Breeding Scheme Language

Shiori Yabe, Hiroyoshi Iwata, and Jean-Luc Jannink

25 February 2017

INDEX

Description……2
Introduction……...3
Required environment……...4
Install BreedingSchemeLanguage……5
Describe breeding scheme……6
Example 1……10

Description

Package: BreedingSchemeLanguage
Type: Package
Title: Describe and simulate breeding schemes
Version: 1.0
Date: 2017-02-25
Author: Shiori Yabe, Hiroyoshi Iwata, and Jean-Luc Jannink
Maintainer: Shiori Yabe <syabe@affrc.go.jp>
Description: Users can simulate breeding schemes by using functions that implement breeding activities.
License: GPL-3
Depends: R (>= 3.0.0), Rcpp (>= 0.11.0), snowfall
Imports:
    ggplot2,
    rrBLUP,
    lme4
LinkingTo: Rcpp
LazyLoad: yes
NeedsCompilation: yes
RoxygenNote: 5.0.1

Introduction

This documentation 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. Users can use BSL under the R environment.

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 "snow fall".

If you use the program, the appropriate citation is:

_____.

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

Required environment

Downloading and installing R is necessary to use the package. Users can download R on the web page:
The Comprehensive R Archive Network (CRAN) https://cran.r-project.org

The BSL package needs Rtools (for Windows) or Xcode (for Mac OS X). These tools are used to install the BSL package from GitHub (https://github.com) and to compile C++ code on your PC. Compiling C++ code is automatically done when you install the BSL package.
Rtools: https://cran.r-project.org/bin/windows/Rtools/
Xcode: Mac App Store (You need your “Apple ID”.)

Install BreedingSchemeLanguage

On your R screen, you can write the code below:

```
install.packages(“devtools”)
devtools::install_github("syabe/BreedingSchemeLanguage")
```

After installing the Breeding Scheme Language package, the only command needed to use it is:

```
library(BreedingSchemeLanguage)
```

Now, you can use the “Breeding Scheme Language” on your PC!

Describe breeding scheme

Help

You can call help pages two ways:

?FunctionName

help(FunctionName)

Functions

- defineSpecies(loadData = NULL , importFounderHap = NULL, saveDataFileName = "previousData", nSim = 1, nCore = 1, nChr = 7, lengthChr = 150, effPopSize = 100, nMarkers = 1000, nQTL = 50, propDomi = 0, nEpiLoci = 0 , domModel = "HetHom")
- defineVariances(sEnv=simEnv, gVariance=1, locCorrelations=NULL, gByLocVar=1, gByYearVar=1, fracGxEAdd=0.8)
- defineCosts(sEnv=simEnv, phenoCost=data.frame(1,1), genoCost=0.25, crossCost=1, selfCost=1, doubHapCost=5, predCost=0, selectCost=0, locCost=0, yearCost=0)
- initializePopulation(sEnv=simEnv, nInd = 100)
- phenotype(sEnv=simEnv, errorVar = 1, popID = NULL, locations=1, years=NULL)
- genotype(sEnv=simEnv, popID = NULL)
- predictValue(sEnv=simEnv, popID = NULL, trainingPopID = NULL, locations=NULL, years=NULL, sharingInfo=”none”)
- select(sEnv=simEnv, nSelect = 40, popID = NULL, random = F, type=”Mass”)
- cross(sEnv=simEnv, nProgeny = 100, equalContribution = F, popID = NULL, popID2)
- selfFertilize(sEnv=simEnv, nProgeny = 100, popID = NULL)
- doubledHaploid(sEnv=simEnv, nProgeny = 100, popID = NULL)
- plotData(sEnv=simEnv, ymax = NULL, add = F, addDataFileName = “plotData”, popID=NULL)
- outputResults(sEnv=simEnv, summarize = T, directory = NULL, saveDataFileName = “BSLoutput”)

The parameters in parenthesis represent the default parameter values.

For all functions other than defineSpecies, the first parameter the R environment that is created by defineSpecies and that contains objects holding the simulated data. Note that the default is an environment called "simEnv" so if the environment created by defineSpecies is called simEnv [that is, use "simEnv <- defineSpecies(…)"], then this environment 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
- 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.

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 genotypes (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 ID of the effect (more than one locus can contribute multiplicatively to one effect if all loci have the same effect ID). Column 8: action type (0 for additive, 1 for dominance, in between for partial dominance). Column 9: the effect of the QTL.

The function “defineVariances” specifies genetic and genotype by environment variances in the founder population.

- gVariance: The genetic variance among founders. The 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 gByLVar (see below). If a correlation matrix is passed, locations are fixed and the matrix species the genetic correlation for performance between locations. Then the genetic covariance is gVariance * locCorrelations.
- gByLocVar: 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.

- gByYearVar: Years are always considered random and evaluations in new years affect individuals in the base population with a deviation of variance gByYearVar.
- 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.

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: Phenotyping can be done with user-specified error variance. The BSL assumes that the cost of phenotyping is a function of this error variance. Thus, phenoCost is a data frame with an error variance in the first column and the cost of a plot affected by that error variance in the second column.
- 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

Breeding functions

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

- errorVar: Environmental error variance of the trial
- popID: Population ID to be phenotyped (Default is the last population created)
- locations: Integer vector specifying at which locations should phenotyping 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 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 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. 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.

- select(sEnv=simEnv, nSelect = 40, popID = NULL, random = F, type=”Mass”)

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. The number of progeny should be larger than the number of parents in popID. 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.
- 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.

The "selfFertilize" function implements inbreeding.

- nProgeny: Number of selfed progeny to make. Parent 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).

- 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
- popID: Optional vector of population IDs allowing the user to specify which

populations to plot. 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.

- 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 = F, 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

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

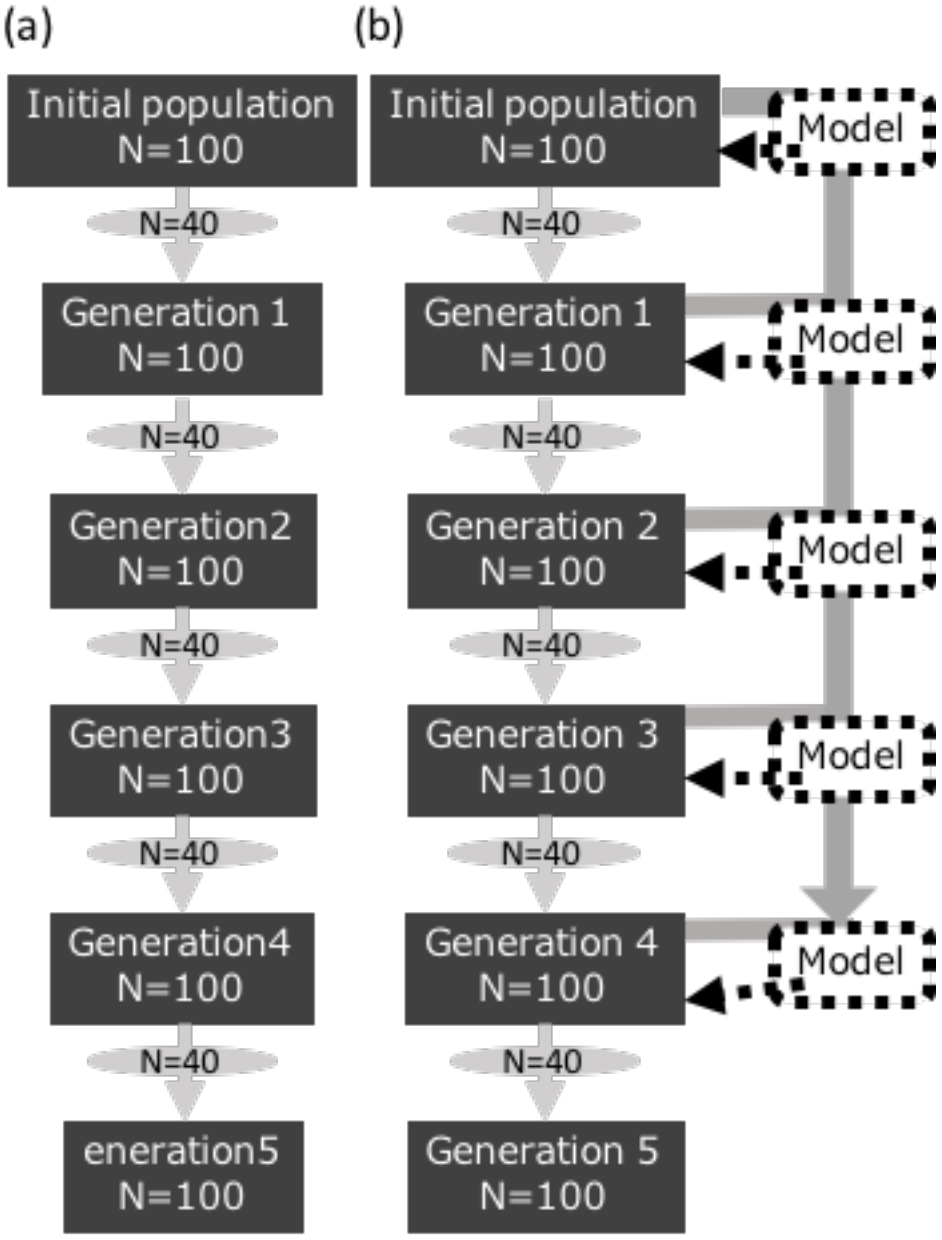
Example 1: Phenotypic selection & genomic selection

1. Phenotypic mass selection (Fig. a)

```
simEnv <- defineSpecies(nSim = 5)
# Because environment named “simEnv” it needn’t to be specified below
defineVariances()
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() # plots popIDs 0, 2, 4, and 6
```

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

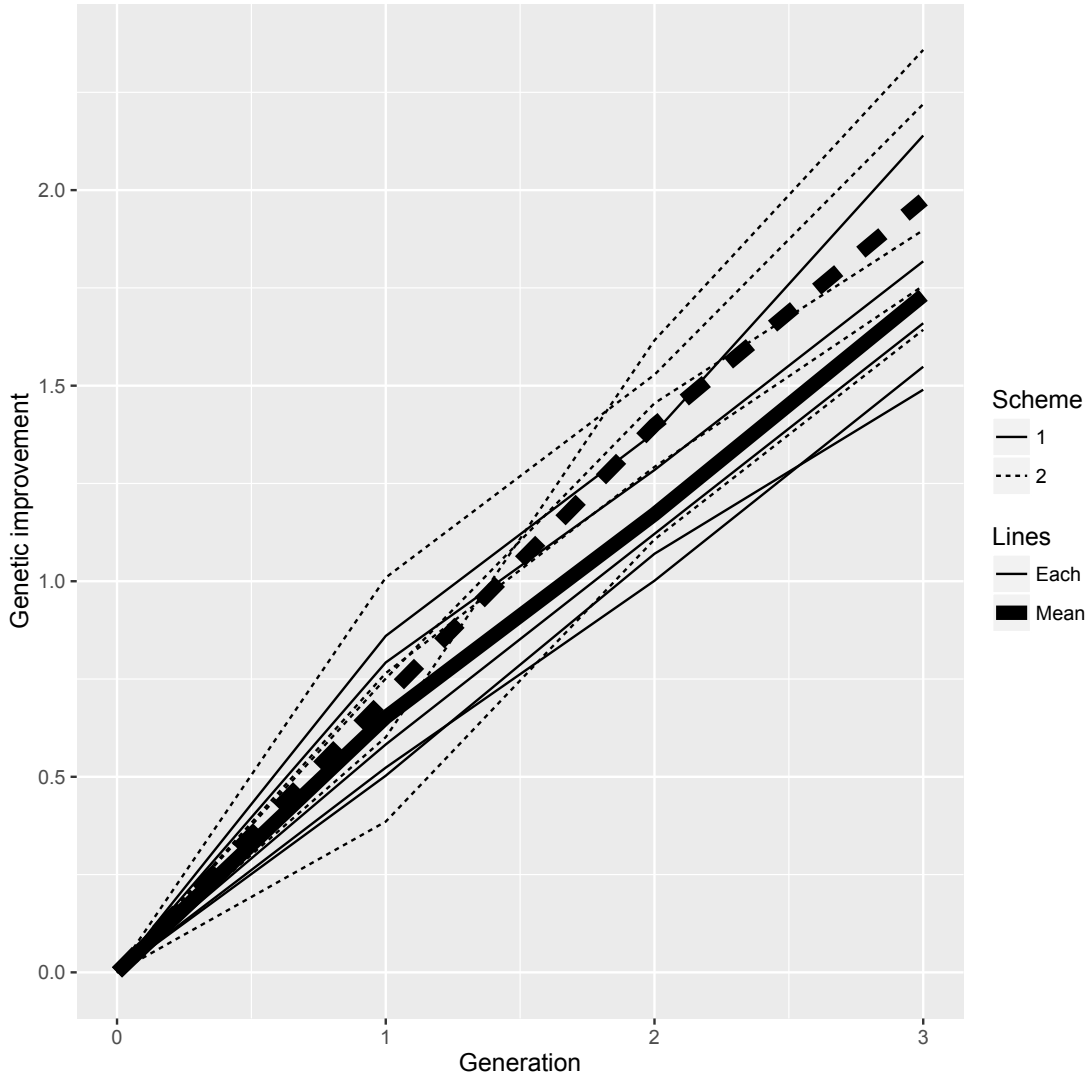
```
simEnv <- defineSpecies(loadData="previousData")
defineVariances()
initializePopulation()
phenotype()
genotype()
predictValue(sharingInfo="markers")
select()
cross()
phenotype()
genotype()
predictValue(sharingInfo="markers")
select()
cross()
phenotype()
genotype()
predictValue(sharingInfo="markers")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



Simulation result:

Scheme 1 is phenotypic selection, and Scheme 2 is genomic selection.

Bold lines represent the mean values of simulation trials (in this case, 5 generations of selection), and each thin line is the result of a simulation trial.


Genetic improvement
2.0
1.5
1.0
0.5
0.0
0
1
2
3
Generation
Scheme
1
2
Lines
Each
Mean

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)
defineVariances(locCorrelations=locCor)
# Two kinds of phenotyping one with error variance 4, the other with error
variance 1. The former costs 2 units the latter 5 units.
phenoCost <- data.frame(error=c(4, 1), cost=c(2, 5))
defineCosts(phenoCost=phenoCost)
initializePopulation()
# Phenotype in all locations over two years the less expensive way
phenotype(locations=1:3, years=1:2, errorVar=4)
predictValue() # No information sharing: individual effects are IID
select()
cross()
# Phenotype in location 3 only, the expensive way. NOTE: the call
# increases to the next year, which is year 3 in this case
phenotype(locations=3, errorVar=1)
# Specify phenotyping in year 3 for a separate trial in location 1
# Otherwise BSL will increment to year 4
phenotype(locations=1, years=3, errorVar=4)
# 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()
```