Breeding Scheme Language

Shiori Yabe, Hiroyoshi Iwata, and Jean-Luc Jannink

22 October 2018

INDEX

DESCRIPTION

Package: BreedingSchemeLanguage

Type: Package

Title: Describe and Simulate Breeding Schemes

Version: 0.9.6

Date: 2018-10-21

Authors@R: c(

person("Shiori", "Yabe", email="syabe@affrc.go.jp", role="aut"),

person("Jean-Luc", "Jannink", email="jeanluc.work@gmail.com", role=c("aut", "cre")),

person("Hiroyoshi", "Iwata", email="aiwata@mail.ecc.u-tokyo.ac.jp", role=c("aut", "ctb")),

person("Liang", "Liming", role="cph", comment="Copyright holder of GENOME software. See ./COPYRIGHTS"),

person("Abecasis", "Goncalo", role="cph", comment="Copyright holder of GENOME software. See ./COPYRIGHTS"),

person("Nishimura", "Takuji", role="cph", comment="Copyright holder of Mersenne Twister code. See ./COPYRIGHTS"),

person("Matsumoto", "Makoto", role="cph", comment="Copyright holder of Mersenne Twister code. See ./COPYRIGHTS")

)

Maintainer: Jean-Luc Jannink <jeanluc.work@gmail.com>

Description: Users can simulate their planned breeding schemes by using functions that do standard breeding tasks. Yabe, Iwata, & Jannink (2017) <doi:10.2135/cropsci2016.06.0538>.

License: GPL-3

Depends:

R (>= 3.0.0),

snowfall,

Rcpp

Imports:

ggplot2,

lme4,

```
    Matrix,
    rrBLUP,
    stats
LinkingTo: Rcpp
LazyLoad: yes
VignetteBuilder: knitr
Suggests: knitr,
        rmarkdown
NeedsCompilation: yes
RoxygenNote: 6.1.0
```

Introduction

This manual describes how to start and use the R package "BreedingSchemeLanguage". BreedingSchemeLanguage (BSL) was developed for breeders to conduct breeding simulations in a simple and flexible system. The BSL is composed of R functions that each simulate a recognizable activity in plant breeding. These functions are documented within R as typical for packages.

The BSL uses the coalescent-based whole genome simulator "GENOME" (Liang et al., 2006) to simulate a founder population. It is also possible to upload haplotypes of a founder population in hapmap format.

The BSL uses the R package "rrBLUP" (Endelman, 2011) in the function conducting genomic prediction.

Parallel processing of simulations depends on the multi-core R package "snowfall".

If you use the program, the appropriate citation is:
Yabe, S., H. Iwata, and J.-L. Jannink. 2017. A Simple Package to Script and Simulate Breeding Schemes: The Breeding Scheme Language. Crop Sci.
http://dx.doi.org/10.2135/cropsci2016.06.0538.

Reference for GENOME:
Liang L., Zöllner S., Abecasis G.R. (2006) GENOME: a rapid coalescent-based whole genome simulator.
Reference for rrBLUP:
Endelman, J.B. 2011. Ridge regression and other kernels for genomic selection with R package rrBLUP. The Plant Genome 4: 250-255.
Reference for snowfall:
Snow (Simple Network of Workstations):
http://cran.r-project.org/src/contrib/Descriptions/snow.html
Reference for lme4:
Bates et al. (2016) lme4: Linear mixed-effects models using Eigen and S4.
https://cran.r-project.org/web/packages/lme4/lme4.pdf

Breeding Scheme Language Overview

Functions

- defineSpecies(loadData = NULL , importFounderHap = NULL, saveDataFileName = NULL, nSim = 1, nCore = 1, nChr = 7, lengthChr = 150, effPopSize = 100, nMarkers = 1000, nQTL = 50, propDomi = 0, nEpiLoci = 0 , domModel = "HetHom, parms = NULL)
- defineVariances(sEnv = NULL, gVariance=1, locCorrelations=NULL, gByLocVar=1, gByYearVar=1, fracGxEAdd=0.8, plotTypeErrVars=c(Standard=1))
- defineCosts(sEnv = NULL, phenoCost=NULL, genoCost=0.25, crossCost=1, selfCost=1, doubHapCost=5, predCost=0, selectCost=0, locCost=0, yearCost=0)
- initializePopulation(sEnv = NULL, nInd = 100, parms = NULL)
- phenotype(sEnv = NULL, plotType="Standard", popID = NULL, locations=1, years=NULL, parms = NULL)
- genotype(sEnv = NULL, popID = NULL, parms = NULL)
- predictValue(sEnv = NULL, popID = NULL, trainingPopID = NULL, locations=NULL, years=NULL, sharingInfo="none", parms = NULL)
- select(sEnv = NULL, nSelect = 40, popID = NULL, random = FALSE, type="Mass", parms = NULL)
- cross(sEnv = NULL, nProgeny = 100, equalContribution = FALSE, notWithinFam=F, pedigree=F, popID = NULL, popID2, parms = NULL)
- selfFertilize(sEnv = NULL, nProgeny = 100, popID = NULL, parms = NULL)
- doubledHaploid(sEnv =simEnv, nProgeny = 100, popID = NULL, parms = NULL)
- plotData(sEnv = NULL, ymax = NULL, add = FALSE, addDataFileName = NULL, popID=NULL)
- outputResults(sEnv = NULL, summarize = TRUE, directory = NULL, saveDataFileName = "BSLoutput")
- testParameterOptimality(sEnv = NULL, schemeFileName, parmList, objectiveFunc, budget = 1000)

The parameters in parenthesis represent the default parameter values.

For all functions other than defineSpecies, the first parameter is the R environment that is created by defineSpecies and that contains objects holding the simulated data. Note that the default is "NULL", which leads the BSL to look for an environment in .GlobalEnv called "simEnv". Therefore if the environment created by defineSpecies is called simEnv [that is, use "simEnv <- defineSpecies(…)"], then sEnv need not be specified.

Functions to initiate simulations

The function "defineSpecies" uses up to five parameters to define the overall simulation settings (i.e., loadData, importFounderHap, saveDataFileName, nSim, and nCore) and then eight parameters to define the genetic architecture of the species.

- loadData: If NULL, simulate new data, else a file name should be given and the function will attempt to load data from a file previously created by setting saveDataFileName to that file name.
- importFounderHap: If "loadData" is not NULL, this parameter has no effect. If "loadData" is null and "importFounderHap" contains a file name the function will attempt to load founder haplotype data from a hapmap format file. See details below.
- saveDataFileName: Name under which to save newly-simulated data. No file suffix needs to be given to this name. If NULL is given, simulated data will not be saved.

If loadData or importFounderHap parameters (above) are not NULL the parameters below are not needed (because they are defined by what is uploaded).

- nSim: Number of repeats of the simulation
- nCore: Simulations can be processed in parallel if the number given here is greater than 1
- nChr: Number of chromosomes of the species
- lengthChr: Length of each chromosome in cM (all chromosomes have the same length)
- effPopSize: Effective population size of the base population. An idealized population is assumed leading up to the beginning of the simulation
- nMarkers: Number of markers observable for making predictions

- nQTL: Number of genetic effects controlling the target trait. If there is no epistasis, this is also the number of QTL. Under epistasis, the expected number of causal loci will be nQTL times (nEpiLoci + 1). The nEpiLoci parameter is defined below
- probDomi: Probability of a QTL locus exhibiting dominance
- nEpiLoci: Expected number of interacting loci contributing to each effect
- domModel: The two options are "HetHom" (the default) a dominance model that gives opposite effects to heterozygotes versus homozygotes, or "Partial", a dominance model that requires specifying a degree of dominance of the derived allele. At the moment, functionality for the "Partial" option is only available when using ImportFounderHap.

ImportFounderHap details. This feature uses a hapmap format with loci in rows, eleven (11) locus annotation columns, and then one column for each haplotype. Annotation columns as follows. Column 1: locus names. Column 2: possible alleles (e.g. "A/G"). Column 3: chromosome number. Column 4: locus position in cM. If users specify QTL information, it is as follows. Column 7: the integer ID of the effect (more than one locus can contribute multiplicatively to one effect if all loci have the same integer ID). Column 8: action type (0 for recessive, 0.5 for additive, 1 for dominant, in between for partial recessive or dominant). Column 9: the effect of the QTL. Columns 12 through 12+n (where n is the number of haplotypes or genotypes) should contain EITHER single letter allele codes (anything other than "A", "T", "C", or "G" will be set to missing) OR two letter phased genotype codes (that is a diploid individual with adjacent loci CT then GA is assume to have the same genotype, but different phase as the individual CT then AG.

The value of "defineSpecies" is an environment. The typical use of defineSpecies should therefore be

```
simEnv <- defineSpecies()
```

However, note that environments in R are like pointers in other languages. If you write over the pointer, that does not delete the objects pointed to. Therefore, before using the above command, it is best to make sure that simEnv does not already exist and point to existing simulation objects:

```
if (exists("simEnv")){
   rm(list=names(simEnv), envir=simEnv)
```

```
  rm(simEnv)
}
```

The function "defineVariances" specifies genetic and genotype by environment variances in the founder population. NOTE: if defineVariances is not called, the default variances given below are used by BSL.

- gVariance: The genetic variance among founders. Default is 1.
- locCorrelations: The default is NULL which indicates random locations where each location generates a genotype x location interaction of variance specified by gByLocVar (see below). If a correlation matrix is passed, locations are fixed and the matrix specifies the genetic correlation for performance between locations. Then the genetic covariance is gVariance * locCorrelations.
- gByLocVar: Numeric value. This parameter is only relevant if locCorrelations is NULL. Then it specifies the genotype by location variance affecting individuals in the base population evaluated in a new location. Default is 1.
- gByYearVar: Numeric value. Years are always considered random and evaluations in new years affect individuals in the base population with a deviation of variance gByYearVar. Default is 1.
- fracGxEAdd: One fraction of the genotype by Year and genotype by Location deviations is determined by segregating QTL and is additive. Another fraction is random and IID. The additive fraction is determined by this parameter which can vary between 0 and 1. Default is 0.8.
- plotTypeErrVars: named vector. The user can define a certain number of plot types that will be used in the breeding scheme, for example, small plots with high error variance for early generations to large plots with low error variance for late generations. The vector contains the assumed error variances and the names are the names of the plot types that will be used when the phenotype() function is called. Default is to name a plot type "Standard" with an error variance of 1.

The function "defineCosts" specifies the cost in arbitrary units for different breeding activities. If

this function is called, the BSL keeps track of the cost of the total scheme by adding costs with each activity simulated. NOTE: to ensure that the costs of locations are properly accounted, call defineCosts *after* you call defineVariances.

- phenoCost: named vector. Phenotyping is done with user-specified plot types, each with their own cost. The vector contains the assumed cost and the names are the names of the plot type used when the phenotype() function is called.
- genoCost: The cost per individual of genotyping.
- crossCost: The cost per individual of creating that individual by crossing.
- selfCost: The cost per individual of creating that individual by self-fertilizing its parent.
- doubHapCost: The cost per individual of creating that individual by doubled haploidy from its parent.
- predCost: The cost of running a prediction analysis. Default is zero because it is assumed that the capacity to run predictions is a fixed cost.
- selectCost: The cost of running a selection analysis. Default is also zero.
- locCost: The cost of maintaining experimental fields at a location. Default is also zero.
- yearCost: The cost per year of maintaining a breeding program. Default is also zero.

The function "initializePopulation" creates a founder population for breeding:

- nInd: The number of founders (Default is 100).

Breeding functions

The function "phenotype" causes a phenotyping trial to be run:

- plotType: String name of plot type. Plot type will determine the error variance and cost on a plot basis. Note that the trial phenotypes will also be affected by genotype x location and genotype x year effects specified in defineVariances. (Default is "Standard")

- nRep: Numeric. The number of replications per genotype. (Default is 1). nRep values greater than 1 will cause the error variance of the trial to be less than the error variance determined by the plot type.
- popID: Population ID to be phenotyped (Default is the last population created)
- locations: Integer vector specifying at which locations phenotyping should take place. If locations are fixed (i.e., the locCorrelations parameter was specified in defineVariances), then all elements of the vector should be lower or equal to the size of the locCorrelations matrix. If locations are random (i.e., locCorrelations = NULL), the genotype by location interaction deviations are sampled for each new location. Thus, for example, if locations=1:5, deviations will be sampled for five locations. A subsequent call to phenotype could specify locations=c(2, 4) and no new deviations would be sampled. The default is location 1.
- years: Integer vector similar to locations, though years are always random. The breeding scheme starts in year 1. The default is that each new phenotyping activity takes place in the same year as the previous phenotyping activity. Thus, to phenotype in a new (the next) year, specify the next year number (e.g., if past phenotyping was in years 1 & 2, specify 3).

The function "genotype" causes marker genotypes to become available for the specified population.

- popID: Population ID to be genotyped (Default is all non-genotyped individuals in the breeding scheme)

The function "predictValue" analyses phenotypes to generate selection criteria:

- popID: Population ID to be predicted (Default is the latest population created)
- trainingPopID: A vector of population IDs to be used to train a prediction model (Default is all populations having phenotype and genotype data)
- locations: Integer vector specifying from what locations you want to get the phenotypes. Default is all locations.
- years: Integer vector specifying from what years you want to get the phenotypes. Default is all years.

- sharingInfo: In predicting the value of an individual, information can be shared from other individuals. If "none" (the default) individual effects are assumed independent, so there is no information sharing. Effects are estimated using lme4::lmer. If "markers" individual effects are related to each other additively through markers (GBLUP model). If "pedigree" individual effects are related to each other additively through their pedigree relationships (ABLUP model). For both options "markers" and "pedigree", effects are estimated using rrBLUP::kin.blup.

The function "select" conducts selection in the defined population:

- nSelect: Number of individuals to select
- popID: Population ID to be selected (Default is the last population created when random = TRUE. When random = FALSE, default is the last evaluated or predicted population)
- random: If TRUE, individuals are selected at random. If FALSE, the selection criterion depends on previous breeding activities: if the last activity was "phenotype", then selection will be on mean phenotypes; if the last activity was "predictValue", then selection will be on predicted values.
- type: The two options are "Mass" (the default) and "WithinFamily". With mass selection, all individuals are ranked and the highest nSelect are taken. If WithinFamily, individuals are ranked within half-sib (if population was randomly mated) or full-sib (if population from selfFertilize or doubledHaploid) and the highest nSelect within families are taken.

Mating functions

The "cross" function conducts random mating among parents.

- nProgeny: Number of progeny to generate by random mating
- equalContribution: If TRUE, all individuals are used the same number of times as parents and self-fertilization is not allowed. This setting increases the effective population size for a given number of progeny. If FALSE, pairs of parents are chosen

at random for each progeny and self-fertilization has a probability of 1 / (number of parents).

- popID: Population ID to be used as parents (Default is the last population created)
- popID2: ID of a second population to be used as parents. If not null, the mating design will force one parent to be taken from popID and one parent from popID2 for each cross. Thus, popID2 can be used to simulate reciprocal recurrent selection.
- notWithinFam: if TRUE, like equalContribution, all individuals are used the same number of times as parents and self-fertilization is not allowed. In addition, half- and full-sibs are not allowed to mate.
- Pedigree: optional two- or three-column matrix: the first two columns are the GIDs that you want to cross, the third column is the number of progeny from that cross. NOTE: pedigree supersedes the nProgeny, equalContribution, popID, popID2, and notWithinFam parameters. You have to know what you are doing to use this parameter. Default: NULL

The "selfFertilize" function implements inbreeding.

- nProgeny: Number of selfed progeny to make. Parents in the population are used as evenly as possible to make the progeny, i.e., each parent is used minimally floor(nProgeny / nParents) times with remaining progeny allocated at random.
- popID: Population ID to be used as parents (Default is the last population created)

The "doubledHaploid" function makes doubled haploids progeny.

- nProgeny: Number of doubled haploid progeny to make, allocated as in selfFertilize.
- popID: Population ID to be used as parents (Default is the last population created)

Result functions

The function "plotData" draws a figure of the genotypic value through generations of breeding. The figure shows population means of each simulation replication and the mean value over repeated simulations (given by the nSim parameter in "defineSpecies" function). If the defineCosts function was called, the total cost of the breeding scheme is the title of the figure. The function

returns a matrix of the values that were just plotted.

- ymax: Maximum genotypic value on the y-axis of the figure.
- add: If TRUE results will be added to previous data obtained from the "addDataFileName" file (see below).
- addDataFileName: String giving the name of a file from which to obtain data from a previous simulation. Also, results used in making this plot will be saved to that data file. No file suffix needs to be given to this name. If NULL is given, plot data will not be saved.
- popID: Optional list of vectors of population IDs allowing the user to specify which populations to plot. The mean of populations in the vector will be plotted in order of the vectors in the list. For example, if popID=list(0:1, 2:3) is given, the average value of individuals in populations 0 and 1 will be plotted and then the average value of individuals in populations 2 and 3. If not given, the default will be to plot each population's mean genotypic value when it was first created by "cross", "selfFertilize", or "doubledHaploid" and before any lines were selected out of it.

The function "outputResults" saves the results. If saveDataFileName is NULL, data is return as an object, else data is saved as a ".rds" file. To examine the data, use the readRDS function.

- summarize: If TRUE results averaged over simulation replications will be saved. If FALSE, all data from breeding simulations will be saved
- directory: If NULL data will be saved in the R working directory. If a string giving the name of a directory, data will be saved there. When summarize = FALSE, extensive data is saved so that dedicating a directory to it may be wise
- saveDataFileName: String giving the name of a file in which simulation results are saved. No file suffix needs to be given to this name

Optimization function

The function "testParameterOptimality" returns the response of a breeding scheme given a simulation environment and a parameter vector.

- sEnv: the environment the breeding scheme will be evaluated in. This environment should be fresh from defineSpecies, defineVariances, and defineCosts. If NULL, the default simEnv is attempted.
- schemeFileName: source file that holds the script of the breeding scheme. Note that sEnv must explicitly used in that source file as the simulation environment.
- parmList: named list with values of parameters characterizing the breeding scheme. The parameters will be referred to in the source file above.
- objectiveFunc: a function that can be applied to the simulation environment after the breeding scheme has been simulated, to return a value showing the realized performance of the breeding scheme
- budget: the maximum budget that is allowed for the breeding scheme

Population ID

Initial population ID = 0

Population ID will be incremented by these functions:

1. select
2. cross
3. selfFertilize
4. doubledHaploid

Be careful that:

select() creates a new population that is a subset of the candidate population

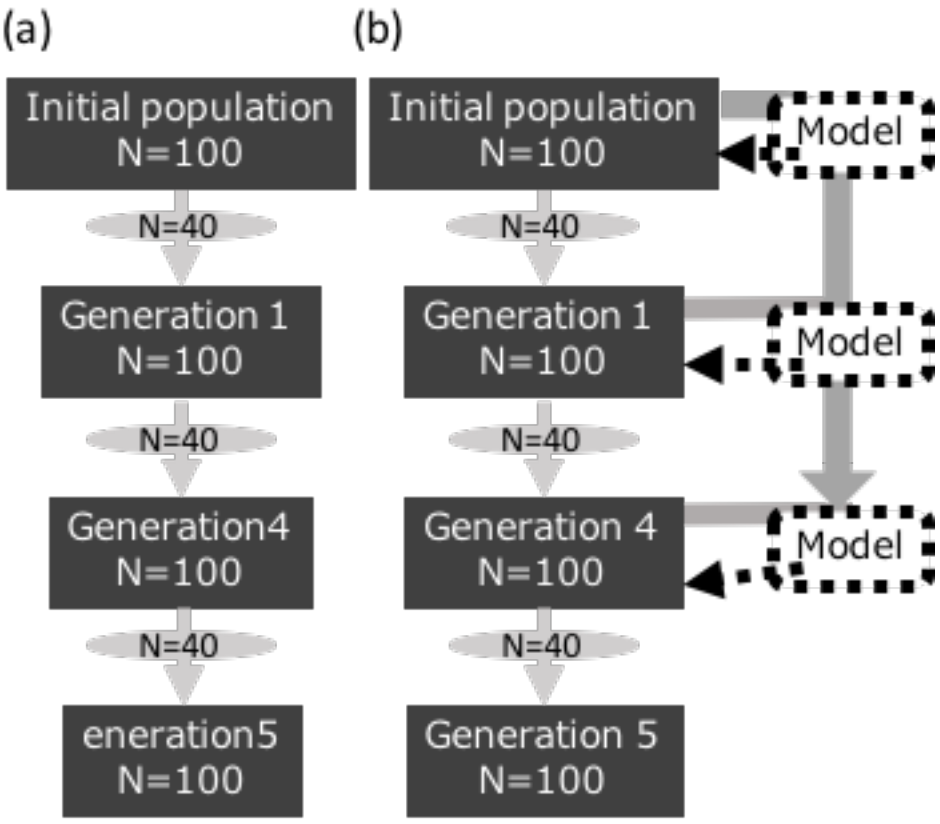
Example 1: Phenotypic selection & genomic selection

1. Phenotypic mass selection (Fig. a)

```
simEnv <- defineSpecies(nSim = 5, saveDataFileName="previousData")
# Because environment named "simEnv" it doesn't need to be specified
below
initializePopulation() # popID 0 created
phenotype()
select() # popID 1 selected out of popID 0
cross() # popID 2 created
phenotype()
select() # popID 3 selected out of popID 2
cross() # popID 4 created
phenotype()
select() # popID 5 selected out of popID 4
cross() # popID 6 created
plotData(addDataFileName="plotData") # plots popIDs 0, 2, 4, and 6
```

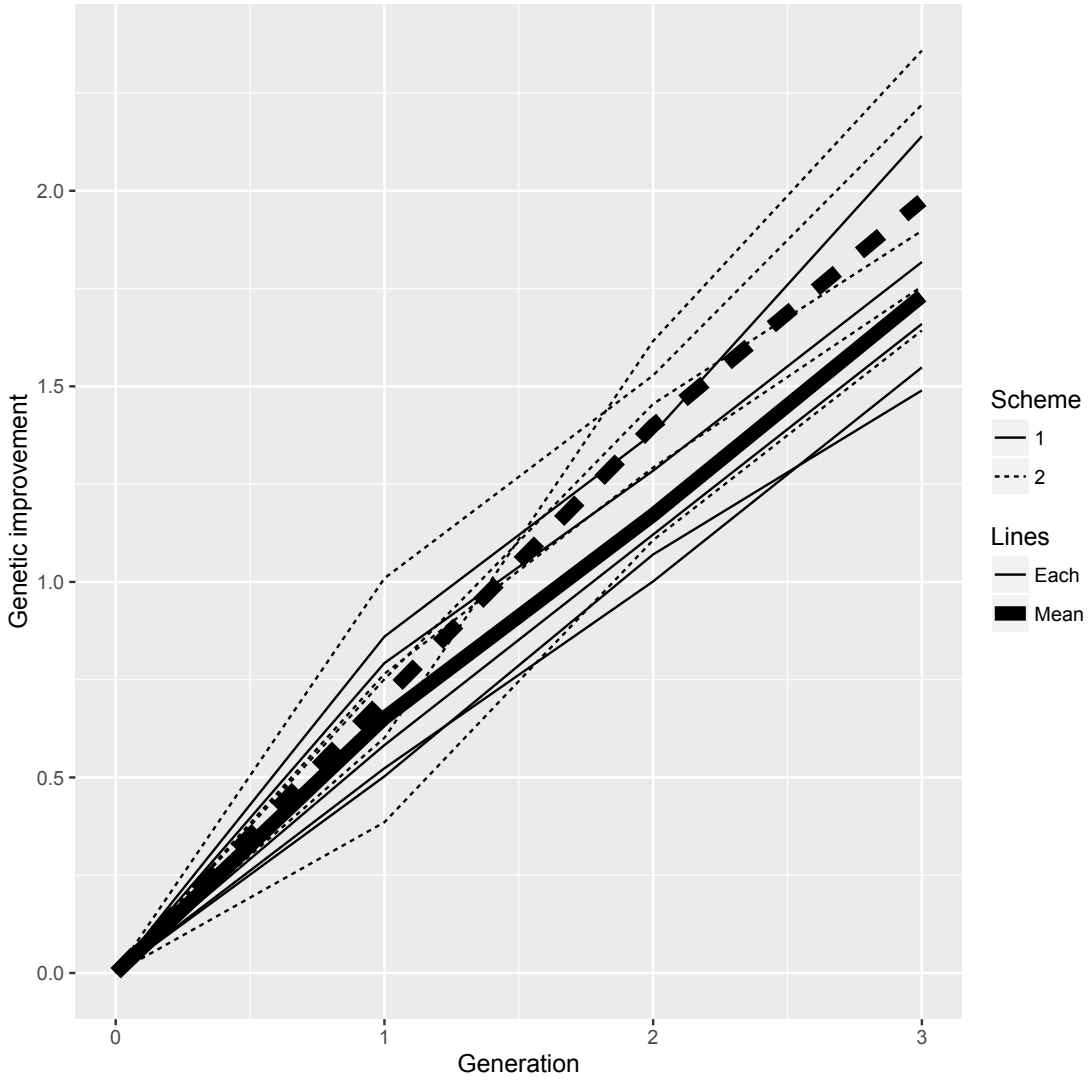
2. Genotypes aiding breeding value estimation on existing phenotypic data (Fig. b)

```
simEnv <- defineSpecies(loadData="previousData")
initializePopulation()
phenotype()
genotype()
predictValue(sharingInfo="markers")
select()
cross()
phenotype()
genotype()
predictValue(sharingInfo="markers")
select()
cross()
phenotype()
genotype()
predictValue(sharingInfo="markers")
select()
cross()
plotData(add=T, addDataFileName="plotData")
```



Simulation diagram

Scheme 1 is phenotypic selection, and Scheme 2 is selection on breeding values determined by a genomic relationship matrix.



Simulation result

Bold lines represent the mean values of simulation trials (in this case, 3 generations of selection), and each thin line is the result of a simulation trial.

Example 2: Fixed locations and different information sharing

```
simEnv <- defineSpecies(loadData = "previousData")
# Fixed locations with cov(l1,l2)=0.6; cov(l1,l3)=0.3; cov(l2,l3)=0.8
locCor <- matrix(c(1, 0.6, 0.3, 0.6, 1, 0.8, 0.3, 0.8, 1), 3)
# Two kinds of plots one with error variance 4, the other with error
variance 1. The former costs 2 units the latter 5 units.
errVars <- c(Early=4, Late=1)
plotCosts <- c(Early=2, Late=5)
defineVariances(locCorrelations=locCor, plotTypeErrVars=errVars)
defineCosts(phenoCost=plotCosts)
initializePopulation()
# Phenotype in all locations over two years the less expensive way
phenotype(locations=1:3, years=1:2, plotType="Early")
predictValue() # No information sharing: individual effects are IID
select()
cross()
# Phenotype in location 3 only, the expensive way.
phenotype(plotType="Late", locations=3, years=3)
phenotype(plotType="Early", locations=1, years=3)
# Use pedigree information for relationship matrix
predictValue(sharingInfo="pedigree")
select()
# Self-fertilize selected individuals to create 120 progeny
selfFertilize(nProgeny=120)
genotype()
# Specify locations=3 to only use trials from that location for
training
predictValue(sharingInfo="markers", locations=3)
select()
cross()
plotData()
```