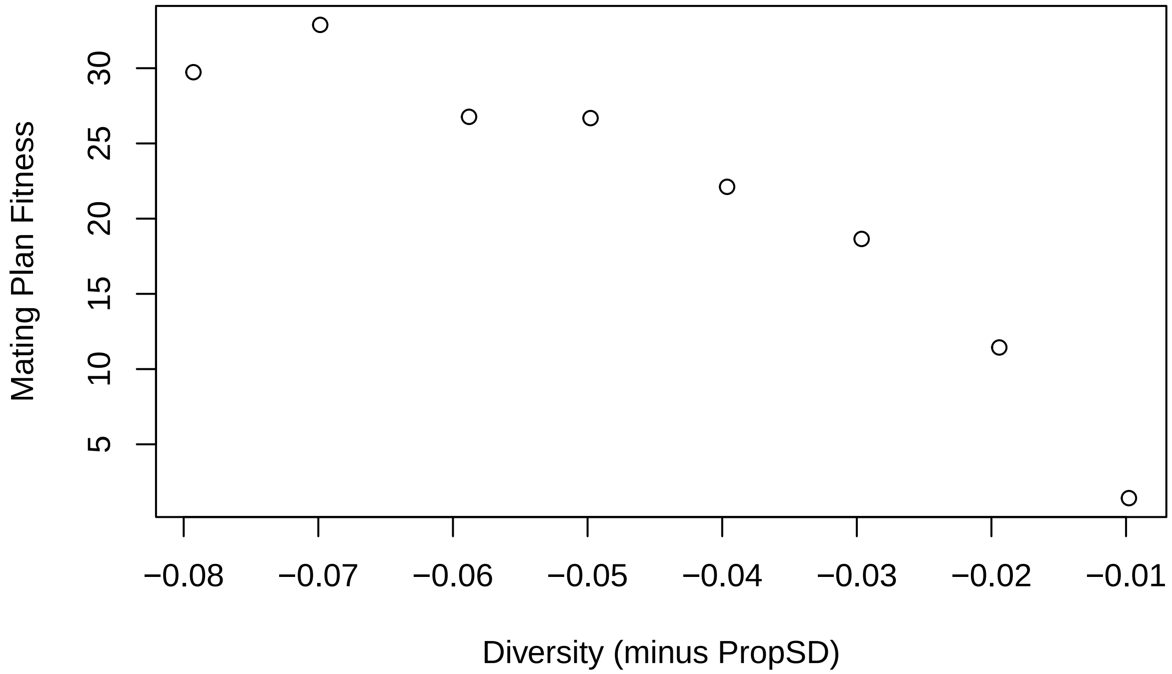
```
## $`g1_1/g1_2`
## [1] 20.90436
##
## $`g1_1/g1_3`
## [1] 16.37305
##
## $`g1_1/g1_4`
## [1] 18.38269
##
## $`g1_1/g1_5`
## [1] 15.85391
##
## $`g1_1/g1_6`
## [1] 19.12954
```

```
#Test different values for the limit of diversity loss
PropSD_values <- seq(0.01, 0.08, by = 0.01)
PropSD_values
```

```
## [1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08
```

```
#Perform tests with different values of the parameters and using the precomputed
#FamilyValues to increase speed:
mating_plan_PropSD <- c()
mating_plan_fitness <- c()
for (PropSD_value in PropSD_values) {
  out <- GenomicMatingMT(Parents1 = Parents,
                         Parents2 = Parents,
                         FamilyValues = FamilyValues, #Include here precomputed values!
                         Markers = Markers,
                         parametrization = "Genotypic",
                         phi = 2, #Diploid or allopolyploid crop
                         markereffects = NULL, #marker effects for both traits
                         n = NULL,
                         #########################################################
                         PropSD = PropSD_value, #test several values for PropSD
                         #########################################################
                         size = Number_of_crosses,
                         offspring_per_cross = F4_individuals_per_cross,
                         within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                         control = control,
                         n_selected_per_family=F4_selected_per_family,
                         Username=username, #you can get one by contacting us
                         Password=password, #you can get one by contacting us
                         Username_TrainSel=username_TrainSel, #you can get one by contacting us
                         Password_TrainSel=password_TrainSel #you can get one by contacting us
  )
  mating_plan_PropSD <- c(mating_plan_PropSD, out$PropSD)
  mating_plan_fitness <- c(mating_plan_fitness, out$Fitness)
}

#Explore trade-off between diversity and gain.
#The curve is not very smooth because we are limited to the demo version of
#TrainSel. This would be solved by upgrading to the full version and leaving
#the default values for the "control" argument in the GenomicMatingMT() function
plot(x = -mating_plan_PropSD,
     y = mating_plan_fitness,
     xlab = "Diversity (minus PropSD)",
     ylab = "Mating Plan Fitness")
```



As expected, mating plans with lower diversity allow to obtain more short-term gain (higher fitness) and vice versa. From this plot, it seems that values for the proportion of additive standard deviation lost of around 0.05 (5%) are optimal in this dataset.

It is important to note that here we performed optimization 8 times using the within-family variance computed with haplotype data. If within-family variance had to be computed every single time, it would have been substantially slower.

Double Haploids

MateR also supports using double haploids (DHs). Here, we will show how we would perform optimization for a breeding program similar to the previous ones but with DHs performed on the F1 generation instead of performing selfing cycles:

```
DHs=TRUE #DHs are performed
Selfing_cycles=0 #0 selfing cycles before the DHs (i.e., the DHs are performed on the F1)
```

```
#Perform optimization to maximize yield with the desired parameters:
out <- GenomicMatingMT(Parents1 = Parents,
                       Parents2 = Parents,
                       Markers = Markers,
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       c_list = c_list,
                       LD="Approx",
                       ##########################################################
                       DHs = DHs, #We are using DHs
                       n = Selfing_cycles, #no selfing cycles
                       ##########################################################
                       markereffects = markereffectsYLD, #marker effects for desired trait
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       coefficients = NULL,
                       offspring_per_cross = F4_individuals_per_cross,
                       within_family_accuracy = sqrt(h2), #narrow sense (only additive)
                       control = control,
                       n_selected_per_family=F4_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_3/g1_15    g1_3   g1_15                 6       8.812312
## 2  g1_4/g1_20    g1_4   g1_20                 7      -1.677246
## 3  g1_6/g2_16    g1_6   g2_16                 6       6.010637
## 4 g1_11/g2_16   g1_11   g2_16                 3      13.973921
## 5 g1_14/g1_40   g1_14   g1_40                 7       9.900989
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             18.18030   26.99261
## 2                             19.43396   17.75671
## 3                             20.52441   26.53505
## 4                             17.45301   31.42694
## 5                             19.04497   28.94596
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04889045
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 2.429672e-09
```

Clonally Propagated, Autotetraploid Crop

In a **clonally propagated crop**, there is no need to get fully homozygous lines because the plant material can be **asexually propagated** without altering its genotype. As a result, breeding in this kind of crops involves crossing the available elite lines to generate new variability, selecting the best and propagating them clonally. This also makes **dominance effects relevant**, as they can improve the phenotype and **they are not lost when propagating clonally.**

Next, we will also consider that in this example the **crop is autotetraploid** to showcase how the package handles it. Finally, the **example data available for this crop does not contain directly the marker effects**. Instead, it has additive and non-additive genotypic values for each line. **We will also show how the marker effects can be extracted** from them for their use in genomic mating.

Computing Marker Effects from Breeding and Dominance Values

First, we will load the example data. **The genotypic values available** are similar to what could be obtained from the following **GBLUP model:**

$$\boldsymbol{y} = \mathbf{1}\mu + \boldsymbol{F}b + Z_a\boldsymbol{a} + Z_d\boldsymbol{d} + \boldsymbol{\epsilon}$$

Where $\boldsymbol{y}$ is a vector of phenotypes, μ is a fixed intercept, $\boldsymbol{F}$ is a vector of inbreeding coefficients (computed as the diagonal values of the VanRaden genomic relationship matrix minus one), b is a regression coefficient indicating the inbreeding effect (directional dominance), $\boldsymbol{a} \sim N(\mathbf{0}, G\sigma_a^2)$ is a vector of additive breeding values, $\boldsymbol{d} \sim N(\mathbf{0}, D\sigma_d^2)$ is a vector of dominance values and $\boldsymbol{\epsilon} \sim N(\mathbf{0}, I\sigma_\epsilon^2)$ is a vector of residuals. $\mathbf{1}$, Z_a and Z_d are design matrices for their corresponding effects. G is an additive relationship matrix, D is a dominance relationship matrix and I is the identity matrix with the needed dimensions. σ_a^2 and σ_d^2 are variance components estimated by the model.

```r
rm(list = ls()) #remove everything from the R environment
```

```r
data(ExampleDataAutotetraploid)

#1) There are 100 parental lines
length(Parents4)
```

```
## [1] 100
```

```r
Parents4[1:5]
```

```
## [1] "l_1" "l_2" "l_3" "l_4" "l_5"
```

```r
#2) Genotypic information of the parents
#marker matrix counting the number of times the alternative allele is present in each locus
Markers4[1:5,1:5]
```

```
##     SNP1 SNP2 SNP3 SNP4 SNP5
## l_1    1    2    2    2    1
## l_2    2    0    0    0    0
## l_3    0    1    2    2    2
## l_4    2    1    1    0    0
## l_5    1    2    1    1    0
```

```r
#3) Additive breeding values (a) for two traits, yield (YLD) and maturity (MAT):
AVs4$YLD[1:5] #we will need to convert these into marker effects
```

```
##      l_1      l_2      l_3      l_4      l_5
## 22.43147 86.02203 17.25468 10.52742 40.14012
```

```r
AVs4$MAT[1:5] #we will need to convert these into marker effects
```

```
##        l_1        l_2        l_3        l_4        l_5
##   4.948174  67.428844 -55.446234  12.011615  40.066247
```

```r
#4) Dominance values (d) for two traits, yield (YLD) and maturity (MAT):
DVs4$YLD[1:5] #we will need to convert these into marker effects
```

```
##        l_1        l_2        l_3        l_4        l_5
## -30.272322   1.998564 -41.930402   2.757924  61.560330
```

```
DVs4$MAT[1:5] #we will need to convert these into marker effects
```

```
##        l_1        l_2        l_3        l_4        l_5 
##   5.997176  20.190428 -52.055431  -2.850171  -1.553687
```

```
#5) Directional dominance (inbreeding effect):
G4[1:5,1:5] #VanRaden relationship matrix
```

```
##               l_1          l_2          l_3           l_4         l_5
## l_1  9.094108e-01 -0.076627398 -0.078758041  6.289246e-05  0.03926158
## l_2 -7.662740e-02  0.967862276  0.004375518 -5.670718e-02  0.26614932
## l_3 -7.875804e-02  0.004375518  0.898141496 -7.680709e-02 -0.01450505
## l_4  6.289246e-05 -0.056707176 -0.076807091  9.004775e-01 -0.01782937
## l_5  3.926158e-02  0.266149323 -0.014505054 -1.782937e-02  0.94421985
```

```
#Extract inbreeding coefficients (F) from the VanRaden relationship matrix
Fcoeff <- diag(G4)-1

#inbreeding effect (b) for yield.
#As it is negative, the crop presents inbreeding depression and heterosis
b4$YLD
```

```
## [1] -2
```

```
#Directional dominance values for yield:
(Fcoeff*b4$YLD)[1:5]
```

```
##        l_1        l_2        l_3        l_4        l_5 
## 0.18117841 0.06427545 0.20371701 0.19904500 0.11156031
```

```
#inbreeding effect (b) for maturity.
#As it is negative, the crop presents inbreeding depression and heterosis
b4$MAT
```

```
## [1] -1
```

```
#Directional dominance values for yield:
(Fcoeff*b4$MAT)[1:5]
```

```
##        l_1        l_2        l_3        l_4        l_5 
## 0.09058920 0.03213772 0.10185850 0.09952250 0.05578015
```

```
#6) Coefficients for a multi-trait selection index
coefficients
```

```
##        YLD        MAT 
##  0.9594875 -0.2817511
```

```
#7) List of Matrices of frequencies of recombination per chromosome
c_list4[[1]][1:5,1:5]
```

```
##            SNP1       SNP2       SNP3       SNP4       SNP5
## SNP1 0.00000000 0.01492425 0.02940303 0.04344964 0.05707698
## SNP2 0.01492425 0.00000000 0.01492425 0.02940303 0.04344964
## SNP3 0.02940303 0.01492425 0.00000000 0.01492425 0.02940303
## SNP4 0.04344964 0.02940303 0.01492425 0.00000000 0.01492425
## SNP5 0.05707698 0.04344964 0.02940303 0.01492425 0.00000000
```

First, we will convert the additive and dominance values for the parental lines into marker effects. This can be easily done with the function `meff_from_GVs()`:

```r
##############################################################################
#additive effects
##############################################################################
meffsYLD <- meff_from_GVs(Markers4,
                          parametrization = "Genotypic",
                       phi=4, #tetraploid. If diploid, just replace 4 with 2
                       GVs = AVs4$YLD, #breeding values
                       type="additive",
                       DirectionalDominance=NULL)
```

```r
#Marker effects computed by the function allow to reconstruct the original values
max(abs(Markers4%*%meffsYLD - AVs4$YLD))
```

```
## [1] 3.268497e-13
```

```r
meffsMAT <- meff_from_GVs(Markers4,
                          parametrization = "Genotypic",
                       phi=4, #autotetraploid. If diploid, just replace 4 with 2
                       GVs = AVs4$MAT, #breeding values
                       type="additive",
                       DirectionalDominance=NULL)
```

```r
#Marker effects computed by the function allow to reconstruct the original values
max(abs(Markers4%*%meffsMAT - AVs4$MAT))
```

```
## [1] 4.831691e-13
```

```r
#Store the marker effects in the format needed for MateR:
markereffects <- list(YLD = meffsYLD,
                      MAT = meffsMAT)


##############################################################################
#dominance effects
##############################################################################
#dominance incidence matrix
#1 for heterozygous positions, 0 otherwise
#We compute this only for validation, the function meff_from_GVs computes it internally
Markers_dom <- Compute_Q(M = Markers4, phi = 4, parametrization = "Genotypic")
Markers4[1:5,1:5] #additive marker matrix
```

```
##     SNP1 SNP2 SNP3 SNP4 SNP5
## l_1    1    2    2    2    1
## l_2    2    0    0    0    0
## l_3    0    1    2    2    2
## l_4    2    1    1    0    0
## l_5    1    2    1    1    0
```

```r
Markers_dom[1:5,1:5] #dominance marker matrix
```

```
##     SNP1 SNP2 SNP3 SNP4 SNP5
## l_1    3    4    4    4    3
## l_2    4    0    0    0    0
## l_3    0    3    4    4    4
## l_4    4    3    3    0    0
## l_5    3    4    3    3    0
```

```
FbYLD <- Fcoeff*b4$YLD #directional dominance effect
meffsYLD <- meff_from_GVs(Markers4, #additive marker matrix even if type = "dominance"!
                          parametrization = "Genotypic",
                       phi=4, #autotetraploid. If diploid, just replace 4 with 2
                       GVs = DVs4$YLD, #dominance values
                       type="dominance",
                       DirectionalDominance=FbYLD)
```

```
#The markers computed combine the original dominance values plus the effect
#of the directional dominance
max(abs(Markers_dom%*%meffsYLD -
      (DVs4$YLD + FbYLD)))
```

```
## [1] 3.339551e-13
```

```
FbMAT <- Fcoeff*b4$MAT #directional dominance effect
meffsMAT <- meff_from_GVs(Markers4, #additive marker matrix even if type = "dominance"!
                          parametrization = "Genotypic",
                       phi=4, #tetraploid. If diploid, just replace 4 with 2
                       GVs = DVs4$MAT, #dominance values
                       type="dominance",
                       DirectionalDominance=FbMAT)
```

```
#The markers computed combine the original dominance values plus the effect
#of the directional dominance
max(abs(Markers_dom%*%meffsMAT -
      (DVs4$MAT + FbMAT)))
```

```
## [1] 3.268497e-13
```

```
#Store the marker effects in the format needed for MateR:
markereffects_d <- list(YLD = meffsYLD,
                      MAT = meffsMAT)
```

It is important to note that the **dominance marker effects computed here have absorbed both the main dominance values** ($\boldsymbol{d}$ in the model) **and the directional dominance** ($\boldsymbol{F}b$ in the model).

Multi-Trait Optimization

Now, we can peform the **genomic mating itself**. For the sake of simplicity, we will use **similar parameters as in the example from the self-pollinated** crop with a few exceptions: **dominance effects** will be considered and there will **not be any selfing** of the offspring. We also need to **specify that the crop is autotetraploid.**

```
#TrainSel hyperparameters. We will use the demo version to be able to run this
#quiciker. However, the demo version is very limited and will not reach
#an optimal solution. It is good for testing but good results are not guaranteed.
control = TrainSel::SetControlDefault(size="demo",
                                      verbose = F)

# #Recommended parameters (slower but converge to a much better solution):
# control = TrainSel::SetControlDefault(size="large",
#                                        complexity = "high_complexity",
#                                        verbose = F)



#limit diversity loss:
```

```
#limit of diversity loss = losing 5% of additive standard deviation
PropSD_limit <- 0.05
Number_of_crosses <- 80 #We are limited to less than 100 by the demo version!
Selfing_cycles <- 0 #No selfing in clonally propagated crops!
F1_individuals_per_cross <- 20
F1_selected_per_family <- 5
phi = 4 #4 indicates autotetraploid crop
H2 = 0.3
h2 = 0.2
```

```
#Perform optimization to maximize the index scores (multi-trait)
#Take LD into account but do not use haplotypes for computational efficiency
out <- GenomicMatingMT(Parents1 = Parents4,
                       Parents2 = Parents4,
                       Markers = Markers4,
                       parametrization = "Genotypic",
                       phi = phi, #Indicate that the crop is tetraploid!
                       LD="Approx",
                       markereffects = markereffects, #additive marker effects
                       markereffects_d = markereffects_d, #dominance marker effects
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list4,
                       coefficients = coefficients, #coefficients for index scores
                       offspring_per_cross = F1_individuals_per_cross,
                       within_family_accuracy = sqrt(H2), #broad sense (a+d)
                       control = control,
                       n_selected_per_family=F1_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##      Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  l_5/l_78     l_5    l_78                10       83.08452
## 2 l_21/l_25    l_21    l_25                 8       85.82247
## 3 l_21/l_91    l_21    l_91                 4      118.76956
## 4 l_44/l_76    l_44    l_76                 7       99.64693
## 5 l_47/l_74    l_47    l_74                 6       99.62113
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             20.05401   103.1385
## 2                             19.43693   105.2594
## 3                             17.23126   136.0008
## 4                             18.82135   118.4683
## 5                             18.39608   118.0172
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04876772
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##             [,1]
## [1,] 2.99948e-10
```

Using Haplotypes for Non-Homozygous Parents

As the parental lines are not fully homozygous in a clonally propagated crop, if we want to set `LD="Full"`, we need to manually provide the phasing information:

```
H_parents4[[1]][,1:5] #phasing information
```

```
##      SNP1 SNP2 SNP3 SNP4 SNP5
## [1,]    0    1    1    1    1
## [2,]    1    0    0    0    0
## [3,]    0    1    1    1    0
## [4,]    0    0    0    0    0
```

```
LD="Full"
```

```
#Perform optimization to maximize the index scores (multi-trait)
#Take LD into account but do not use haplotypes for computational efficiency
out <- GenomicMatingMT(Parents1 = Parents4,
                       Parents2 = Parents4,
                       Markers = Markers4,
                       parametrization = "Genotypic",
                       phi = phi, #Indicate that the crop is tetraploid!
                       #########################################################
                       LD=LD, #LD="Full" --> Use haplotype information
                       H_parents = H_parents4, #We need to manually provide haplotypes!
                       #########################################################
                       markereffects = markereffects, #additive marker effects
                       markereffects_d = markereffects_d, #dominance marker effects
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list4,
                       coefficients = coefficients, #coefficients for index scores
                       offspring_per_cross = F1_individuals_per_cross,
                       within_family_accuracy = sqrt(H2), #broad sense (a+d)
                       control = control,
                       n_selected_per_family=F1_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##       Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  l_2/l_89     l_2    l_89                 7       97.04581
## 2 l_21/l_59    l_21    l_59                 6      108.18589
## 3 l_42/l_85    l_42    l_85                10       89.26452
## 4 l_44/l_91    l_44    l_91                 5      118.60955
## 5 l_48/l_76    l_48    l_76                 8      116.10491
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             17.71953   114.7653
## 2                             16.75910   124.9450
## 3                             18.91322   108.1777
## 4                             15.41652   134.0261
## 5                             18.33611   134.4410
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04929062
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 4.824773e-10
```

Hybrid Crop

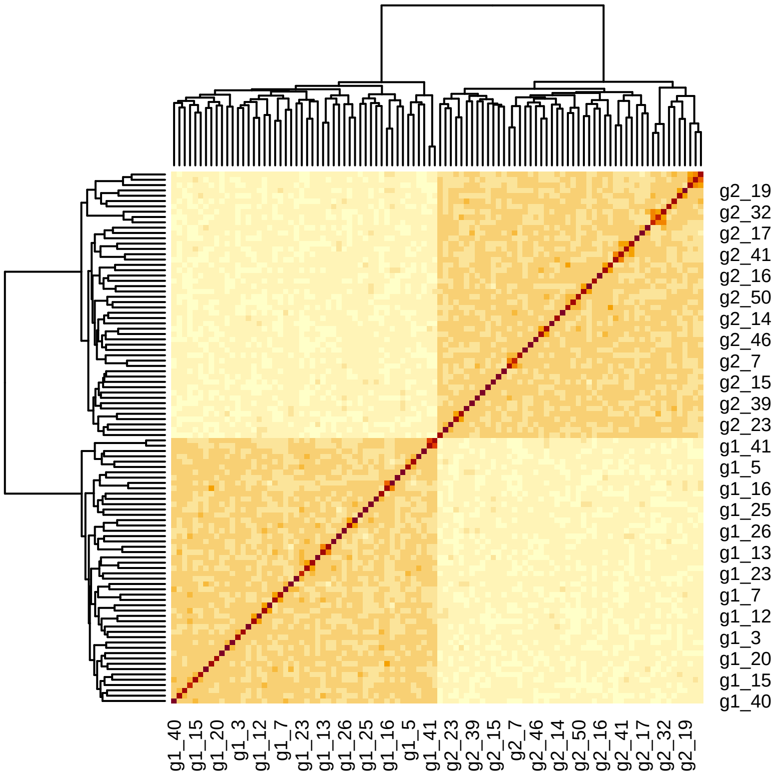
In **hybrids**, fully inbred **parental lines from complementary heterotic groups** are crossed to generate **highly heterozygous hybrids that display heterosis**, enhancing their performance. As such, considering **dominance effects is critical** for hybrids. For this example, we will go back to the diploid dataset. Actually, this dataset has two distinct subpopulations, which allows us to divide it into two different heterotic groups. This will allow us to showcase two ways to leverage genomic mating for hybrid breeding.

```
rm(list = ls()) #remove everything from the R environment
```

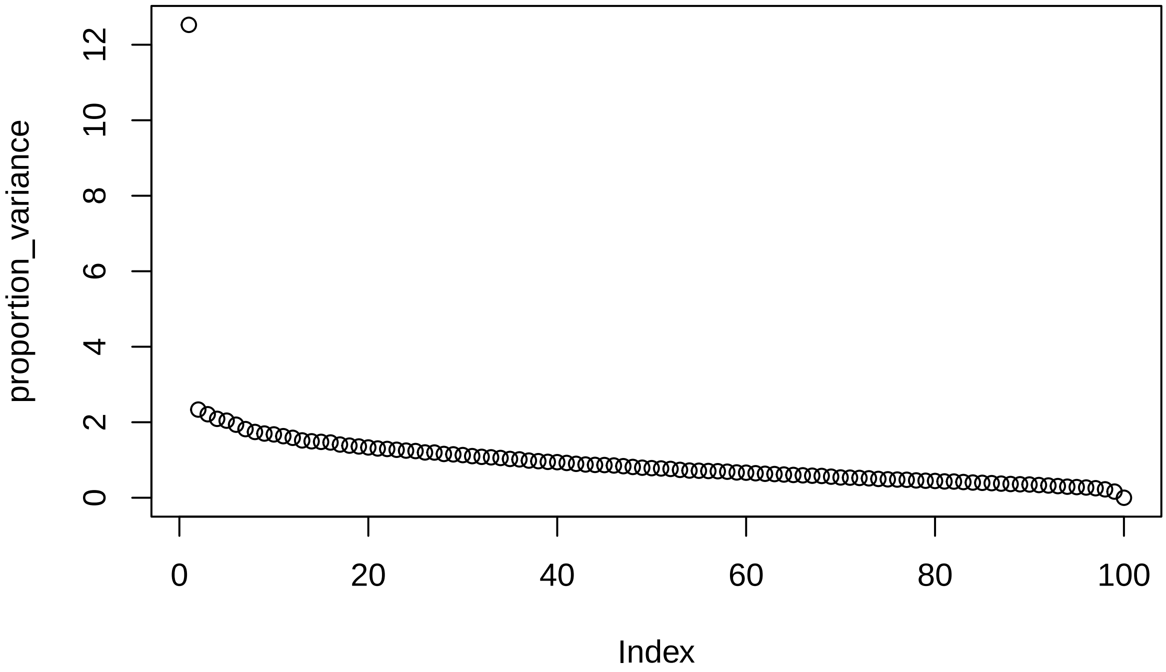
```
#Load the example data
data(ExampleDataDiploid)



#Explore population structure:
#Check population structure in the overall population
#We can clearly see two subpopulations:
#The first 50 lines have stronger relationships among themselves than with the
#remaining 50 and vice-versa
heatmap(G)
```



```
#get proportion of variance explained by the first principal components
#The first principal component explains a lot of variance, around 20% of the total!
#That confirms that the population structure is strong
eigenvalues <-eigen(G)$values
proportion_variance <- (eigenvalues/sum(eigenvalues))*100
plot(proportion_variance)
```



From the description of the dataset (available by typing `?ExampleDataDiploid` in R), we know that the individuals in the first group have the names “g1_1” through “g1_50” and the second group has the names “g2_1” through “g2_50”. We will divide the data into two heterotic groups along these lines.

```
#1) There are 100 parental lines divided into two groups.
#They can be seen as two heterotic groups
Parents_g1 <- paste0("g1_",1:50) #First heterotic group
Parents_g2 <- paste0("g2_",1:50) #Second heterotic group
Parents_g1[1:5]
```

```
## [1] "g1_1" "g1_2" "g1_3" "g1_4" "g1_5"
```

```
Parents_g2[1:5]
```

```
## [1] "g2_1" "g2_2" "g2_3" "g2_4" "g2_5"
```

```
#2) Genotypic information of the parents
#marker matrix counting the number of times the alternative allele is present in each locus
Markers_g1 <- Markers[Parents_g1,]
Markers_g2 <- Markers[Parents_g2,]
rm(Markers)

#3) Additive marker effects for two traits, yield (YLD) and maturity (MAT):
markereffects$YLD[1:5]
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
##  0.2288611 -1.0132430 -0.7770513 -0.4573172 -1.2113599
```

```
markereffects$MAT[1:5]
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5 
##  1.6079669  1.1523572 -0.2692306  0.2806039  1.4864272
```

```
#4) Dominance marker effects for two traits, yield (YLD) and maturity (MAT):
markereffects_d$YLD[1:5]
```

```
##      SNP1      SNP2      SNP3      SNP4      SNP5 
## 0.9850114 0.4712876 1.3134464 1.1436437 1.3044140
```

```
markereffects_d$MAT[1:5]
```

```
##        SNP1        SNP2        SNP3        SNP4        SNP5 
##  1.59984818  0.02891102  1.26850629  1.07151226 -0.08323753
```

```
#5) Coefficients for a multi-trait selection index
coefficients
```

```
##        YLD        MAT 
##  0.9594875 -0.2817511
```

```
#6) Haplotypes and recombination matrix
#Store separately the haplotypes of the two heterotic groups
H_Parents_g1 <- H_parents[Parents_g1]
H_Parents_g2 <- H_parents[Parents_g2]
rm(H_parents)
c_list[[1]][1:5,1:5]
```

```
##            SNP1       SNP2       SNP3       SNP4       SNP5
## SNP1 0.00000000 0.01492425 0.02940303 0.04344964 0.05707698
## SNP2 0.01492425 0.00000000 0.01492425 0.02940303 0.04344964
## SNP3 0.02940303 0.01492425 0.00000000 0.01492425 0.02940303
## SNP4 0.04344964 0.02940303 0.01492425 0.00000000 0.01492425
## SNP5 0.05707698 0.04344964 0.02940303 0.01492425 0.00000000
```

Optimal Crosses Between Heterotic Pools for Hybrid Generation

In this example, we want to **find the best possible crosses to perform between parents from the two groups** to obtain **hybrids with maximum performance**. As in the hybrids heterozygosity and low inbreeding is desired, **no selfing** is performed after crossing the parents. Furthermore, **dominance effects** have to be included to account specific combining ability and heterosis. Finally, as hybrids are a **F1 from two fully homozygous parents**, they are completely **homogeneous** and their **within-family variance is zero**. This has two implications:

1. Performing several crosses for the same family does not result in any advantage for maximizing usefulness. Thus, we have to **disallow making several crosses per family.**
2. Considering **LD** is only required for **computing within-family variance** and including it **would increase computational burden**. As we know that within-family **variance will be zero** for all possible F1 crosses, **we can completely disregard LD** to minimize the computational time without loss of accuracy.

```
#TrainSel hyperparameters. We will use the demo version to be able to run this
#quiciker. However, the demo version is very limited and will not reach
#an optimal solution. It is good for testing but good results are not guaranteed.
control = TrainSel::SetControlDefault(size="demo",
                                      verbose = F)
```

```
# #Recommended parameters (slower but converge to a much better solution):
# control = TrainSel::SetControlDefault(size="large",
#                                       complexity = "high_complexity",
#                                       verbose = F)


#When possible, keep the same parameters as for previous examples
#limit diversity loss:
#limit of diversity loss = losing 5% of additive standard deviation
PropSD_limit <- 0.05
Number_of_crosses <- 80 #We are limited to less than 100 by the demo version!
Selfing_cycles <- 0 #No selfing for hybrids!
F1_individuals_per_cross <- 20
F1_selected_per_family <- 5
Several_Crosses_per_family <- FALSE #Do not cross a pair of parents more than once
h2 = 0.2
H2 = 0.3
```

```
LD="Ind" #within family variance will always be zero --> use the fastest method

#Perform optimization to maximize the index scores (multi-trait)
#Disregard LD for computational efficiency. As within-family variance will always
#be zero, this has no impact on performance
out <- GenomicMatingMT(Parents1 = Parents_g1, #heterotic pool 1
                       Parents2 = Parents_g2, #heterotic pool 2
                       Markers = rbind(Markers_g1, Markers_g2),
                       parametrization = "Genotypic",
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffects, #additive marker effects
                       markereffects_d = markereffects_d, #dominance marker effects
                       n = Selfing_cycles,
                       ########################################################
                       LD=LD, #LD="Ind" --> Disregard LD completely
                       ########################################################
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       coefficients = coefficients, #coefficients for index scores
                       offspring_per_cross = F1_individuals_per_cross,
                       within_family_accuracy = sqrt(H2), #broad sense (a+d)
                       control = control,
                       n_selected_per_family=F1_selected_per_family,
                       ########################################################
                       replication = Several_Crosses_per_family, #Only 1 cross per family
                       ########################################################
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##       Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1 g1_2/g2_16    g1_2   g2_16                 1      -8.525958
## 2 g1_2/g2_28    g1_2   g2_28                 1     -22.308197
## 3 g1_3/g2_49    g1_3   g2_49                 1      -5.158148
## 4  g1_4/g2_4    g1_4    g2_4                 1     -12.578882
## 5 g1_4/g2_12    g1_4   g2_12                 1      -6.331059
##   Deviation_From_Average_Selected_Best Usefulness
## 1                                    0  -8.525958
## 2                                    0 -22.308197
## 3                                    0  -5.158148
## 4                                    0 -12.578882
## 5                                    0  -6.331059
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.02212686
```

```
#The value above is very similar to the desired limit
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
out$alpha
```

```
##              [,1]
## [1,] 3.768558e-09
```

The column "Deviation_From_Average_Selected_Best" always has a value of zero for all families because they are a homogenoeus F1. All crosses are performed only once because we disallowed repeating them.

Regarding the parents of each family, in the function `GenomicMatingMT()`, we specified two different pools of parents. Therefore, the two parents of a family will always come from different pools. As seen in the output, the first parent will always belong to the first heterotic group (lines "g1_1" through "g1_50") and the second parent will belong to the second heterotic pool (lines "g2_1" through "g2_50").

Breeding Within a Heterotic Pool Given its Complementary Heterotic Pool

When breeding parental lines **within a heterotic pool**, crosses are made between the parental lines and the **offspring are selfed** to make them fully inbred, similarly to what happens in self-pollinated crops. However, the aim in this case is not maximizing the usefulness of these individuals themselves, but rather **maximizing the usefulness of the hybrids obtained when they are crossed with another heterotic pool**. To showcase how to breed within a heterotic pool through genomic mating, we will use the following scheme as an example:

1. **Crosses** are made between the parents of the first **heterotic pool**.
2. The resulting **F1 offspring are selfed to increase homozygosity**.
3. For instance, after **3 cycles of selfing (F4)**, the families will be **crossed with testers** from the second heterotic pool to **generate hybrids**. The **families will be evaluated according to the usefulness of the hybrids they produce.**

In summary, we will breed within the first heterotic pool, using the second one as testers:

```
head(Parents_g1) #heterotic group 1
```

```
## [1] "g1_1" "g1_2" "g1_3" "g1_4" "g1_5" "g1_6"
```

```
head(Parents_g2) #heterotic group 2 will be used as testers
```

```
## [1] "g2_1" "g2_2" "g2_3" "g2_4" "g2_5" "g2_6"
```

```
#Please note that generally it is advised to not use many testers for
#computational reasons.
```

The `GenomicMatingMT()` function allows to **include genotypic information about the testers** that will be used. As a result, a mating plan that maximizes the usefulness of the hybrids can be generated:

```
#Keep all parameters as before with a few exceptions:
Selfing_cycles <- 3 #S3 cycles of selfing before crossing with testers

#Please, note that the TestersApproach argument allows to control how the information
#of the testers is used when computing the usefulness of each cross:

# TestersApproach <- "max" #the usefulness of each family will be the one it obtains
# #when crossed with the tester most suitable for it

TestersApproach <- "average" #the usefulness of each family will be the average
#usefulness obtained when crossed with all available testers. This is the default value
```

```
#Perform optimization to maximize the index scores (multi-trait)
#Breed within the heterotic group comprised of the lines in "Parents_g1"
#Take LD into account but do not use haplotypes for computational efficiency (default)
out <- GenomicMatingMT(Parents1 = Parents_g1, #heterotic pool 1
                       Parents2 = Parents_g1, #heterotic pool 1.
                       Markers = Markers_g1,
                       parametrization = "Genotypic",
                       ##########################################################
                       Markers_T = Markers_g2, #Marker matrix of the testers!
                       TestersApproach = TestersApproach, #either "average" or "max"
                       LD = "Approx", #faster than LD="Full"
                       ##########################################################
                       phi = 2, #Diploid or allopolyploid crop
                       markereffects = markereffects, #additive marker effects
                       markereffects_d = markereffects_d, #dominance marker effects
                       n = Selfing_cycles,
                       PropSD = PropSD_limit,
                       size = Number_of_crosses,
                       c_list = c_list,
                       coefficients = coefficients, #coefficients for index scores
                       offspring_per_cross = F1_individuals_per_cross,
                       within_family_accuracy = sqrt(H2), #broad sense (a+d)
                       control = control,
                       n_selected_per_family=F1_selected_per_family,
                       Username=username, #you can get one by contacting us
                       Password=password, #you can get one by contacting us
                       Username_TrainSel=username_TrainSel, #you can get one by contacting us
                       Password_TrainSel=password_TrainSel #you can get one by contacting us
)
```

```
#output
out$OptimalMatingScheme[1:5,] #summary of optimal mating scheme
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Recommended_Tester
## 1  g1_4/g1_15    g1_4   g1_15                13              g2_16
## 2  g1_7/g1_14    g1_7   g1_14                 3              g2_49
## 3  g1_7/g1_42    g1_7   g1_42                 2              g2_16
## 4  g1_7/g1_43    g1_7   g1_43                 2              g2_16
## 5 g1_11/g1_46   g1_11   g1_46                 6              g2_16
##   Family_average Deviation_From_Average_Selected_Best Usefulness
## 1      -38.76896                             17.11860 -21.650367
## 2      -22.43379                             13.73492  -8.698867
## 3      -25.22698                             10.31891 -14.908069
## 4      -23.69829                             11.86860 -11.829684
## 5      -36.19465                             16.33256 -19.862095
```

```
#diversity loss:
out$PropSD
```

```
## [1] 0.04885345
```

```
#The value above is very close to the desired limit:
PropSD_limit
```

```
## [1] 0.05
```

```
#likelihood of a solution with better gain and similar diversity than the current
#mating plan:
#out$alpha #NOTE: the stronger the heterosis, the less meaningful alpha becomes!
```

Some things to note in this scenario:

1. The **output** for the optimal mating scheme **includes the "Recommended_Tester"** column, indicating the best tester for each family.
2. The **usefulness values** in the output correspond to the **usefulness of the hybrids** resulting from crossing the F4 families with the testers.
3. In the arguments of the function `GenomicMatingMT()`, we have set `Parents1=Parents_g1` and `Parents2=Parents_g1`. This means that we are exploring which are the **best possible crosses among the full diallel of the parents** in `Parents_g1`, i.e., we are breeding within **the first heterotic group**.
4. We need to compute the within-family variance for all combinations between the testers and the full diallel of the first heterotic group. This can be **very computationally intensive**, specially **when using haplotype data**. Reducing the number of lines in the group of testers can greatly reduce the computational burden. Therefore, for large datasets, we **advise selecting only the most interesting individuals** from the second heterotic group **for their use as testers**.
5. If **DHs** are used, they can be specified following the same steps specified for the self-pollinated crops. Firstly, set `DHs = TRUE`. Secondly, set `n` to be the number of selfing cycles before the DHs, e.g., `n=0` if DHs are performed on the F1, `n=1` if DHs are performed on the F2, etc.

Post-Optimization

Actually implementing the optimal mating plans can be challenging in some breeding schemes. A good **mating scheme** involves generating a **relatively large number of families** and selecting **several offspring from them** in the **initial screening** for further characterization and selection, which is reflected by the GenomicMatingMT() function. However, **often only the best few lines among the individuals selected in the initial screening are reused as parental lines for the next generation**. Simply selecting these genotypes by **truncation selection** on the available offspring would be **problematic**. Genomic mating carefully selects families with maximum genetic diversity. Simply selecting the best performing offspring in a later stage would neglect this and **waste most of the diversity of the mating plan**. As a result, we have developed a tool we called **"post-optimization"** that allows to **select a subset of the available offspring** that i) **maximizes genetic value** and ii) has the **desired genetic diversity**.

The dataset "ExampleMatingPlan" contains an example for this. It is the output of the following process:

1. Using the datset "ExampleDataDiploid", genomic mating was performed with the GenomicMatingMT() function. We used the following parameters: Parental lines were the lines g1_1 through g1_50 (first subpopulation). Haplotypes were used to compute their family variance. 80 crosses with 20 offspring per cross were desired. Only additive marker effects were considered, using the marker effects and multi-trait coefficients from "ExampleDataDiploid" dataset. It was considered that 5 offspring per family would be selected after 3 cycles of selfing. The best 5 offspring would be selected by phenotypic selection with a heritability of 0.25 (accuracy of 0.5, the square root of heritability). The selected offspring would be subsequently genotyped, i.e., their markers are available. The cutoff for the proportion of additive standard deviation lost (`PropSD`) was set to 0.05.

2. The optimal mating plan contained 16 unique families, with varying numbers of crosses per family.

3. All offsrping from the optimal mating plan were created. We simulated both their markers and their phenotypic values following the desired heritability.

4. The initial screening was performed. The 5 individuals from each family with the highest phenotype were selected.

```
rm(list = ls()) #remove everything from the R environment
```

```
data(ExampleMatingPlan)


#1) Optimal mating plan:
GenomicMatingOtput$OptimalMatingScheme[1:5,]
```

```
##        Family Parent1 Parent2 Number_Of_Crosses Family_average
## 1  g1_4/g1_43    g1_4   g1_43                 8       2.946254
## 2 g1_14/g1_15   g1_14   g1_15                 5      13.685203
## 3 g1_14/g1_34   g1_14   g1_34                 3      15.782364
## 4 g1_14/g1_42   g1_14   g1_42                 4      17.395103
## 5 g1_14/g1_46   g1_14   g1_46                 2      15.812060
##   Deviation_From_Average_Selected_Best Usefulness
## 1                             18.37688   21.32314
## 2                             18.28990   31.97510
## 3                             13.10109   28.88345
## 4                             17.35998   34.75509
## 5                             14.27559   30.08765
```

```
#16 different families generated
length(GenomicMatingOtput$OptimalMatingScheme$Family)
```

```
## [1] 14
```

```
#80 total crosses
sum(GenomicMatingOtput$OptimalMatingScheme$Number_Of_Crosses)
```

```
## [1] 80
```

```
#2) Parental lines of the mating plan:
rownames(Markers_P)[1:5]
```

```
## [1] "g1_1" "g1_2" "g1_3" "g1_4" "g1_5"
```

```
#3) Lines that could be used as testers in a hybrid breeding program
rownames(Markers_potential_testers)[1:5]
```

```
## [1] "g2_1" "g2_2" "g2_3" "g2_4" "g2_5"
```

```
#4) Additive marker effects
markereffects$YLD[1:5]
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
##  0.2288611 -1.0132430 -0.7770513 -0.4573172 -1.2113599
```

```
markereffects$MAT[1:5]
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
##  1.6079669  1.1523572 -0.2692306  0.2806039  1.4864272
```

```
#5) Dominance marker effects
markereffects_d$YLD[1:5]
```

```
##      SNP1      SNP2      SNP3      SNP4      SNP5
## 0.9850114 0.4712876 1.3134464 1.1436437 1.3044140
```

```
markereffects_d$MAT[1:5]
```

```
##        SNP1        SNP2        SNP3        SNP4        SNP5
##  1.59984818  0.02891102  1.26850629  1.07151226 -0.08323753
```

```
#6) Multi-trait selection index coefficients
coefficients
```

```
##        YLD        MAT
##  0.9594875 -0.2817511
```

```
#7) Markers of the 6 best individuals from each family:
Markers_Offspring[1:6,1:10]
```

```
##                 SNP1 SNP2 SNP3 SNP4 SNP5 SNP6 SNP7 SNP8 SNP9 SNP10
## g1_4/g1_43.119    2    0    0    0    0    0    0    0    0     0
## g1_4/g1_43.127    2    0    0    0    0    0    0    0    0     0
## g1_4/g1_43.2      0    0    0    0    0    0    0    2    0     0
## g1_4/g1_43.144    0    0    0    0    0    0    0    2    0     0
## g1_4/g1_43.93     1    0    0    0    0    0    0    1    0     0
## g1_14/g1_15.20    2    0    0    2    2    2    2    0    0     0
```

```
#80 individuals in total (16 families times 5 individuals selected per family)
dim(Markers_Offspring)
```

```
## [1]   70 1000
```

```
#8) Link between each individual name, the family to which it belongs and its parents
SelectedKeyTable[1:6,]
```

```
##   Parent1 Parent2      Family      Offspring
## 3    g1_4   g1_43  g1_4/g1_43 g1_4/g1_43.119
## 4    g1_4   g1_43  g1_4/g1_43 g1_4/g1_43.127
## 1    g1_4   g1_43  g1_4/g1_43   g1_4/g1_43.2
## 5    g1_4   g1_43  g1_4/g1_43 g1_4/g1_43.144
## 2    g1_4   g1_43  g1_4/g1_43  g1_4/g1_43.93
## 7   g1_14   g1_15 g1_14/g1_15 g1_14/g1_15.20
```

```
#Create multi-trait marker effects for index scores:
markereffectsAll <- rep(0, ncol(Markers))
markereffectsAll_d <- rep(0, ncol(Markers))
for (trait in names(coefficients)) {
  markereffectsAll <- markereffectsAll + markereffects[[trait]]*coefficients[trait]

  if (!is.null(markereffects_d)) {
    markereffectsAll_d <- markereffectsAll_d + markereffects_d[[trait]]*coefficients[trait]
  }

}
markereffectsAll[1:5] #additive
```

```
##       SNP1       SNP2       SNP3       SNP4       SNP5
## -0.2334571 -1.2968719 -0.6697150 -0.5178506 -1.5810872
```

```
markereffectsAll_d[1:5] #dominance
```

```
##      SNP1      SNP2      SNP3      SNP4      SNP5
## 0.4943472 0.4440488 0.9028323 0.7954121 1.2750212
```

Using this dataset we will **showcase** how **post-optimization** can be performed in a **self-pollinated** crop and in a **hybrid** crop. It could also be used in a similar manner for **clonally propagated** crops. In both cases, we will assume that

Post-Optimization, self-pollinated crop

In this scenario, we will assume that, **from the available 80 offspring, we want to select the best 15 to include in the pool of parental lines** for the future generations. As we are in a **self-pollinated** crop, we are only interested in maximizing **additive breeding values** and we will ignore dominance. Also, we will limit loss of diversity to the diversity reduction that would be expected from selecting the top 25% of the available individuals and discarding the rest. In other words, the **alleles** carried by the selected **15 individuals cannot be less diverse** than the alleles present in the **top 25%**.

```
phi = 2 #diploid crop
size = 15 #we want to select 15 individuals
#limit diversity loss:
PropSD_limit <- 0.05

#TrainSel hyperparameters. We will use the demo version to be able to run this
#quiciker. However, the demo version is very limited and will not reach
#an optimal solution. It is good for testing but good results are not guaranteed.
control = TrainSel::SetControlDefault(size="demo",
                                      verbose = F)

# #Recommended parameters (slower but converge to a much better solution):
# control = TrainSel::SetControlDefault(size="large",
#                                        complexity = "high_complexity",
#                                        verbose = F)
```

```
set.seed(12)
out <- PostOpt(Markers_Cand = Markers_Offspring,
               parametrization = "Genotypic",
               phi = phi,
               size = size,
               markereffects = markereffects,
               markereffects_d = NULL, #self-pollinated --> no dominance
               coefficients = coefficients,
               PropSD = PropSD_limit,
               control = control,
               Username=username, #you can get one by contacting us
               Password=password, #you can get one by contacting us
               Username_TrainSel=username_TrainSel, #you can get one by contacting us
               Password_TrainSel=password_TrainSel) #you can get one by contacting us)
```

```
PropSD_limit #desired limit to diversity loss
```

```
## [1] 0.05
```

```
out$PropSD #below the desired cutoff
```

```
## [1] 0.04754456
```

```
#Selected individuals
out$OptimalSelection #the Fitness column is simply the breeding value
```

```
##         Genotypes  Fitness
## 1  g1_14/g1_15.20 60.79553
## 2  g1_14/g1_15.13 61.74161
## 3  g1_14/g1_34.54 49.25167
## 4  g1_14/g1_42.43 51.25888
## 5  g1_14/g1_42.41 51.05556
## 6  g1_14/g1_42.57 59.96783
## 7   g1_14/g1_46.5 52.78535
## 8  g1_14/g1_46.28 56.40305
## 9   g1_34/g1_43.7 46.23143
## 10 g1_34/g1_43.62 46.83346
## 11 g1_34/g1_43.39 48.48654
## 12 g1_39/g1_42.65 50.37106
## 13 g1_41/g1_46.17 46.91099
## 14 g1_41/g1_46.88 52.91840
## 15 g1_42/g1_46.42 49.43964
```

```
#The selected individuals are much better than the initial parents
#1) Calculate breeding values
BVsParents <- c(Markers_P%*%markereffectsAll)
BVsOpt <- c(Markers_Offspring[out$OptimalSelection$Genotypes,]%*%markereffectsAll)

#Original parental population
mean(BVsParents) #average
```

```
## [1] -29.94445
```

```
max(BVsParents) #best
```

```
## [1] 18.37439
```

```
#Selected Offspring
mean(BVsOpt) #much larger than parental average
```

```
## [1] 52.29673
```

```
max(BVsOpt) #much better than the best parent
```

```
## [1] 61.74161
```

Selecting the offspring to include in the parental population this way has a few **advantages over truncation selection**. Firstly, the individuals selected by **truncation selection** will all **perform very well**, but **most will be redundant** as they are likely to be **extremely similar to one another**. Secondly, our **estimated marker effects** and breeding values can have **some error**. Selecting **only the best individuals** according to them is extremely **risky**, as it is possible that what we are selecting are not the the individuals with the actual best true breeding values. Using **optimization** to select a more **diverse** set of individuals mitigates this risk, as a **better screening** of the available genotypes is performed.

Post-Optimization, hybrid crop

Here, we will use the same parameters as before, but for selecting the best 15 offspring to use as parents, we will **evaluate** them according to the expected **performance of the hybrids they generate when crossed with testers**. As a result, we need to take **dominance effects** into account.

```
phi = 2 #diploid crop
size = 15 #we want to select 10 individuals
#limit diversity loss:
PropSD_limit <- 0.05
#we want to maximize the average genotypic values of the hybrid generated by the
#selected individuals. Thus, TestersApproach = "average". It would also be possible
#to set TestersApproach = "max" to select individuals according to the genotypic
#value of the best hybrid they generate.
TestersApproach = "average"

#TrainSel hyperparameters. We will use the demo version to be able to run this
#quiciker. However, the demo version is very limited and will not reach
#an optimal solution. It is good for testing but good results are not guaranteed.
control = TrainSel::SetControlDefault(size="demo",
                                      verbose = F)

# #Recommended parameters (slower but converge to a much better solution):
# control = TrainSel::SetControlDefault(size="large",
#                                         complexity = "high_complexity",
#                                         verbose = F)
```

```
set.seed(12)
out <- PostOpt(#Markers_P = Markers_P, #if desired, a reference population can be used
               Markers_Cand = Markers_Offspring,
               Markers_T = Markers_potential_testers,
               parametrization = "Genotypic",
               TestersApproach = TestersApproach,
               phi = phi,
               size = size,
               markereffects = markereffects,
               markereffects_d = markereffects_d, #hybrid --> we need dominance
               coefficients = coefficients,
               PropSD = PropSD_limit,
               control = control,
               Username=username, #you can get one by contacting us
               Password=password, #you can get one by contacting us
               Username_TrainSel=username_TrainSel, #you can get one by contacting us
               Password_TrainSel=password_TrainSel)
```

```
PropSD_limit #desired limit to diversity loss
```

```
## [1] 0.05
```

```
out$PropSD #below the desired cutoff
```

```
## [1] 0.04264921
```

```
#Selected individuals
#The negative fitness of the selected lines can be explained because, in this
#example dataset, the specific combining ability with the testers was very poor.
out$OptimalSelection
```

```
##          Genotypes Recommended_Tester    Fitness
## 1     g1_4/g1_43.2              g2_35  1.2036184
## 2   g1_14/g1_15.37              g2_49 -1.0884335
## 3   g1_14/g1_34.54              g2_36  2.3621168
## 4   g1_14/g1_42.43              g2_40  2.1385734
## 5   g1_14/g1_42.41               g2_8 17.0946115
## 6   g1_14/g1_42.57               g2_7 17.4829581
## 7   g1_14/g1_42.25              g2_34  0.2849694
## 8    g1_14/g1_46.5              g2_21  1.1034215
## 9   g1_14/g1_46.28              g2_49 -0.8678944
## 10  g1_15/g1_43.36              g2_13  1.8535998
## 11 g1_26/g1_46.128               g2_2 -0.7544374
## 12  g1_34/g1_42.41              g2_28 -1.5009414
## 13  g1_39/g1_42.65              g2_26  0.9571335
## 14  g1_42/g1_46.42              g2_47 -1.5707835
## 15  g1_42/g1_46.67              g2_14  3.8890750
```

```
#The selected individuals are much better than the initial parents
#1) Calculate breeding values (General Combining Ability)
BVsParents <- c(Markers_P%*%markereffectsAll)
BVsOpt <- c(Markers_Offspring[out$OptimalSelection$Genotypes,]%*%markereffectsAll)

#Original parental population (General Combining Ability)
mean(BVsParents) #average
```

```
## [1] -29.94445
```

```
max(BVsParents) #best
```

```
## [1] 18.37439
```

```
#Selected Offspring (General Combining Ability)
mean(BVsOpt) #much larger than parental average
```

```
## [1] 47.09218
```

```
max(BVsOpt) #much better than the best parent
```

```
## [1] 59.96783
```

The output of the optimization is similar to what we had in the self-pollinated crop with a few differences. For hybrids, we have an additional column with the **recommended tester**, i.e., the tester that is expected to generate the best hybrid when crossed with the selected genotype. Furthermore, the **"Fitness"** column now contains the average **genotypic value for the hybrids** that can be created by crossing each genotype with all testers (when `TestersApproach = "average"` is used). Alternatively, it can contain the genotypic value of the hybrid obtained with the recommended tester specifically if `TestersApproach = "max"` is used.

Advantages and Limitations

In this section, we will summarize some of the key characteristics that make MateR appealing. To find the theoretical background that supports our claims, please see the MateR publication. The main **advantages** of the MateR package are the following:

1. It is highly **versatile and easy to use**, accommodating for numerous breeding schemes.
2. It supports **multi-trait** analysis.
3. MateR accepts both **proportion of additve standard deviation lost** and **inbreeding rate** as **parameters controlling** the maximum acceptable **diversity loss**. The latter is the most typical one, but its computation from a genomic relationship matrix incurs in some error and it is difficult to directly link it to the rate of reduction in genetic gain. Conversely, The former has been newly developed to solve these issues and we recommend its use. In any case, both parameters rely on metrics frequently used by breeders, which facilitates the choice of its desired value by the users.
4. Highly **informative and readable output**.
5. The **usefulness of each cross is calculated integrating a lot of information**, allowing to account for **selfing**, **number of offspring** generated per cross, **additive** and **dominance** effects, directional dominance (i.e., **heterosis and inbreeding depression**), **interaction between additive and dominance** effects within a locus, **interaction between** marker effects of **correlated traits** and **linkage disequilibrium**.
6. **Diploids, allopolyploids and autotetraploids** are supported.
7. Genomic mating requires finding the best crosses among a very large set of possibilities, making optimization often very time-consuming. MateR has a **computationally efficient implementation**. Some alternatives, like convex optimization, can be substantially faster. However, convex optimization only works for a convex objective function. This could be possible if the way in which usefulness is computed is substantially simplified, but that could have negative impacts on the quality of the results.
8. **Additional utility functions** are provided to facilitate the calculation of the inputs needed for genomic mating. For instance, `meff_from_GVs()` allows to easily compute the marker effects.
9. MateR supports both the **"Genotypic"** and **"Breeding" parametrizations** of marker effects and marker scores.

However, it also has some **limitations**:

1. **Epistasis is disregarded**.
2. **Allopolyploids are treated as diploids**. Therefore, the **dominance between different subgenomes** is modeled as epistasis, which **is not considered** in MateR.
3. Calculating within-family variance can be very **computationally intensive if haplotypes are considered**. However, MateR provides much faster alternatives that can provide comparable accuracy, mitigating this issue. If computational time is a bottleneck, we generally recommend using the simplification `LD="Approx"`.
4. Autotetraploids are supported, but **other autopolyploids are not**.
5. MateR **requires license keys**, but they will be provided for **free to public bodies**, such as Universities and non-profit organizations. You can get license keys by contacting javier.fgonzalez@upm.es or j.isidro@upm.es.