Genetic algorithm using the evola package

Giovanny Covarrubias-Pazaran

2024-09-03

The evola package is nice wrapper of the AlphaSimR package that enables the use of the evolutionary algorithm (EA) to solve complex questions in a simple manner.

The vignettes aim to provide several examples in how to use the evola package under different optimization scenarios. We will spend the rest of the space providing examples for:

1) Optimizing the selection of one feature with a constraint in other
2) Obtaining a subsample of a population to maximize a feature while constraining the relationship between possibilities 2a) best parents for the next generation 2b) best crosses of the next generation
3) Optimizing a subsample of size N to be representative 3a) Of its own 3b) For another sample
4) How to specify constraints: 4a) gender 4b) number of times a parent should be used
5) How to optimize the number of progeny to produce per cross
6) Predictive model using GA

Because of CRAN requirements I will only run few generations but please when you run your analysis let it run for many generations.

1) Optimizing the selection of one feature with a constraint in other

The example presented here is a list of gems (Color) that have different weights in Kg (Weight) and a given value (Value).

```r
set.seed(1)
# Data
Gems <- data.frame(
  Color = c("Red", "Blue", "Purple", "Orange",
            "Green", "Pink", "White", "Black",
            "Yellow"),
  Weight = round(runif(9,0.5,5),2),
  Value = round(abs(rnorm(9,0,5))+0.5,2)
)
head(Gems)
```

```
##    Color Weight Value
## 1    Red   1.69  8.20
## 2   Blue   2.17  5.14
## 3 Purple   3.08  1.97
## 4 Orange   4.59  0.53
## 5  Green   1.41 12.52
## 6   Pink   4.54  4.32
```

The task to optimize here is to be able to pick in your bag all the possible gems that maximize the value with the contraint that your bag would break after 10Kg. In the evolafit function this would be specified as follows:

```
# Task: Gem selection.
# Aim: Get highest combined value.
# Restriction: Max weight of the gem combined = 10.
res0<-evolafit(cbind(Weight,Value)~Color, dt= Gems,
               # constraints: if greater than this ignore
               constraintsUB = c(10,Inf),
               # constraints: if smaller than this ignore
               constraintsLB= c(-Inf,-Inf),
               # weight the traits for the selection
               traitWeight = c(0,1),
               # population parameters
               nCrosses = 100, nProgeny = 20, recombGens = 1,
               # coancestry parameters
               A=NULL, lambda=0, nQTLperInd = 1,
               # selection parameters
               propSelBetween = .9, propSelWithin =0.9,
               nGenerations = 30, verbose = FALSE
)
```

Noticed that the formula cbind(Weight,Value)~Color specified the traits to be considered in the optimization problem are specified in the left side of the formula whereas the right side of the formula specifies the term corresponding to the genes that will form the 'genome' of the solution. Each trait in the formula requires a value for the contraints, weights in the fitness function (i.e., selection index) and lambda (weight for group relationship between the genes in the genome). The rest of the parameters are the parameters controling the evolution of the population of solutions.

When looking at the results of the evolution we can observe that the best solution for the traits under the contraints can be extracted with the bestSol() function.

```
best=bestSol(res0)["pop","Value"]; best # best solution for trait 1
```

```
## [1] 24
```

```
res0$M[best,]
```

```
##    Red   Blue Purple Orange  Green   Pink  White  Black Yellow
##      1      0      0      0      1      0      0      1      1
```

```
xa = res0$M[best,] %*% as.matrix(Gems[,c("Weight","Value")]); xa
```

```
##      Weight Value
## [1,]    9.9 28.91
```

The best selection of Gems was the one one found in the M element of the resulting object.

2) Obtaining subsample of a population to maximize a feature while constraining the relationship between possibilities

2a) Best parents for the next generation One situation that occurs in plant and animal breeding is the so called 'optimal contribution' problem where we want to pick a set of parents that can maximize the gain while managing genetic variance as much as possible. In the following example we take a population of 363 possible parents and pick the best 20 while conserving genetic variance.

```
data(DT_cpdata)
DT <- DT_cpdata
head(DT)
```

```
##        id Row Col Year      color  Yield FruitAver Firmness Rowf Colf occ
## P003 P003   3   1 2014 0.10075269 154.67     41.93  588.917    3    1   0
## P004 P004   4   1 2014 0.13891940 186.77     58.79  640.031    4    1   1
## P005 P005   5   1 2014 0.08681502  80.21     48.16  671.523    5    1   1
## P006 P006   6   1 2014 0.13408561 202.96     48.24  687.172    6    1   1
## P007 P007   7   1 2014 0.13519278 174.74     45.83  601.322    7    1   1
## P008 P008   8   1 2014 0.17406685 194.16     44.63  656.379    8    1   1
```

Our surrogate of fitness will be the Yield trait and we will have a second trait to control the number of individuals we can select. We will set a contraint for the occurrence (occ) trait to 20 but the only trait contributing to fitness will be Yield (traitWeight parameter).

```
# get best 20 individuals weighting variance by 0.5
res<-evolafit(cbind(Yield, occ)~id, dt= DT,
              # constraints: if sum is greater than this ignore
              constraintsUB = c(Inf,20),
              # constraints: if sum is smaller than this ignore
              constraintsLB= c(-Inf,-Inf),
              # weight the traits for the selection
              traitWeight = c(1,0),
              # population parameters
              nCrosses = 100, nProgeny = 10,
              # coancestry parameters
              A=A, lambda= (30*pi)/180 , nQTLperInd = 2,
              # selection parameters
              propSelBetween = 0.5, propSelWithin =0.5,
              nGenerations = 20, verbose=FALSE)
```

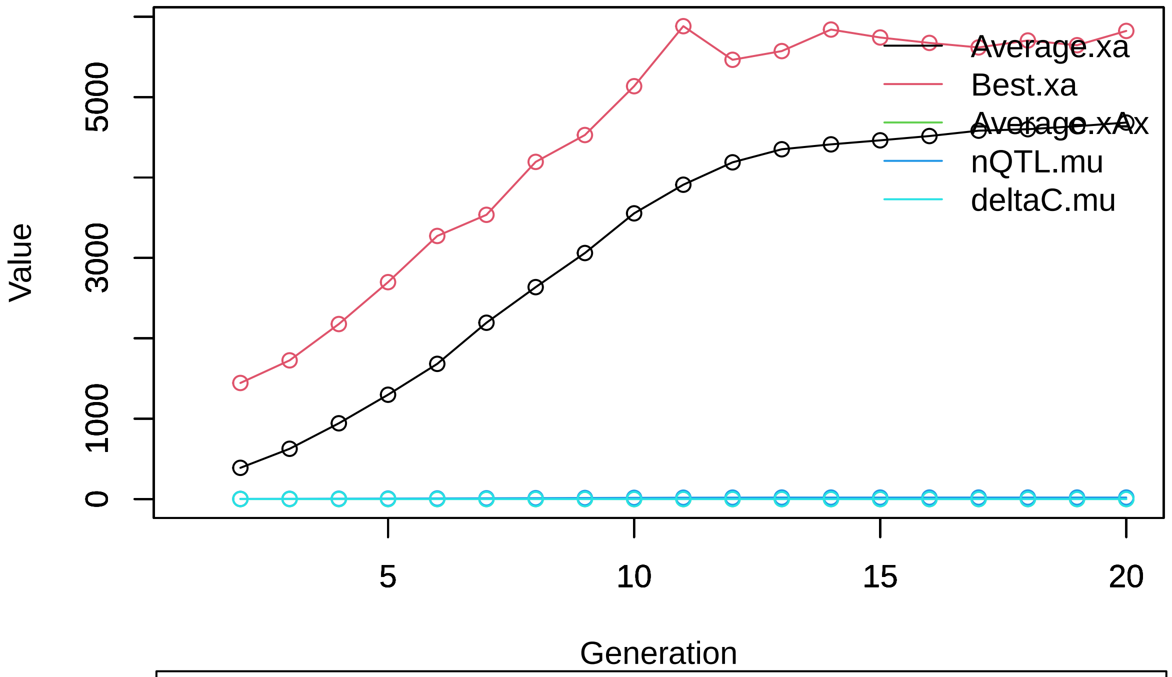
We then use the bestSol() function to extract the solution that maximizes our fitness function and constraints.

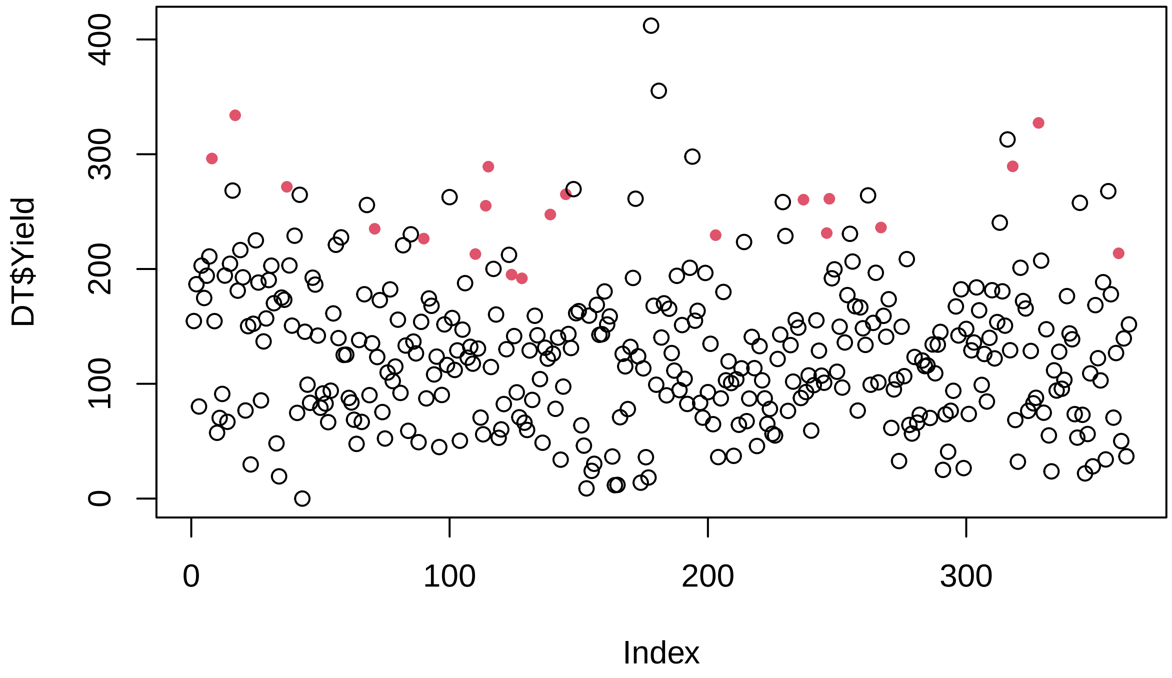
```
best = bestSol(res)["pop","Yield"];
sum(res$M[best,]) # total # of inds selected
```

```
## [1] 20
```

We can use the pmonitor() function to see if convergence was achieved between the best and teh average solutions.

```
pmonitor(res)
plot(DT$Yield, col=as.factor(res$M[best,]),
     pch=(res$M[best,]*19)+1)
```





2b) Obtaining optimal N crosses from a population for a given feature A variation of the same problem is when we want to pick the best crosses instead of the best parents to directly find the optimal solution for a crossing block. In the following example we use a dataset of crosses with marker and phenotype information to show how to optimize this problem.

```r
data(DT_technow)
DT <- DT_technow
DT$occ <- 1; DT$occ[1]=0
M <- M_technow
A <- A.mat(M)
head(DT)
```

```
##    hybrid dent flint     GY    GM      hy occ
## 1 518.298  518   298  -8.04 -0.85 518:298   0
## 2 518.305  518   305 -11.10  1.70 518:305   1
## 3 518.306  518   306 -16.85  2.24 518:306   1
## 4 518.316  518   316   2.08 -1.33 518:316   1
## 5 518.323  518   323   5.65 -2.71 518:323   1
## 6 518.327  518   327 -16.95 -0.52 518:327   1
```

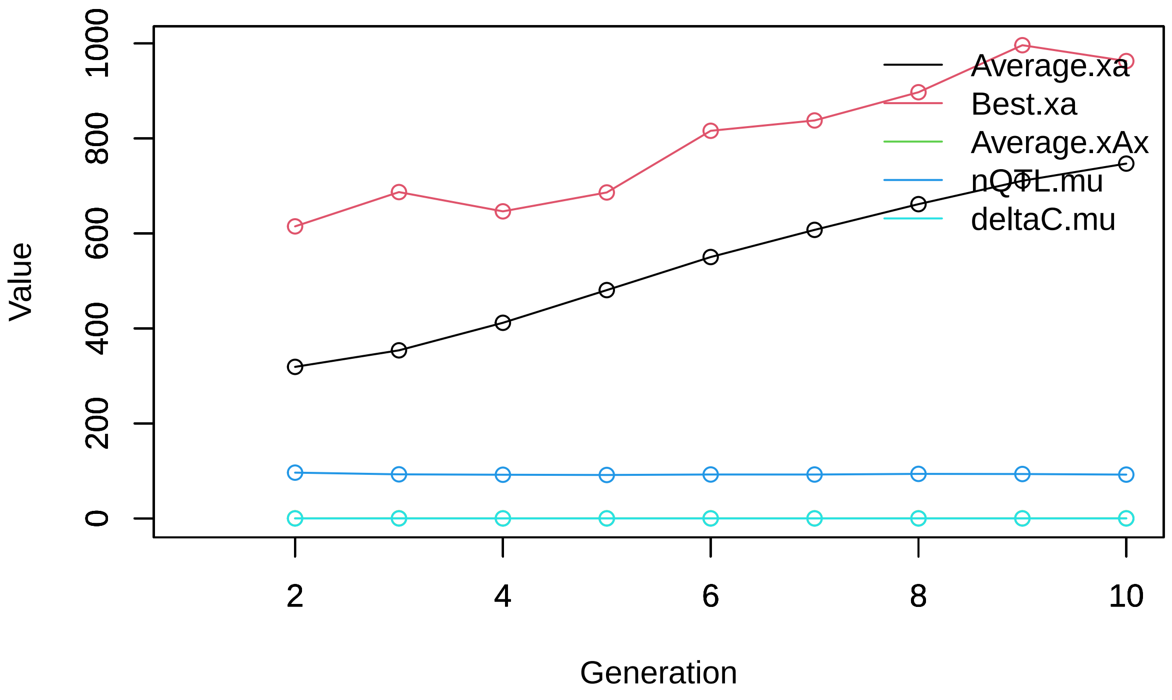
The way to specify this problem is exactly the same than with the optimization of parents but the input information is at the level of predicted crosses instead of individuals.

```
# run the genetic algorithm
  res<-evolafit(formula = c(GY, occ)~hy, dt= DT,
                # constraints: if sum is greater than this ignore
                constraintsUB = c(Inf,100),
                # constraints: if sum is smaller than this ignore
                constraintsLB= c(-Inf,-Inf),
                # weight the traits for the selection
                traitWeight = c(1,0),
                # population parameters
                nCrosses = 100, nProgeny = 10,
                # coancestry parameters
                A=A, lambda= (20*pi)/180 , nQTLperInd = 100,
                # selection parameters
                propSelBetween = 0.5, propSelWithin =0.5,
                nGenerations = 10, verbose=FALSE)
best = bestSol(res)["pop","GY"]
sum(res$M[best,]) # total # of inds selected
```

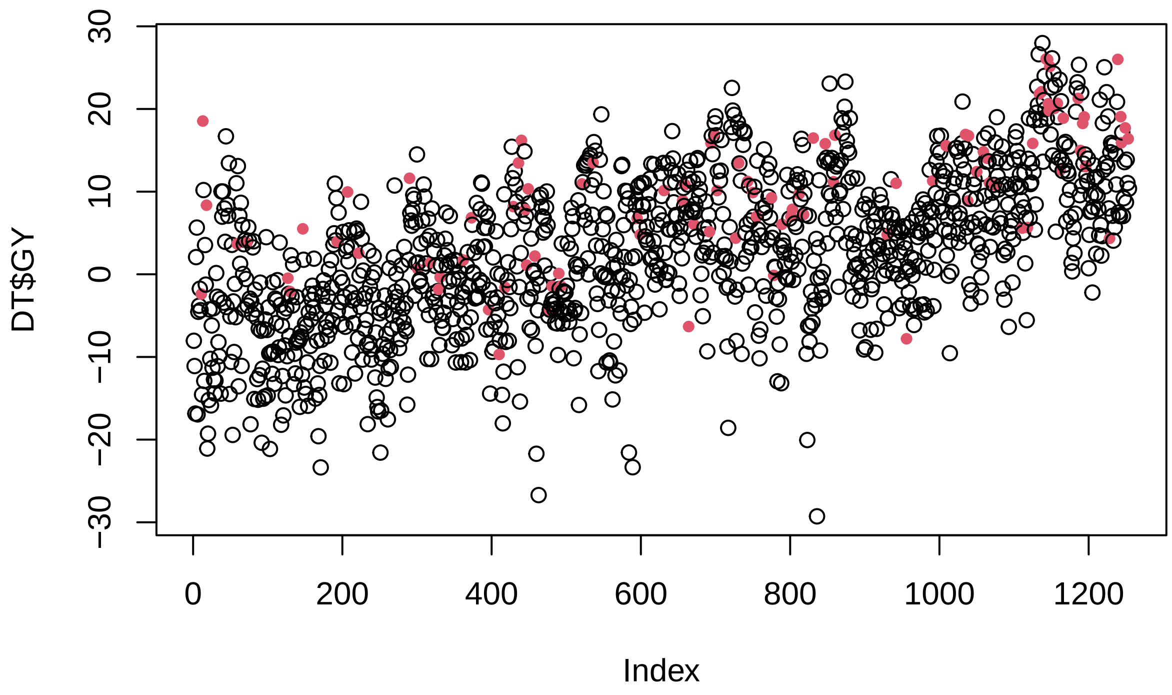
```
## [1] 99
```

You can use the pmonitor() or pareto() functions to see the evolution of the solution and see the performance of the solution selected.

```
pmonitor(res)
```



```r
plot(DT$GY, col=as.factor(res$M[best,]),
        pch=(res$M[best,]*19)+1)
```



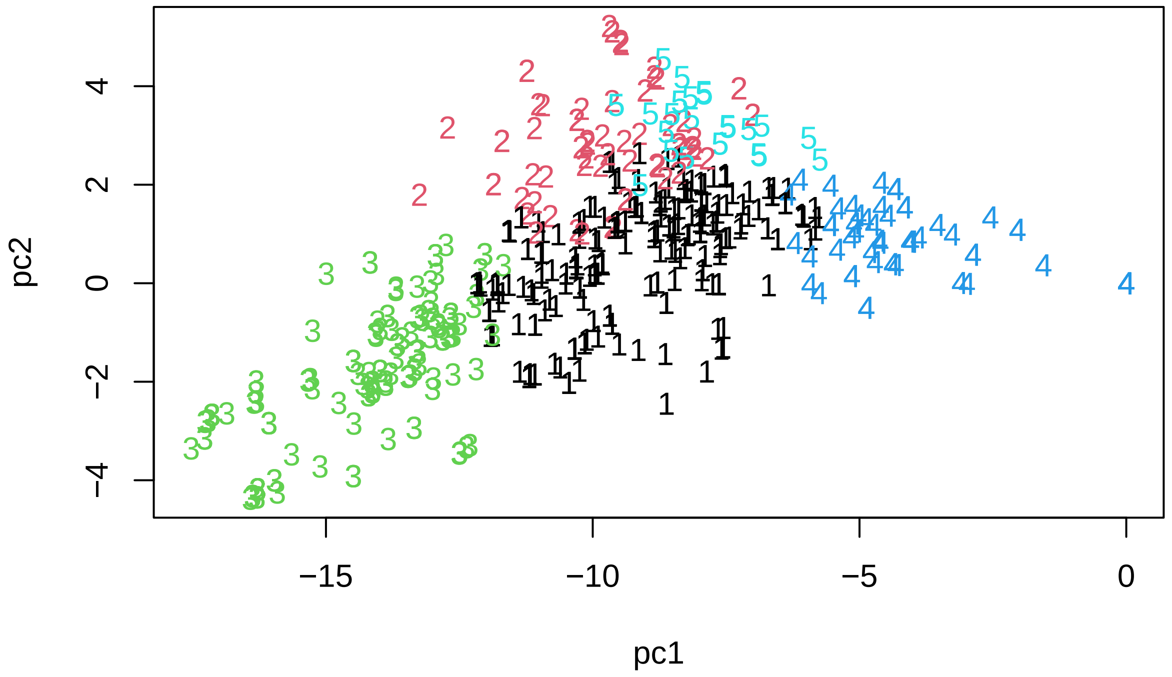
3) Optimizing a subsample of size N to be representative

One particular case when we want to pick a representative subsample is when we don't have the resources to test everything (e.g., in the field/farm). In this example we use the information from 599 wheat lines to pick a subsample that maximizes the prediction accuracy for the entire sample. We start loading the data, in particular the phenotypes (DT) and the pedigree relationship matrix (A).

```r
data(DT_wheat)
DT <- as.data.frame(DT_wheat)
DT$id <- rownames(DT) # IDs
DT$occ <- 1; DT$occ[1]=0 # to track occurrences
DT$dummy <- 1; DT$dummy[1]=0 # dummy trait
# if genomic
# GT <- GT_wheat + 1; rownames(GT) <- rownames(DT)
# A <-  GT%*%t(GT)
# A <- A/mean(diag(A))
# if pedigree
A <- A_wheat
```

Now in order to pick a structured sample we will do a PCA and pick the cluster number 3 to be a subset to predict later (vp), while we will focus in rest of the population as candidates for the training set (tp).

```r
##Perform eigenvalue decomposition for clustering
##And select cluster 5 as target set to predict
pcNum=25
svdWheat <- svd(A, nu = pcNum, nv = pcNum)
PCWheat <- A %*% svdWheat$v
rownames(PCWheat) <- rownames(A)
DistWheat <- dist(PCWheat)
TreeWheat <- cutree(hclust(DistWheat), k = 5 )
plot(PCWheat[,1], PCWheat[,2], col = TreeWheat,
     pch = as.character(TreeWheat), xlab = "pc1", ylab = "pc2")
```



```r
vp <- rownames(PCWheat)[TreeWheat == 3]; length(vp)
```

```
## [1] 159
```

```r
tp <- setdiff(rownames(PCWheat),vp)
```

3a) Optimizing a subsample of size N to be representative of its own Since the objective is to select a set of 100 lines that represent best the training set (tp) of ~400 lines we will subset a relationship matrix for that training set (As).

```
As <- A[tp,tp]
DT2 <- DT[rownames(As),]
```

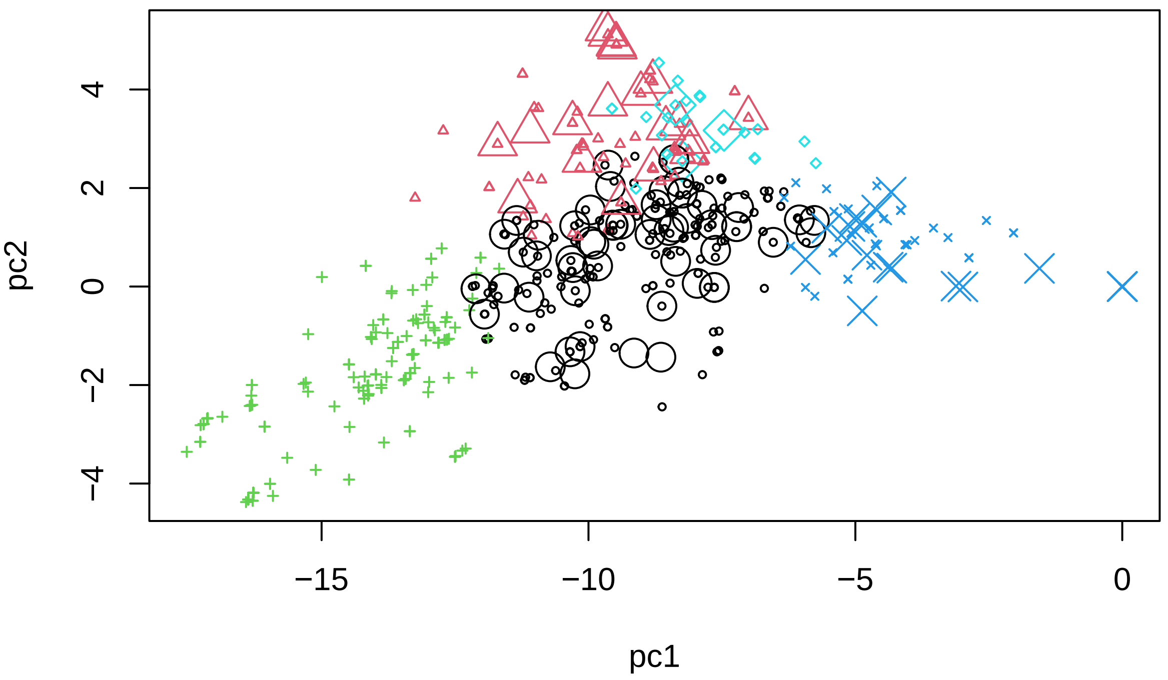
For this particular case there is no trait to optimize (x'a) but we just want to make sure that we maintain as much variation in the sample as possible (x'Ax). We then just create a dummy trait in the dataset (dummy) to put all the weight into the group relationship (x'Ax) using the lambda argument. The trait for occurrence we will use it as before to control the number of individuals to be in the sample.

```
res<-evolafit(cbind(dummy, occ)~id, dt= DT2,
                # constraints: if sum is greater than this ignore
                constraintsUB = c(Inf, 100),
                # constraints: if sum is smaller than this ignore
                constraintsLB= c(-Inf, -Inf),
                # weight the traits for the selection
                traitWeight = c(1,0),
                # population parameters
                nCrosses = 100, nProgeny = 10,
                # coancestry parameters
                A=As,
                lambda=(60*pi)/180, nQTLperInd = 80,
                # selection parameters
                propSelBetween = 0.5, propSelWithin =0.5,
                nGenerations = 15, verbose = FALSE)

best = bestSol(res)["pop","dummy"]
sum(res$M[best,]) # total # of inds selected
```

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

You can see which individuals were selected.

```
cex <- rep(0.5,nrow(PCWheat))
names(cex) <- rownames(PCWheat)
cex[names(which(res$M[best,]==1))]=2
plot(PCWheat[,1], PCWheat[,2], col = TreeWheat, cex=cex,
     pch = TreeWheat, xlab = "pc1", ylab = "pc2")
```



3b) Optimizing a subsample of size N to be representative of another population

we can use the covariance between the training population and the validation population to create a new trait (x'a) that can be used in addition to the group relationship (x'Ax).

```
DT2$cov <- apply(A[tp,vp],1,mean)
```

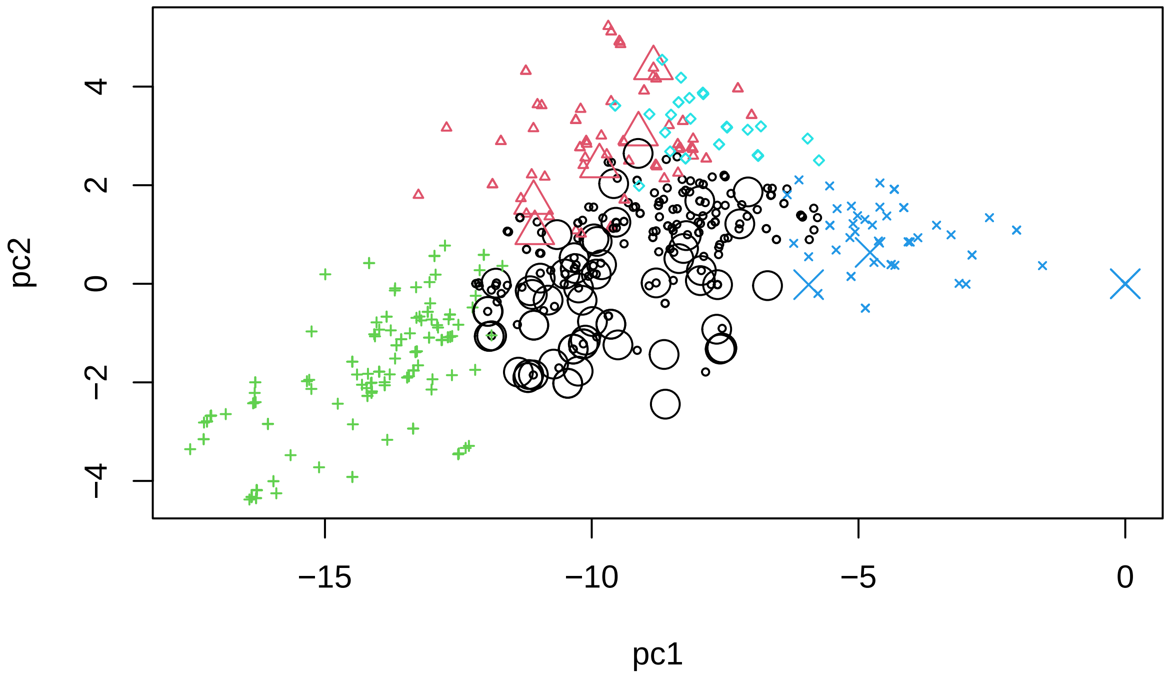
The model can be specified as before with the suttle difference that the covariance between the training and validation population contributes to the fitness function.

```
res<-evolafit(cbind(cov, occ)~id, dt= DT2,
                # constraints: if sum is greater than this ignore
                constraintsUB = c(Inf, 100),
                # constraints: if sum is smaller than this ignore
                constraintsLB= c(-Inf, -Inf),
                # weight the traits for the selection
                traitWeight = c(1,0),
                # population parameters
                nCrosses = 100, nProgeny = 10,
                # coancestry parameters
                A=As,
                lambda=(60*pi)/180, nQTLperInd = 80,
                # selection parameters
                propSelBetween = 0.5, propSelWithin =0.5,
                nGenerations = 15, verbose = FALSE)
best = bestSol(res)["pop","cov"]
sum(res$M[best,]) # total # of inds selected
```

```
## [1] 69
```

You can plot the final results and see which individuals were picked.

```r
cex <- rep(0.5,nrow(PCWheat))
names(cex) <- rownames(PCWheat)
cex[names(which(res$M[best,]==1))]=2
plot(PCWheat[,1], PCWheat[,2], col = TreeWheat, cex=cex,
     pch = TreeWheat, xlab = "pc1", ylab = "pc2")
```



6) How to specify constraints

Gender In this case is better if you only create the cross combinations that are possible (e.g., where male and female can couple) and you handed them to the evolutionary algorithm. That means, the rows of the crosses to be in the searching space only include the realistic ones.

Number of times a parent should be used In this case you can modify the fitness function to set to a low value the fitness of solutions that have used too many times the same parent. Using the DT_technow dataset this would be done the following way:

First you create an incidence matrix for parents in columns and hybrids in crosses:

```r
data(DT_technow)
DT <- DT_technow
DT$occ <- 1; DT$occ[1]=0
M <- M_technow
A <- A.mat(M)

Z=with(DT,sommer::overlay(dent,flint) )#  Matrix::sparse.model.matrix(~dent-1, data=DT)
rownames(Z) <- DT$hy # needed to link to the QTL matrix
```

the secons step is to create a new fitness function for the genetic algorithm. Our objective function to be maximized is normally of the form Yb - d, where Y is the trait values, b is the trait weights, and d is the group relationship x'Ax. We then are going to put some additional constrait that parents of the crosses can't show up more than twice. This can be done in the following way:

```r
# regular fitness function
fitnessf <-function (Y, b, d, Q, Z) {
  fit <- Y %*% b - d
  return(fit)
}
# new fitness function with constraint
fitnessf <-function (Y, b, d, Q, Z) {
  X=Q%*%Z[colnames(Q),]
  bad <- as.vector( apply(X,1, function(x){length(which(x > 5))}) )
  bad <- which(bad > 0)
  fit <- Y %*% b - d
  if(length(bad) > 0){fit[bad,1]=min(fit[,1])}
  return(fit)
}
```

Notice that we have added a matrix product Q%*%Z to see how may times each parent is used in the proposed solution of crosses. The next step would be to provide the new fitness function to the evolafit() function and the additional argument Z which is the overlay matrix formed in the first chunck of code:

```r
res<-evolafit(formula = c(GY, occ)~hy,
              dt= DT,
              # constraints: if sum is greater than this ignore
              constraintsUB = c(Inf,50),
              # constraints: if sum is smaller than this ignore
              constraintsLB= c(-Inf,-Inf),
              # weight the traits for the selection
              traitWeight = c(1,0),
              # population parameters
              nCrosses = 100, nProgeny = 10,
              # coancestry parameters
              A=A, lambda= (10*pi)/180 , nQTLperInd = 40,
              # new fitness function and additional args
              fitnessf = fitnessf, Z=Z,
              # selection parameters
              propSelBetween = 0.5, propSelWithin =0.5,
              nGenerations = 15, verbose=FALSE)

best = bestSol(res)["pop","GY"]
xa = (res$M %*% DT$GY)[best,]; xa
```

```
##  29319
## 390.07
```

```r
xAx = res$M[best,] %*% A %*% res$M[best,]; xAx
```

```
##          [,1]
## [1,] 38.13456
```

```r
sum(res$M[best,]) # total # of inds selected
```

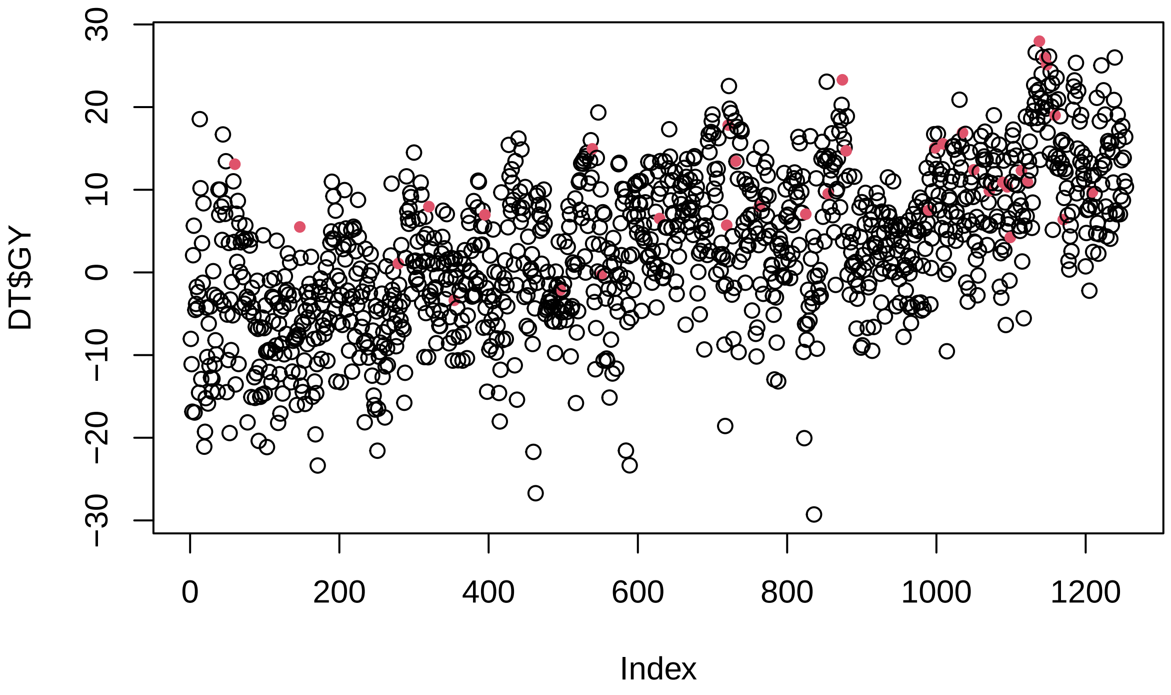
```
## [1] 35
```

Now, last but not least we check how many times each parent was used:

```r
# check how many times an individual was used in the final crosses
crosses <- data.frame(cross=names(which( res$M[best,] == 1)))
table(unlist(strsplit(crosses$cross,":")))
```

```
##
## 1863 3205 3206 3207 3208 3209 3212 3215 3216 3217 3219 3283  330  338  348  352 3605 3606 3607
##    1    1    1    1    1    1    1    1    2    2    1    1    1    1    1    1    1    1    2
## 3608 3609 3613 3615 3616 3617 3618 3619  365  377  389  392  396  397  410  430  444  457  475
##    1    1    1    2    1    2    1    1    1    1    1    3    1    4    1    1    1    1    1
##  482  490  536  560  578  586  596  604  608  632  643  647  662  671  679  714  725  742  793
##    1    1    1    1    1    1    1    1    1    1    1    2    1    1    2    1    1    1    1
##  797
##    1
```

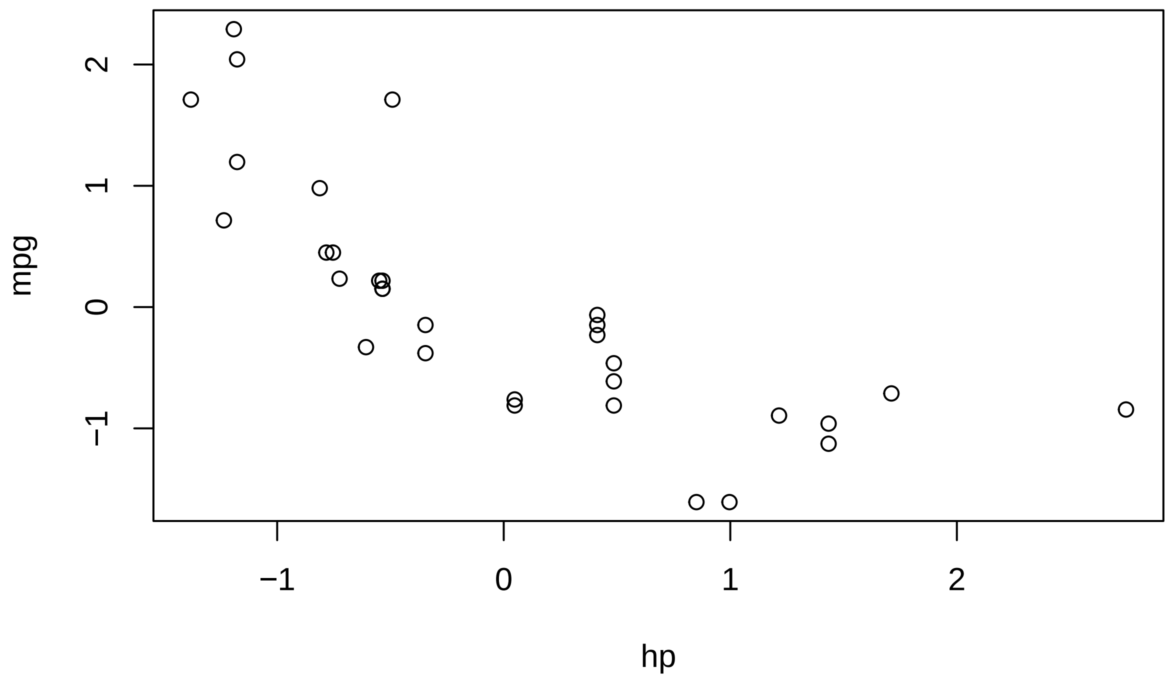
```r
# check performance of crosses selected
plot(DT$GY, col=as.factor(res$M[best,]),
       pch=(res$M[best,]*19)+1)
```



6) Predictive model using GA

The following example show how the genetic algorithm can be tweeked to do a predictive model of the type of of a linear regression.

```r
data("mtcars")
mtcars <- as.data.frame(apply(mtcars,2,scale))
mtcars$inter <- 1
# head(mtcars)
# relationship between the 2 variables
plot(mpg~hp, data=mtcars)
```



```
mod <- lm(mpg~hp, data=mtcars);mod
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Coefficients:
## (Intercept)           hp
##   6.331e-16   -7.762e-01
```

```
# create initial QTL effects
a <- seq(-1,1,.1);a
```

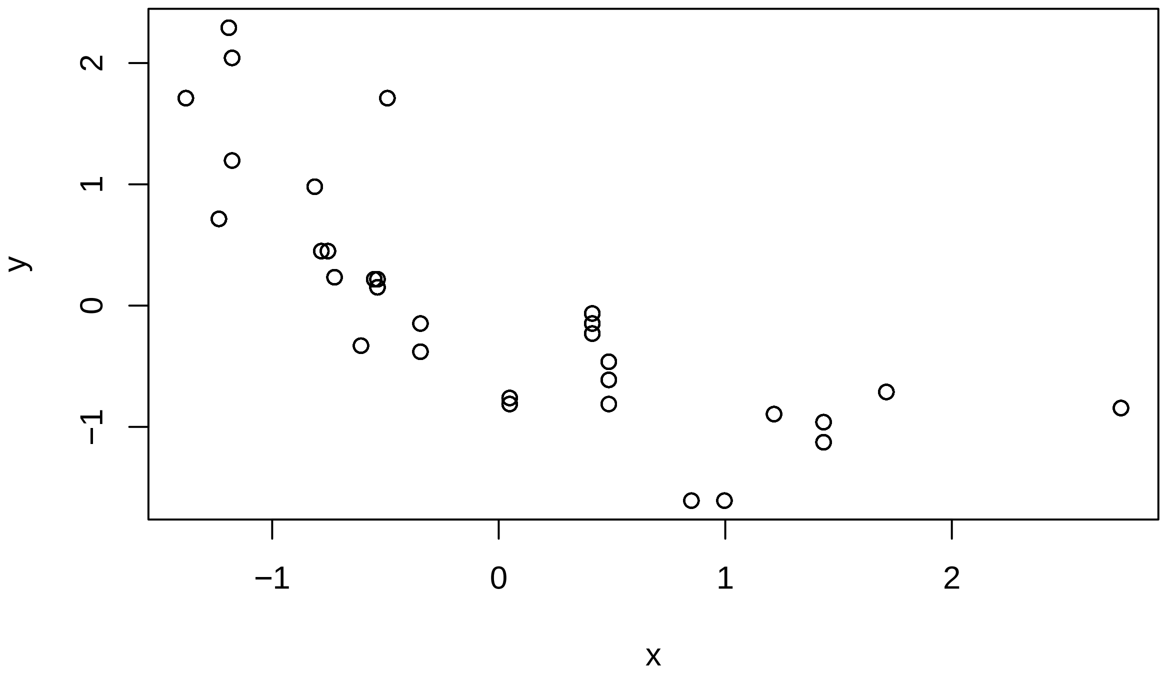
```
##  [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7
## [19]  0.8  0.9  1.0
```

```
dt <- as.data.frame(expand.grid(a,a))
colnames(dt) <- paste0("alpha",1:ncol(dt))
dt$qtl=paste0("Q",1:nrow(dt))
dt$inter <-1; dt$inter[1] <- 0
dt$count <- 1; dt$count[1] <- 0
head(dt)
```

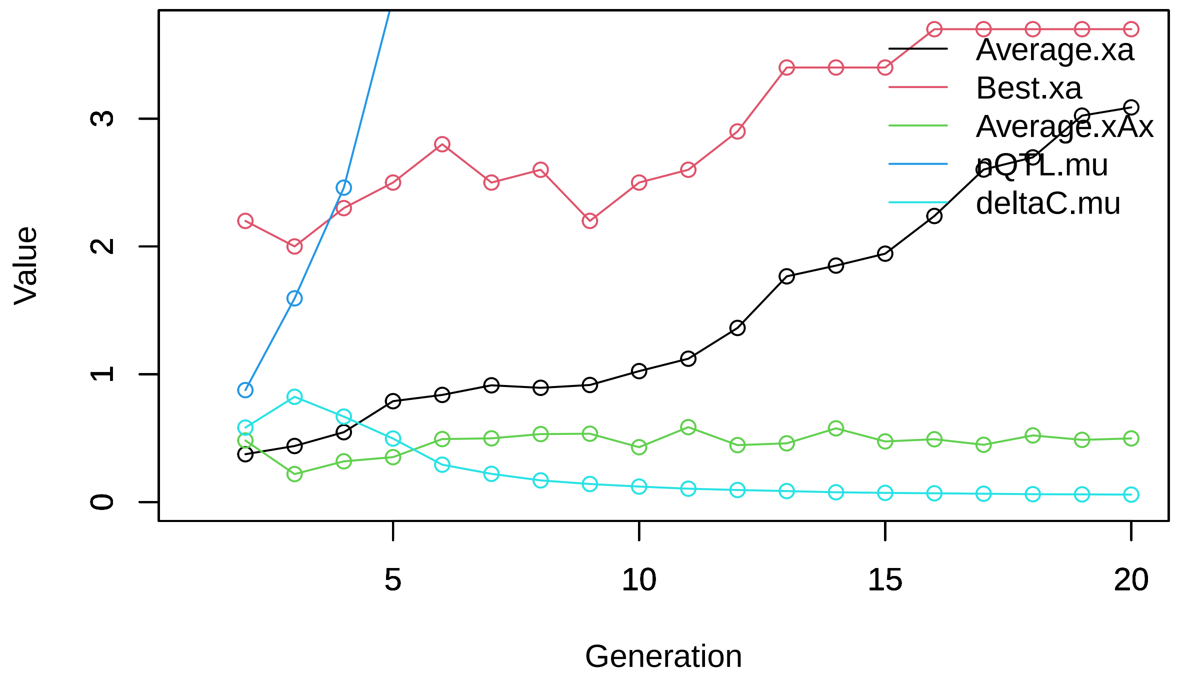
```
##   alpha1 alpha2 qtl inter count
## 1   -1.0     -1  Q1     0     0
## 2   -0.9     -1  Q2     1     1
## 3   -0.8     -1  Q3     1     1
## 4   -0.7     -1  Q4     1     1
## 5   -0.6     -1  Q5     1     1
## 6   -0.5     -1  Q6     1     1
```

```
# create n samples equivalent to the number of progeny
# you are planning to simulate (e.g., 1000)
sam <- sample(1:nrow(mtcars),500,replace = TRUE)
y <- mtcars$mpg[sam]
```

```
one <- rep(1,length(y))
x <- mtcars$hp[sam]
x2 <- mtcars$hp[sam]^2
X <- cbind(one,x)
plot(x,y)
```



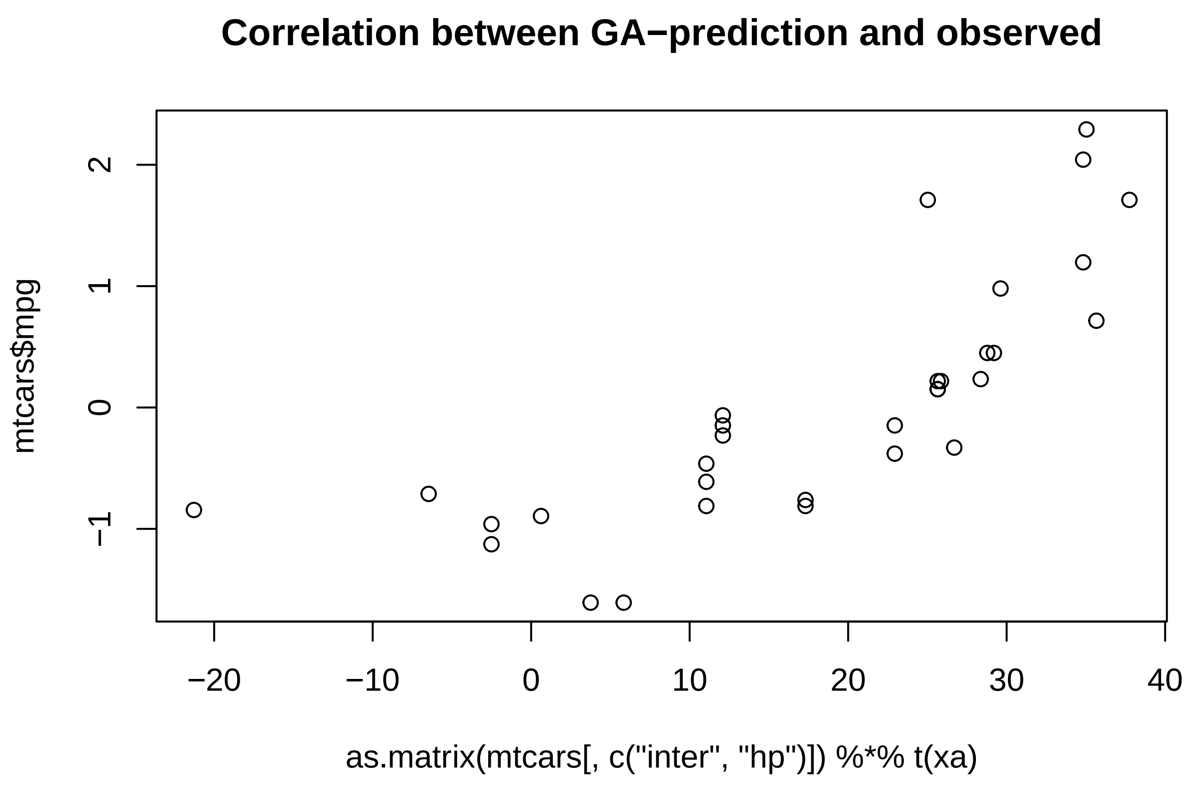
```
# Task: linear regression
res0<-evolafit(formula=cbind(inter,alpha1)~qtl, dt= dt,
               # constraints: if greater than this ignore
               constraintsUB = c(Inf,Inf),
               # constraints: if smaller than this ignore
               constraintsLB= c(-Inf,-Inf),
               # weight the traits for the selection
               traitWeight = c(1,1),
               # population parameters
               nCrosses = 50, nProgeny = 10, recombGens = 1,
               # coancestry parameters
               A=NULL, lambda=0, nQTLperInd = 1,
               # least MSE function (y - Xb)^2
               fitnessf=function(Y,b,d,Q,x,y){ apply(( (y%*%Jc(500)) - ( X%*%t(Y)) )^2,2,sum) },
               # selection parameters
               propSelBetween = 0.5, propSelWithin =0.5, selectTop=FALSE,
               nGenerations = 20, y=y, x=x, verbose = FALSE
)

# develop a joint fitness function that uses all traits

pmonitor(res0)
```



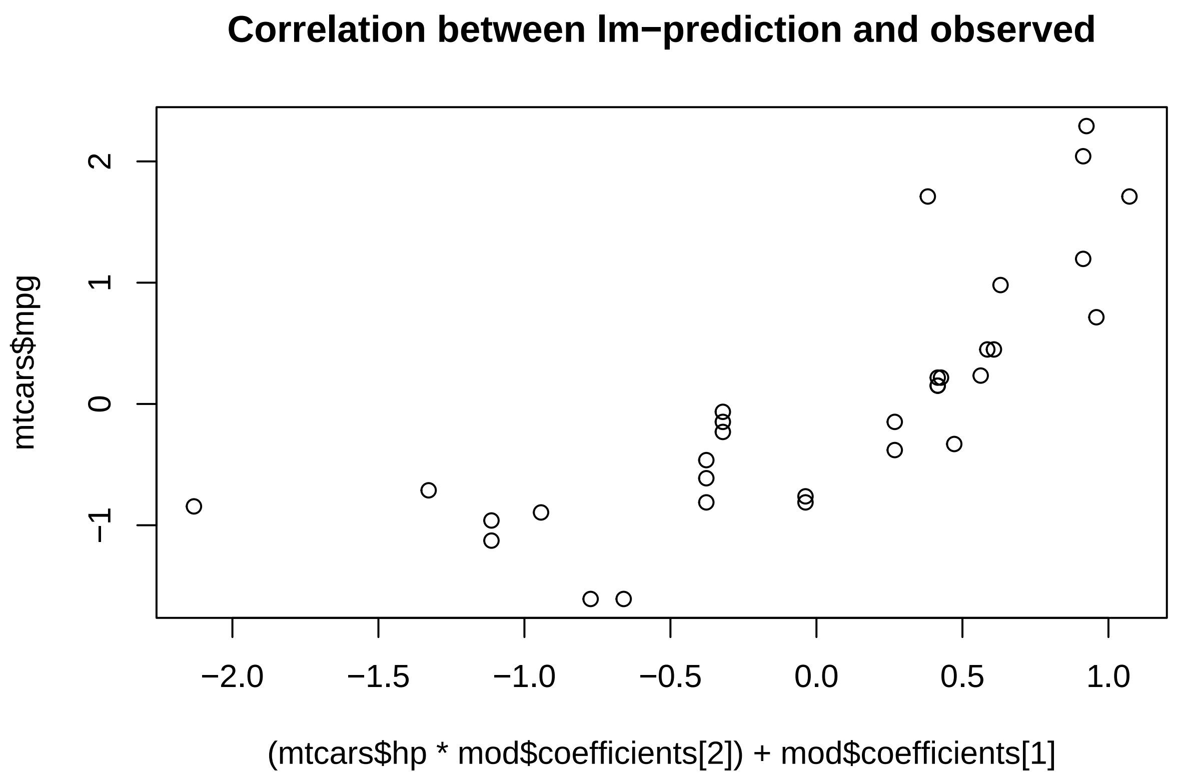
```
# this time the best solution is the one that minimizes the error
error = ( stan(y) - apply( X*res0$pheno,1,sum ) )^2
best=which(error == min(error))[1]
xa=res0$M[best,] %*% as.matrix(dt[,c("inter","alpha1")]); xa
```

```
##      inter alpha1
## [1,]    18  -14.3
```

```
plot( as.matrix(mtcars[,c("inter","hp")]) %*% t(xa)  , mtcars$mpg,
      main="Correlation between GA-prediction and observed") # GA
```



```
plot( (mtcars$hp * mod$coefficients[2] ) + mod$coefficients[1] , mtcars$mpg,
      main="Correlation between lm-prediction and observed") # LM
```



```
# Correlation between GA-prediction and observed
cor( as.matrix(mtcars[,c("inter","hp")]) %*% t(xa)   , mtcars$mpg)
```

```
##           [,1]
## [1,] 0.7761684
```

```
# Correlation between lm-prediction and observed
cor( (mtcars$hp * mod$coefficients[2] ) + mod$coefficients[1] , mtcars$mpg) # LM
```

```
## [1] 0.7761684
```

7) How to optimize the number of progeny to produce per cross

The advice here is to upload directly the phased genotypes (haplotypes) to the AlphaSimR machinery and simulate the possible crosses to explore how many individuals are required to sample a given trait (oligogenic or polygenic) with a given probablility. No need to use the evola package.

Literature

Giovanny Covarrubias-Pazaran (2024). evola: a simple evolutionary algorithm for optimization of complex problems. To be submitted to Bioinformatics.

Gaynor, R. Chris, Gregor Gorjanc, and John M. Hickey. 2021. AlphaSimR: an R package for breeding program simulations. G3 Gene|Genomes|Genetics 11(2):jkaa017. https://doi.org/10.1093/g3journal/jkaa017.

Chen GK, Marjoram P, Wall JD (2009). Fast and Flexible Simulation of DNA Sequence Data. Genome Research, 19, 136-142. http://genome.cshlp.org/content/19/1/136.