Breeding Scheme Language

Shiori Yabe, Hiroyoshi Iwata, and Jean-Luc Jannink

December 26, 2016

INDEX

Description

Package: BreedingSchemeLanguage
Type: Package
Title: Describe and simulate breeding schemes
Version: 1.0
Date: 2016-03-08
Author: Shiori Yabe, Hiroyoshi Iwata, and Jean-Luc Jannink
Maintainer: Shiori Yabe <syabe@affrc.go.jp>
Description: Users can simulate their planned breeding schemes by using functions that imitate events in breeding.
License: GPL-3
Depends: R (>= 3.0.0), Rcpp (>= 0.11.0), snowfall
Imports:
    ggplot2,
    rrBLUP
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 utilizes the coalescent-based whole genome simulator "GENOME" (Liang et al., 2006) to simulate a founder population for simulation.

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

In simulations, the multi-core calculation depends on the R package "snow fall".

If you use the program, the appropriate citation is:
Yabe S., Iwata H., Jannink J.L. (2017) A Simple Package to Script and Simulate Breeding Schemes: The Breeding Scheme Language. Crop Science (in print).

Reference for GENOME:
Liang L., Zöllner S., Abecasis G.R. (2006) GENOME: a rapid coalescent-based whole genome simulator. Bioinformatics 23: 1565-1567.

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 version of R should satisfy the required condition shown in 'Depends' on the Description page.

The BSL package needs Rtools or Xcode for windows and Mac PC, respectively. 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 the package ‘BreedingSchemeLanguage’

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

```
install.packages(“devtools”)                # only at the first time
library(devtools)              # only at the first time
install_github("syabe/BreedingSchemeLanguage")                  # only at the first time
library(BreedingSchemeLanguage)
```

Once you install the package, the first three commands can be removed.
If you asked the mirror cite when you conduct the code, please chose one place you prefer.

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

Functions

List of functions

defineSpecies
initializePopulation
phenotype
genotype
predictBreedVal
select
cross
selfFertilize
doubledHaploid
plotData
outputResults

Population ID

Many functions in the package BreedingSchemeLanguage use the parameter “popID”. This is the abbreviation of the “Population ID”. The population ID starts at 0 for the initial population. Functions that generate a new population assign a popID number to it by incrementing the current popID value. The functions that create a population with new PopID are:

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

In the function “cross” (the function for random mating; see below the function usage), you can find an input **‘popID2’**. When you make hybrids derived from two populations, you can use ‘popID2’, here. The genotype from popID is crossed with the genotype from popID2. This allows us to conduct the reciprocal recurrent selection, which is expecting the effective utilization of dominance effect. In maize breeding, the method is frequently used in plant breeding region.

Note: while the function "select" increments the population ID, the function "plotData"

only shows generations created by the functions "cross", "selfFertilize", and "doubledHaploid", if you do not have any input to 'popIDplot' in the function "plotData". The function plotData, under such situation, does not show the difference between a population of selection candidates and the individuals picked from it by "select".

Functions that you can execute in the package

(i) Functions to initiate simulations

Define and create species:

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)

The function "defineSpecies" requires five parameters to define the overall simulation settings (i.e., loadData, importFounderHap, saveDataFileName, nSim, and nCore) and then seven parameters to define the genetic architecture of the species (i.e., nChr, lengthChr, effPopSize, nMarkers, nQTL, propDomi, nEpiLoci). Here, the software GENOME (Liang et al., 2007) was used to create a new founder haplotypes.

- loadData: if null create a new species (default), else the file name of previously created species like "fileName.RData".
- importFounderHap: if null create new founder haplotypes (default), else the file name of externally generated founder haplotypes in hapmap format.
- saveDataFileName: the name to save the species data with double-quotation, like "species1_1". (default: "previousData")
- nSim: the number of simulation trials
- nCore: simulation processed in parallel over this number of CPUs (Check computer capacity before setting above 1.)
- nChr: the number of chromosomes
- lengthChr: the length of each chromosome (cM; all chromosomes are the same length)
- effPopSize: the effective population size in the base population

- nMarkers: the number of markers, which is used especially for genomic selection
- nQTL: the number of QTLs controlling the target trait
- propDomi: the probability of dominant QTL among the all QTL
- nEpiLoci: the expected number of epistatic loci for each effect

The result of defineSpecies() should be assigned to a variable like

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

The variable 'simEnv' must be used in all the following functions as the receiver of simulated data.

Create a founder population:

initializePopulation(simEnv, nPop = 100, gVariance = 1)

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

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results
- nPop: the population size
- gVariance: the genetic variance in the initial population

You do not need to assign the simulated results to any variables. For all the following functions, assignment of the results is not necessary.

(ii) Breeding functions

Evaluate the phenotypic value:

phenotype(simEnv, errorVar = 1, popID = NULL)

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

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results
- errorVar: the error variance
- popID: the population ID to be evaluated (default: the latest population)

Genotype markers:

genotype(simEnv)

The function "genotype" causes marker genotypes to become available for all individuals generated in the breeding scheme.

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results

Genomic prediction:

predictBreedVal(simEnv, popID = NULL, trainingPopID = NULL)

The function "predictBreedVal" performs genomic prediction by using phenotypic data and genotypic data generated in the breeding scheme. This function uses the R package "rrBLUP" (Endelman, 2011) to conduct GBLUP assuming that we know the values of environmental error variance (i.e., the errorVar parameter in the function "phenotype").

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results
- popID: the population ID to be predicted (default: the latest population)
- trainingPopID: the population ID to be used for training a prediction model (default: all populations with phenotype data)

Select individuals:

select(simEnv, nSelect = 40, popID = NULL, random = F)

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

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results
- nSelect: the number of selected individuals
- popID: the population ID to be selected (default: When random=T, the last population. When random=F, it is the last evaluated population)
- random: assuming random selection or selection according to their features (T: random selection, F: selection of good individuals)

(iii) Mating functions

Random mate:

cross(simEnv, nProgeny = 100, equalContribution = F, popID = NULL, popID2 = NULL)

The "cross" function conducts random mating among parents.

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results
- nProgeny: the number of progenies
- equalContribution: if T all individuals used the same number of times as parents, if F individuals chosen at random to be parents
- popID: the population ID to be crossed (default: the latest population)
- popID2: the population ID to be crossed with popID to make hybrids (default: no population is used for making hybrids)

If you use 'popID2', the input 'equalContribution' will be neglected.

Self-fertilize:

selfFertilize(simEnv, nProgenyPerInd = 1, popID = NULL)

The "selfFertilize" function implements inbreeding.

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results
- nProgenyPerInd: the number of progenies per one seed parent
- popID: the population ID to be self-fertilized (default: the latest population)

Doubled haploids:

doubledHaploid(simEnv, nProgeny = 100, popID = NULL)

The "doubledHaploid" function makes doubled haploids progeny.

- simEnv: the name of variable that you create in the function 'defineSpecies' to assigned the results
- nProgeny: the number of progenies
- popID: the population ID to be devided by meiosis and doubled (default: the latest population)

(iv) Results functions

Plot the results:

plotData(simEnv, ymax = NULL, add = F, addDataFileName = "plotData", popIDplot = NULL)

The function “plotData” draws a figure of the genotypic value through generations of breeding. In this figure, if user do not have any input to ‘popIDplot’, ‘generation’ is incremented when a new breeding population is generated by the mating functions (i.e., “cross”, “selfFertilize”, and “doubledHaploid”). The figure represents population means of each repeated simulation and the overall mean across repeats.

- simEnv: the name of variable that you create in the function ‘defineSpecies’ to assigned the results
- ymax: the maximum value of y-axis (default: the maximum value in the data)
- add: if T a new result will be added to an existing plot, if F a new plot will be drawn (default)
- addDataFileName: the name to save the summarized data for the next simulation with double-quotation, like "plot1_1". (default: "plotData")
- popIDplot: vector of the population IDs you want plotted

The plot is shown as ggplot style.

Save the results:

outputResults(simEnv, summarize = T, directory = NULL, saveDataFileName = "BSLoutput")

The function “outputResults” saves the results.

- simEnv: the name of variable that you create in the function ‘defineSpecies’ to assigned the results
- summarize: if T a result averaged over all the replications is saved, if F each replication's result is saved
- directory: the directory to which the output will be saved (Enclose the phrase in double quotation!) (default: the current directory)
- saveDataFileName: the file name to save the simulated data with double-quotation, like "result1_1". (default: "BSLoutput")

The simulation results (The output data was saved as BSLoutput.RData. After you load the data in R, you can find the data named as BSLoutput.)

Example 1
"Phenotypic selection & genomic selection"

Code:

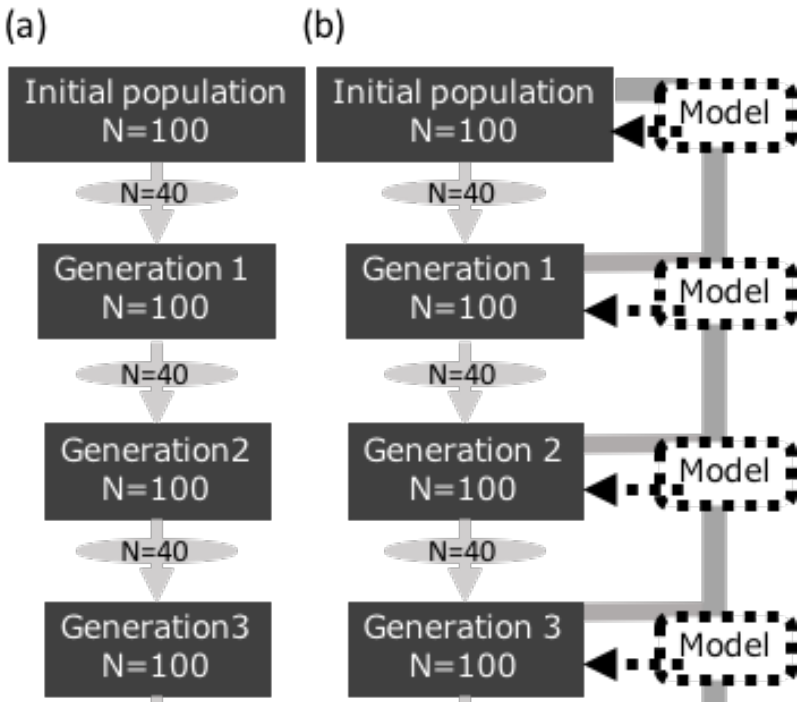
1. phenotypic selection (Fig. a)

```
simEnv <- defineSpecies(nSim = 5)     # popID 0
initializePopulation(simEnv)
phenotype(simEnv)
select(simEnv)      # popID 1
cross(simEnv)       # popID 2
phenotype(simEnv)
select(simEnv)      # popID 3
cross(simEnv)       # popID 4
phenotype(simEnv)
select(simEnv)      # popID 5
cross(simEnv)       # popID 6
plotData(simEnv)
simEnv1 <- simEnv
```

2. genomic selection (Fig. b)

```
simEnv <- defineSpecies(loadData="previousData")     # popID 0
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)     # popID 1
cross(simEnv)      # popID 2
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)     # popID 3
cross(simEnv)      # popID 4
phenotype(simEnv)
```

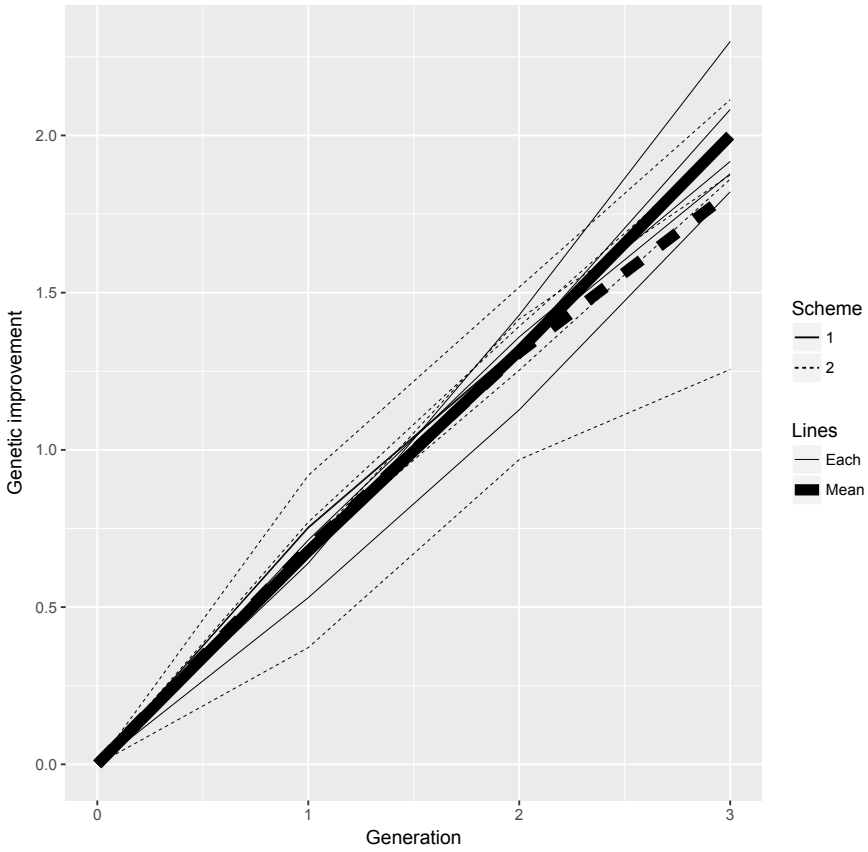
```
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)     # popID 5
cross(simEnv)      # popID 6
plotData(simEnv, add = T)
simEnv2 <- simEnv
```



Simulation result:

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

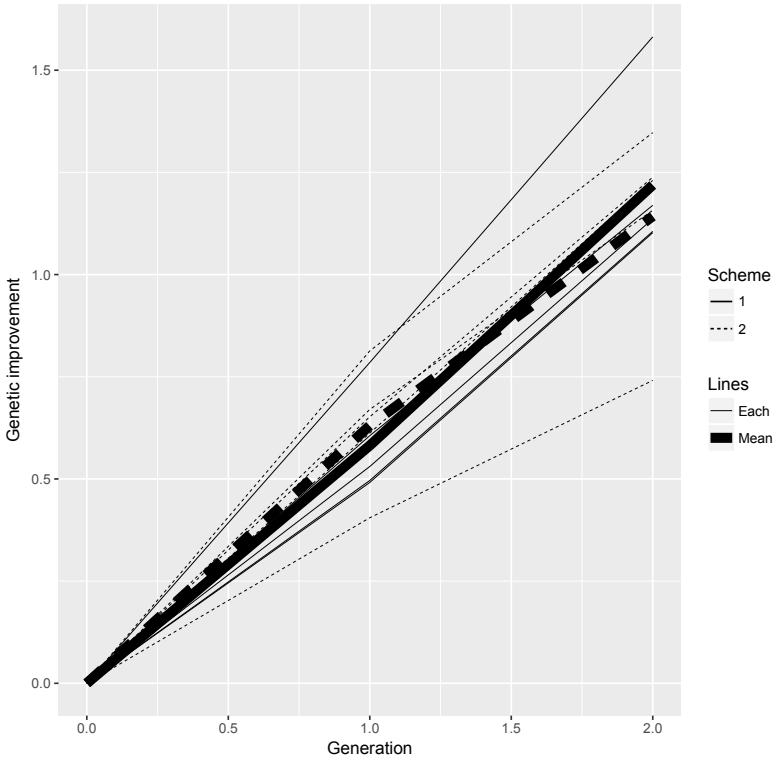
Bold lines represent the averaged values over all simulation trials (i.e., 5 simulation trials) per breeding scheme, and thin lines show the result of each simulation trial.



Here, let's think if the target species would be an autogamous species or species that could be propagated by clone, which requires select just one genotype showing the best performance. So, breeders would hope to look the performance of genotypes with better performance instead of all genotypes in a breeding population.

You can do as:

```
plotData(simEnv1, popIDplot = c(1,3,5))
plotData(simEnv2, add = T, popIDplot = c(1,3,5))
```

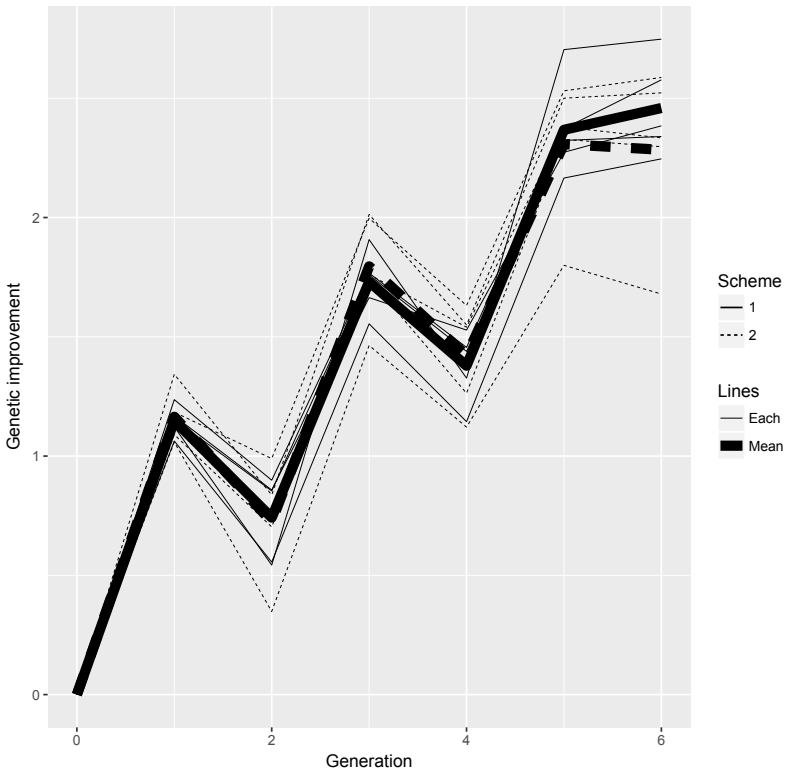


These commands show only selected genotypes from a population. Note that the initial values of genetic improvement are automatically adjusted to 0, and that the x-axis (Generation) is automatically labeled from 0 to the number of popID written in 'popIDplot'.

If you assigned all popID you created in the function "plotData" as:

```
plotData(simEnv1, popIDplot = 0:6)
plotData(simEnv2, add = T, popIDplot = 0:6)
```

The result figure represents all popID separately regardless that the population ID was derived by mating functions or the function "select". The result lines walk in zigzag because the unselected genotypes and selected genotypes in an identical breeding cycle are shown separately in this figure.



Example 2
"Phenotypic selection with different population size"

Code:

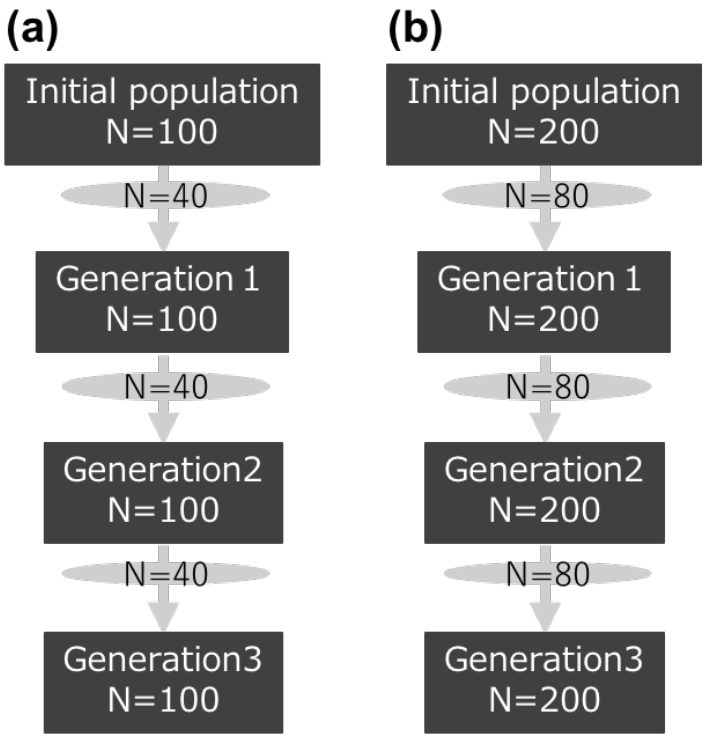
1. phenotypic selection with population size of 100 (Fig. a)

```
simEnv <- defineSpecies(saveDataFileName = "PSpopsize", nSim = 5)
initializePopulation(simEnv)
phenotype(simEnv)
select(simEnv)
cross(simEnv)
phenotype(simEnv)
select(simEnv)
cross(simEnv)
phenotype(simEnv)
```

```
select(simEnv)
cross(simEnv)
plotData(simEnv)
```

2. phenotypic selection with population size of 200 (Fig. b)

```
simEnv <- defineSpecies(load = "PSpopsize")
initializePopulation(simEnv, nPop = 200)
phenotype(simEnv)
select(simEnv, nSelect = 80)
cross(simEnv, nProgeny = 200)
phenotype(simEnv)
select(simEnv, nSelect = 80)
cross(simEnv, nProgeny = 200)
phenotype(simEnv)
select(simEnv, nSelect = 80)
cross(simEnv, nProgeny = 200)
plotData(simEnv, add = T)
```

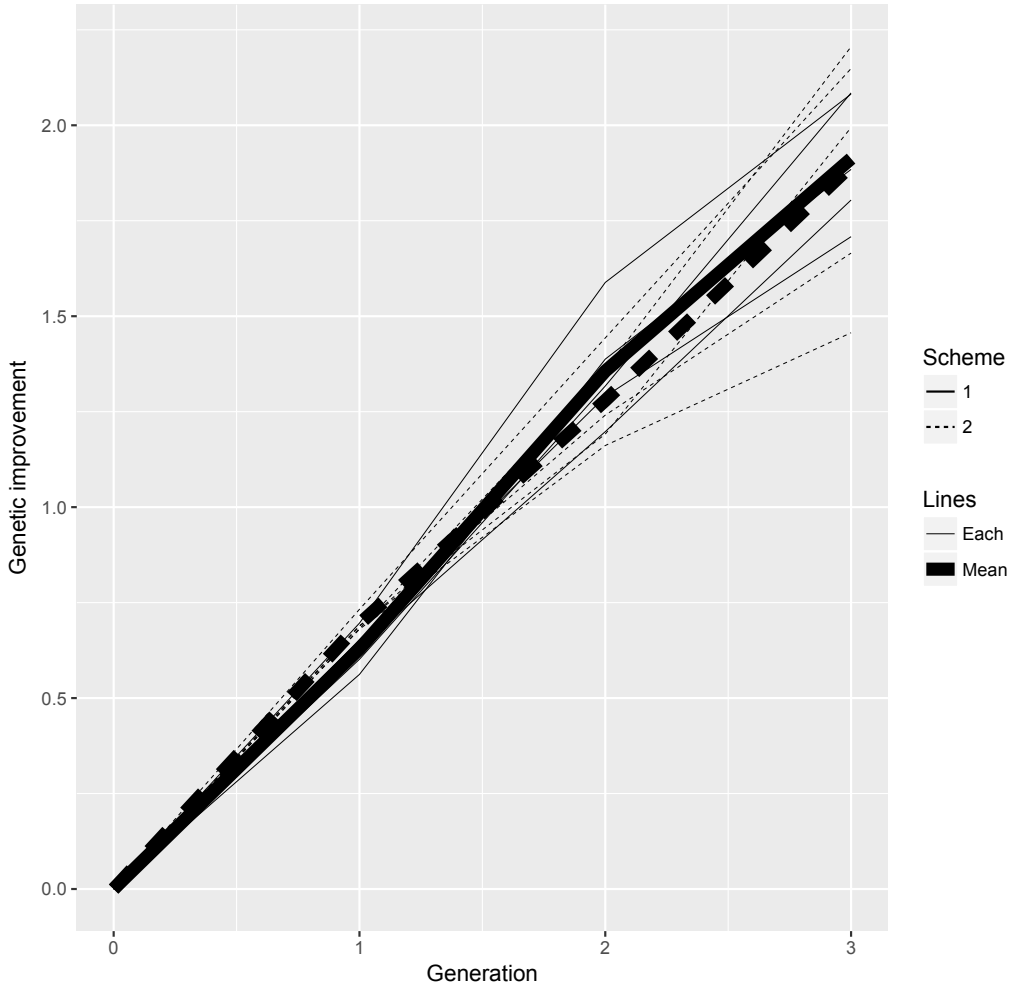


Note that we change the input if the function “select” also, because we would like the selection intensity to be set as 40% in both simulation settings.

Simulation result:

Scheme 1 is phenotypic selection with the population size 100, and Scheme 2 is phenotypic selection with the population size of 200.

Both simulation settings simulated the selection intensity of 40%.



Example 3
"Phenotypic selection with different selection intensity"

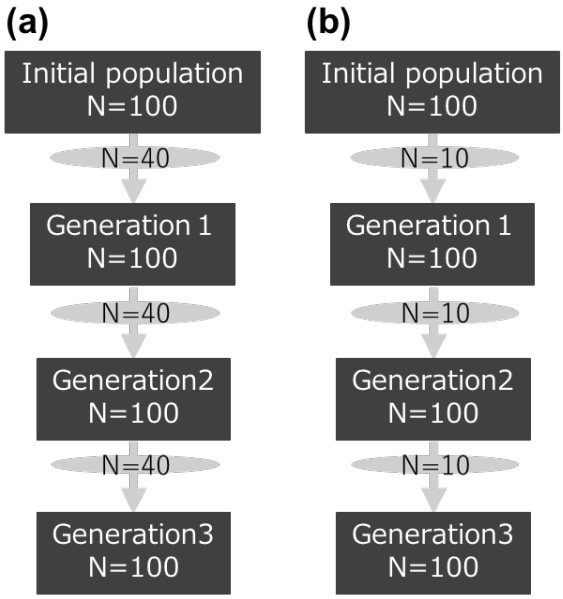
Code:

1. phenotypic selection in which upper 40% of genotypes are selected (Fig. a)

```
simEnv <- defineSpecies(saveDataFileName = "PSselectintensity", nSim = 5)
initializePopulation(simEnv)
phenotype(simEnv)
select(simEnv)
cross(simEnv)
phenotype(simEnv)
select(simEnv)
cross(simEnv)
phenotype(simEnv)
select(simEnv)
cross(simEnv)
plotData(simEnv)
```

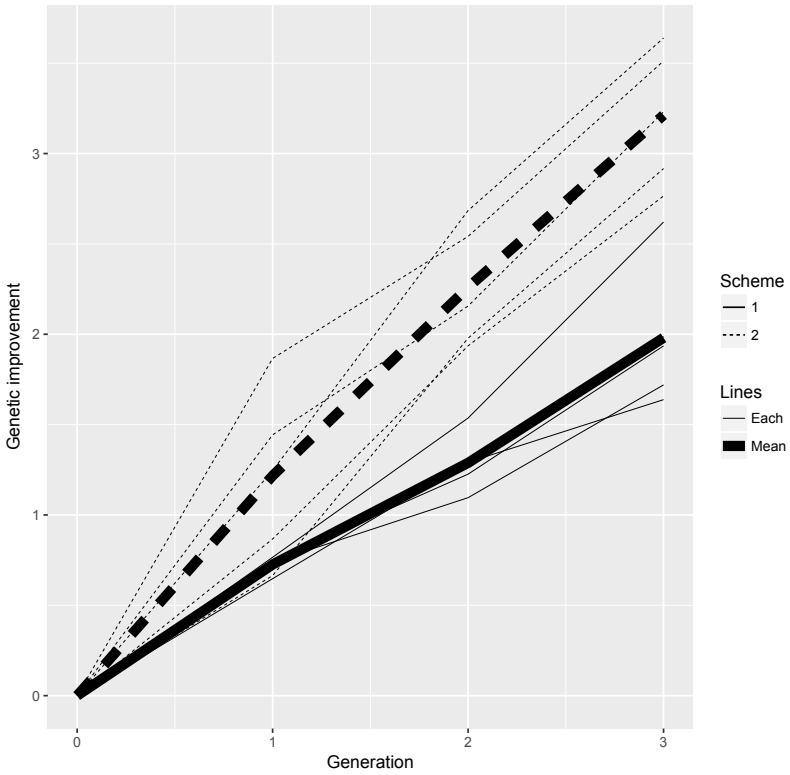
2. phenotypic selection in which upper 10% of genotypes are selected (Fig. b)

```
simEnv <- defineSpecies(load = "PSselectintensity")
initializePopulation(simEnv)
phenotype(simEnv)
select(simEnv, nSelect = 10)
cross(simEnv)
phenotype(simEnv)
select(simEnv, nSelect = 10)
cross(simEnv)
phenotype(simEnv)
select(simEnv, nSelect = 10)
cross(simEnv)
plotData(simEnv, add = T)
```



Simulation result:

Scheme 1 is phenotypic selection, in which upper 40% of genotypes are selected. Scheme 2 is phenotypic selection, in which upper 10% of genotypes are selected.



Example 4
"Genomic selection with and without model updating"

Code:

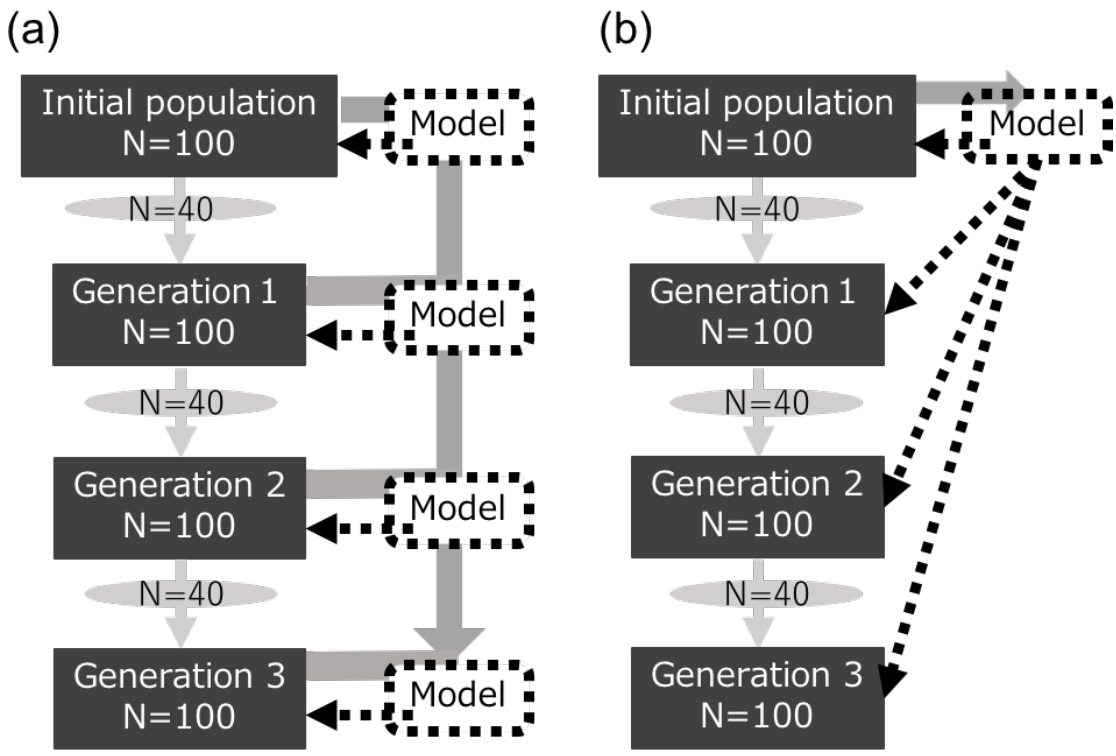
1. genomic selection with model updating each cycle (Fig. a)

```
simEnv <- defineSpecies(saveDataFileName = "GSmodelupdate", nSim = 5)
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
plotData(simEnv)
```

2. genomic selection without model updating each cycle (Fig. b)

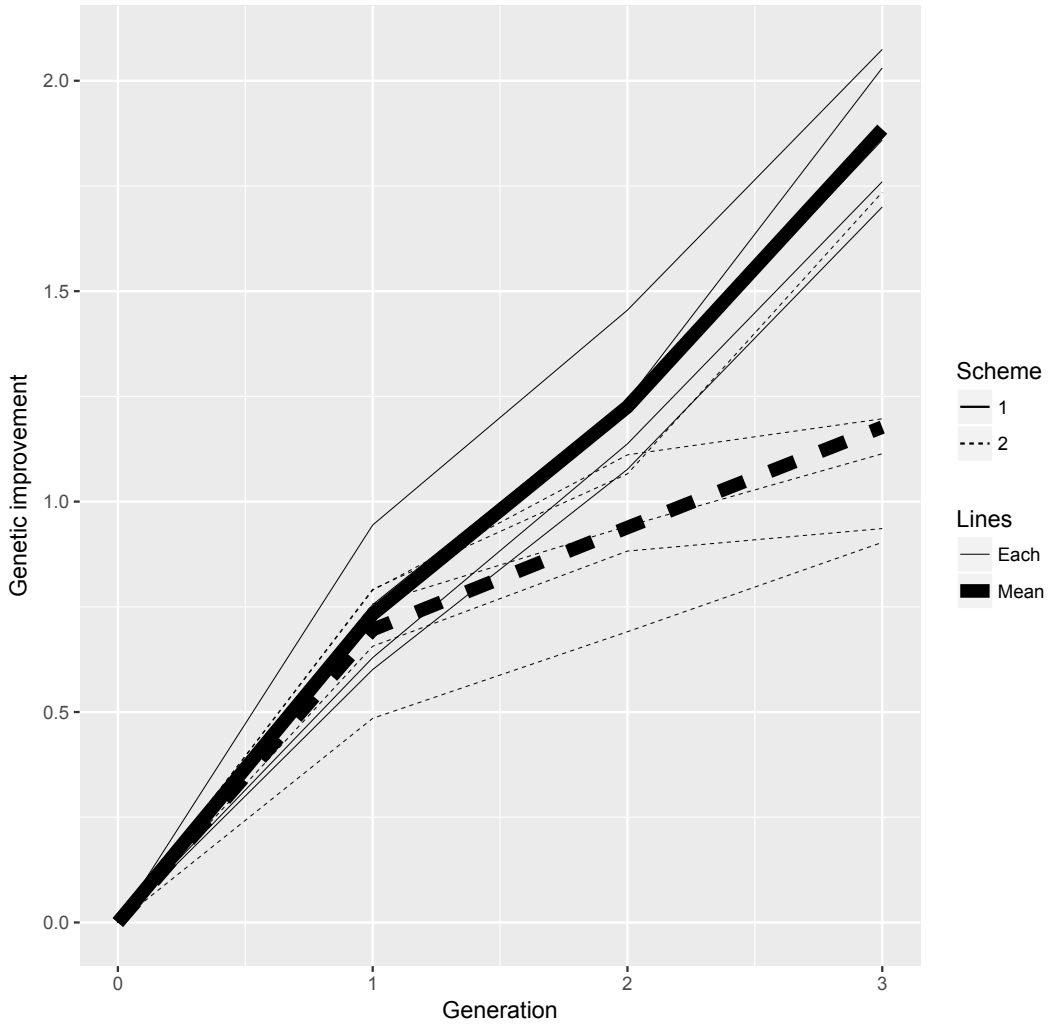
```
simEnv <- defineSpecies(load = "GSmodelupdate")
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
```

```
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
plotData(simEnv, add = T)
```



Simulation result:

Scheme 1 is genomic selection with model updating, in which all available data are used for training. Scheme 2 is genomic selection without model updating.



Example 5
"Genomic selection with model updating by recent data every two cycles"

Code:

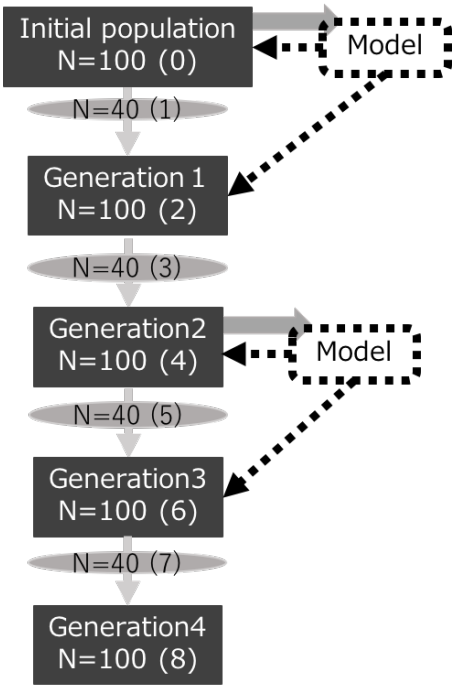
genomic selection with model updating

```
simEnv <- defineSpecies(load = "GSmodelupdate")
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
```

```
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv, trainingPopID = 4)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv, trainingPopID = c(4, 5))
select(simEnv)
cross(simEnv)
plotData(simEnv)
```

Here, we used previously created data in Example 4 by using the ‘load’ option in the function “defineSpecies”.

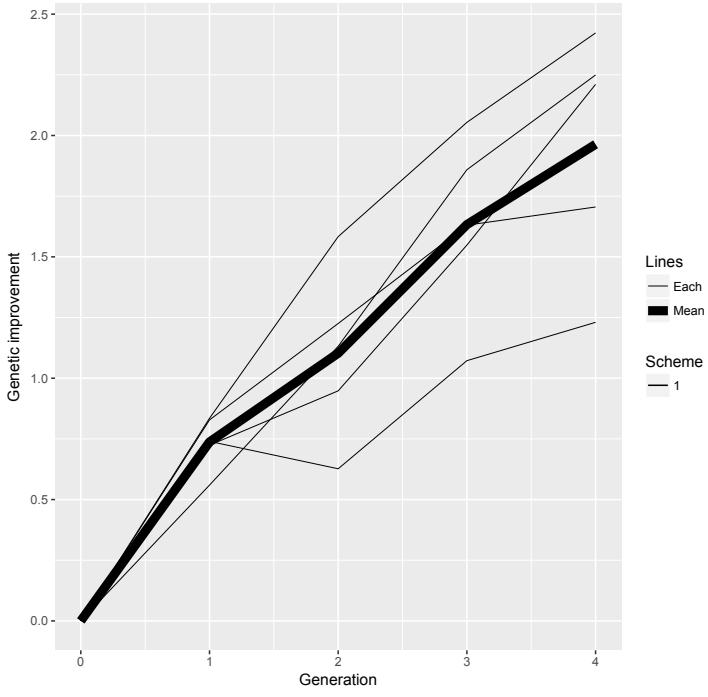
Note that the breeding population numbered to popID = 4 was divided into two population at third selection event by the function “select”. Thus, at the fourth selection event, the function “predictBreedVal” should have two population IDs (i.e., popID = 4 and 5) as the training population.



Numbers in parentheses represent the population ID.

Simulation result:

Genomic selection with model updating once per two cycles based on the recent phenotypic data.



Example 6
"Genomic selection using different number of markers"

Code:

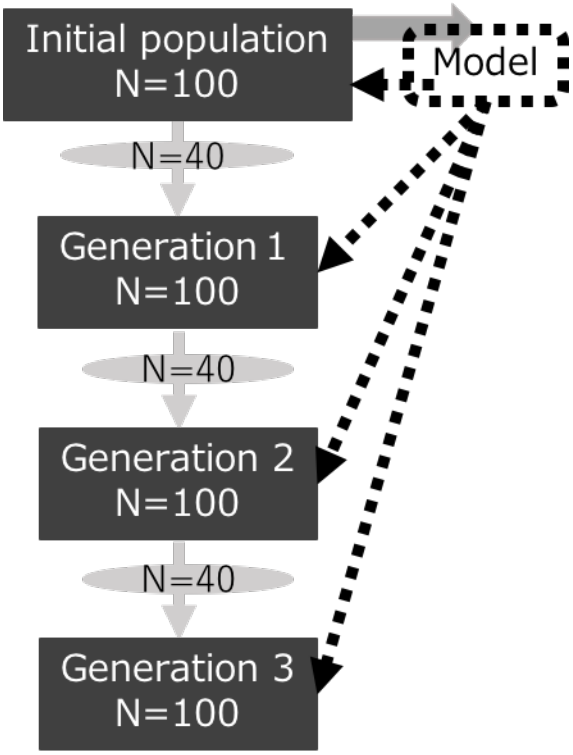
1. genomic selection with 1000 markers

```
simEnv <- defineSpecies(saveDataFileName = "1000m", nSim = 5)
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
plotData(simEnv)
```

2. genomic selection with 3000 markers

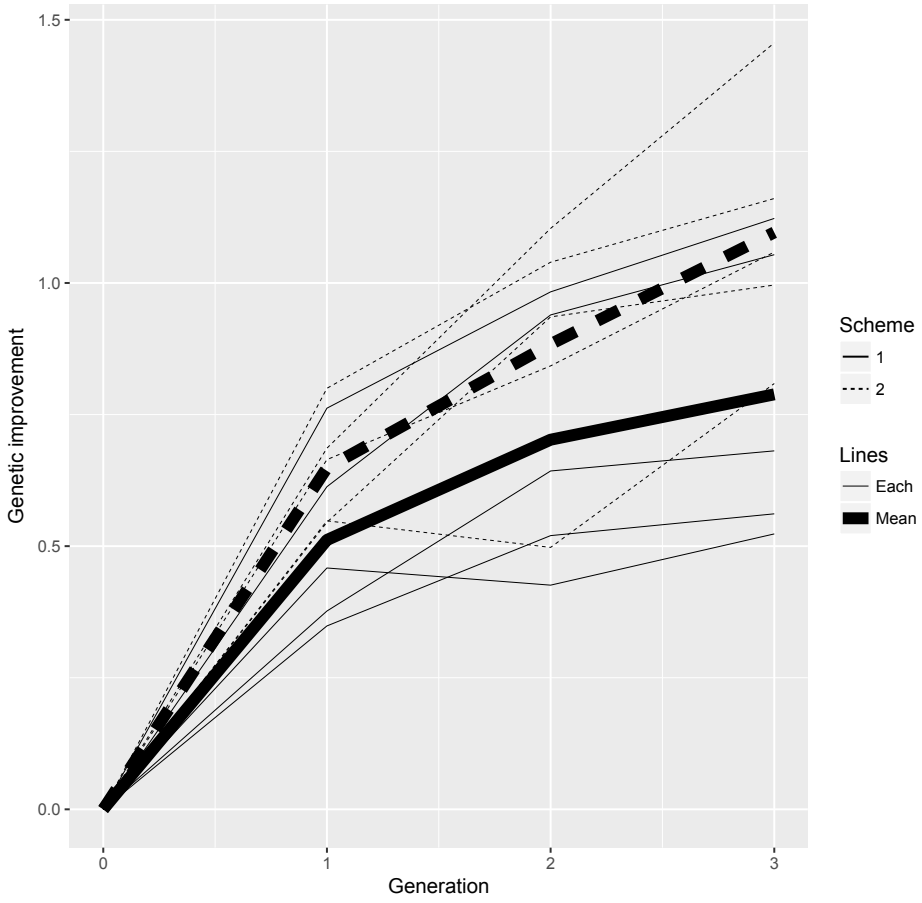
```
simEnv <- defineSpecies(saveDataFileName = "3000m", nSim = 5, nMarkers =
3000)
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
```

```
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
plotData(simEnv, add=T)
```



Simulation result:

Scheme 1 is genomic selection using 1000 markers. Scheme 2 is genomic selection using 3000 markers.



Example 7
"Genomic selection with different levels of linkage disequilibrium (LD)"

Code:

1. genomic selection for a population with high LD (effective population size is set as 100 in the base population)

```
simEnv <- defineSpecies(saveDataFileName = "ep50", nSim = 5, effPopSize = 50)
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
```

```
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
plotData(simEnv)
```

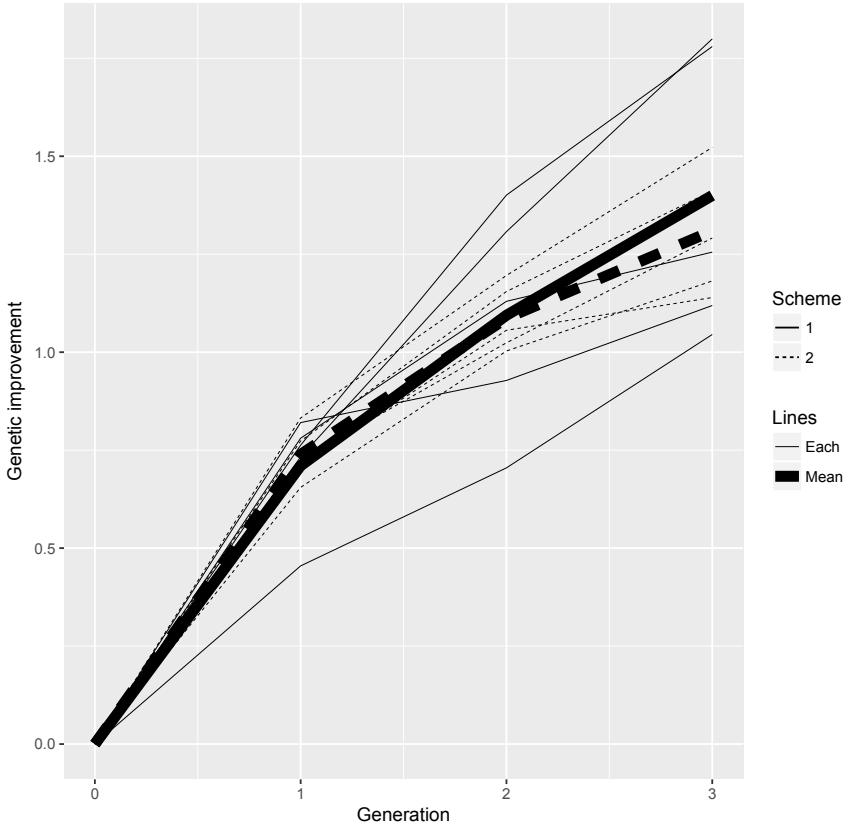
2. genomic selection for a population with low LD (effective population size is set as 200 in the base population)

```
simEnv <- defineSpecies(saveDataFileName = "ep200", nSim = 5, effPopSize = 200)
initializePopulation(simEnv)
phenotype(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
genotype(simEnv)
predictBreedVal(simEnv)
select(simEnv)
cross(simEnv)
plotData(simEnv, add=T)
```

The scheme is the same as the Example 6.

Simulation result:

Scheme 1 is genomic selection for a population with high LD. Scheme 2 is genomic selection for population with low LD.



Example 8
"making hybrids from two inbred populations for a trait controlled by dominant QTL"

Code:

Example 9
“Reciprocal recurrent selection for a trait controlled by dominant QTL”

Code:

Now, we are preparing this manual…