Breeding Scheme Language

Shiori Yabe, Hiroyoshi Iwata, and Jean-Luc Jannink

October 15, 2016

INDEX

Description……………………………………………………………………2
Introduction……………………………………………………………………...3
Required environment…………………………………………………………...4
Install BreedingSchemeLanguage………………………………………………….5
Describe breeding scheme………………………………………………………….6
Example 1……………………………………………………………………10
Example 2……………………………………………………………………13
Example 3……………………………………………………………………16
Example 4……………………………………………………………………19
Example 5……………………………………………………………………22
Example 6……………………………………………………………………25
Example 7……………………………………………………………………28

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:
____________________________________.

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 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 BreedingSchemeLanguage

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

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

Once you install the package, the first and second sentences 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!

Describe breeding scheme

Help

You can call help page by two types of code:

?FUNCTION

help(FUNCTION)

NOTE: FUNCTION should be replaced with the name of function you hope to ask.

Functions

- defineSpecies(loadData = NULL, saveDataFileName = "previousData", nSim = 1, nCore = 1, nChr = 7, lengthChr = 150, effPopSize = 100, nMarkers = 1000, nQTL = 50, propDomi = 0, nEpiLoci = 0)
- initializePopulation(nPop = 100, gVariance = 1)
- phenotype(errorVar = 1, popID = NULL)
- genotype()
- predictBreedVal(popID = NULL, trainingPopID = NULL)
- select(nSelect = 40, random = F, popID = NULL)
- cross(nProgeny = 100, equalContribution = F, popID = NULL)
- selfFertilize(nProgenyPerInd = 1, popID = NULL)
- doubledHaploid(nProgeny = 100, popID = NULL)
- plotData(ymax = NULL, add = F, addDataFileName = “plotData”)
- outputResults(summarize = T, directory = NULL, saveDataFileName = “BSLoutput”)

The parameters in the parenthesis represent the default inputs.

Functions to initiate simulations

The function “defineSpecies” requires four parameters to define the overall simulation settings (i.e., loadData, saveDataFileName, nSim, and nCore) and then seven parameters to define the genetic architecture of the species.

- loadData: If null, simulate new data, else attempt to load data with the name given.

- 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

The function “initializePopulation” creates a founder population for breeding:

- nPop: Size of the founder population
- gVariance: Genetic variance in the founder population

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)

The function “genotype” causes marker genotypes to become available for all individuals generated in the breeding scheme. It has no input.

The function “predictBreedVal” performs genomic prediction by using phenotype data and genotype data generated in the breeding scheme:

- 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)

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=T. When random=F, default is the last evaluated or predicted population)
- random: If T, individuals are selected at random. If F, the selection criterion depends on previous breeding activities: if the last activity was “phenotype”, then selection will be on phenotypes; if the last activity was “predictBreedVal”, then selection will be on predicted breeding values.

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)

The “selfFertilize” function implements inbreeding.

- nProgenyPerInd: Number of progenies derived from each selfed parent
- 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.
- 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

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"

Code:

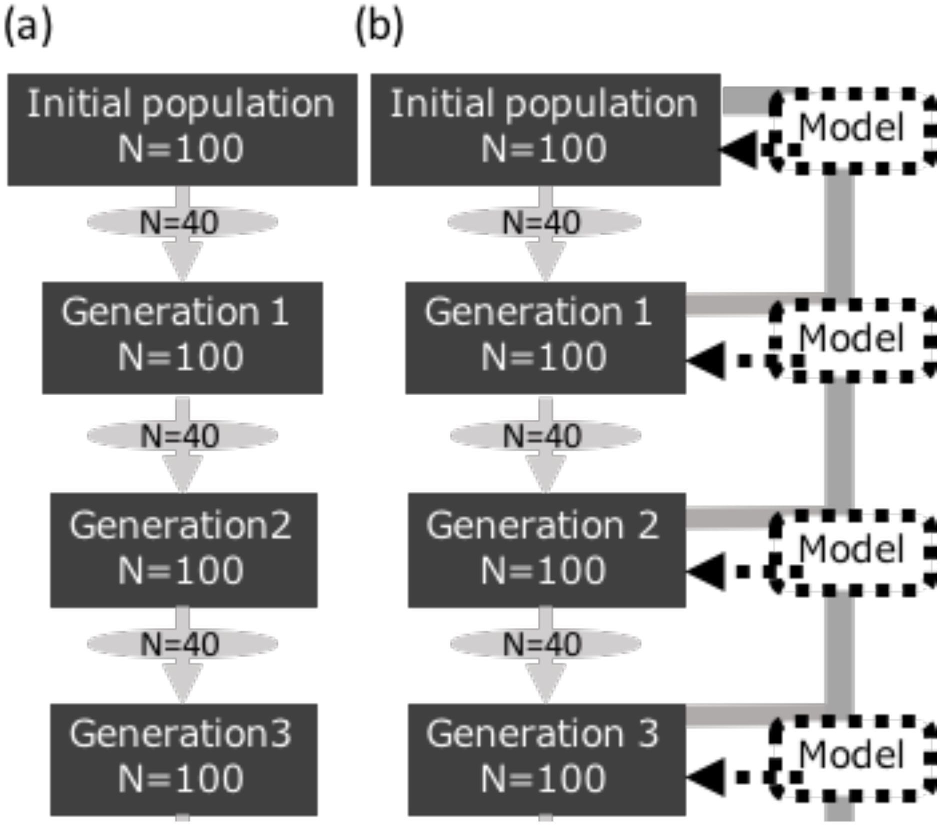
1. phenotypic selection (Fig. a)

```
defineSpecies(nSim = 5)
initializePopulation()
phenotype()
select()
cross()
phenotype()
select()
cross()
phenotype()
select()
cross()
plotData()
```

2. genomic selection (Fig. b)

```
defineSpecies(loadData="previousData")
initializePopulation()
phenotype()
genotype()
predictBreedVal()
select()
cross()
phenotype()
genotype()
predictBreedVal()
select()
cross()
phenotype()
genotype()
```

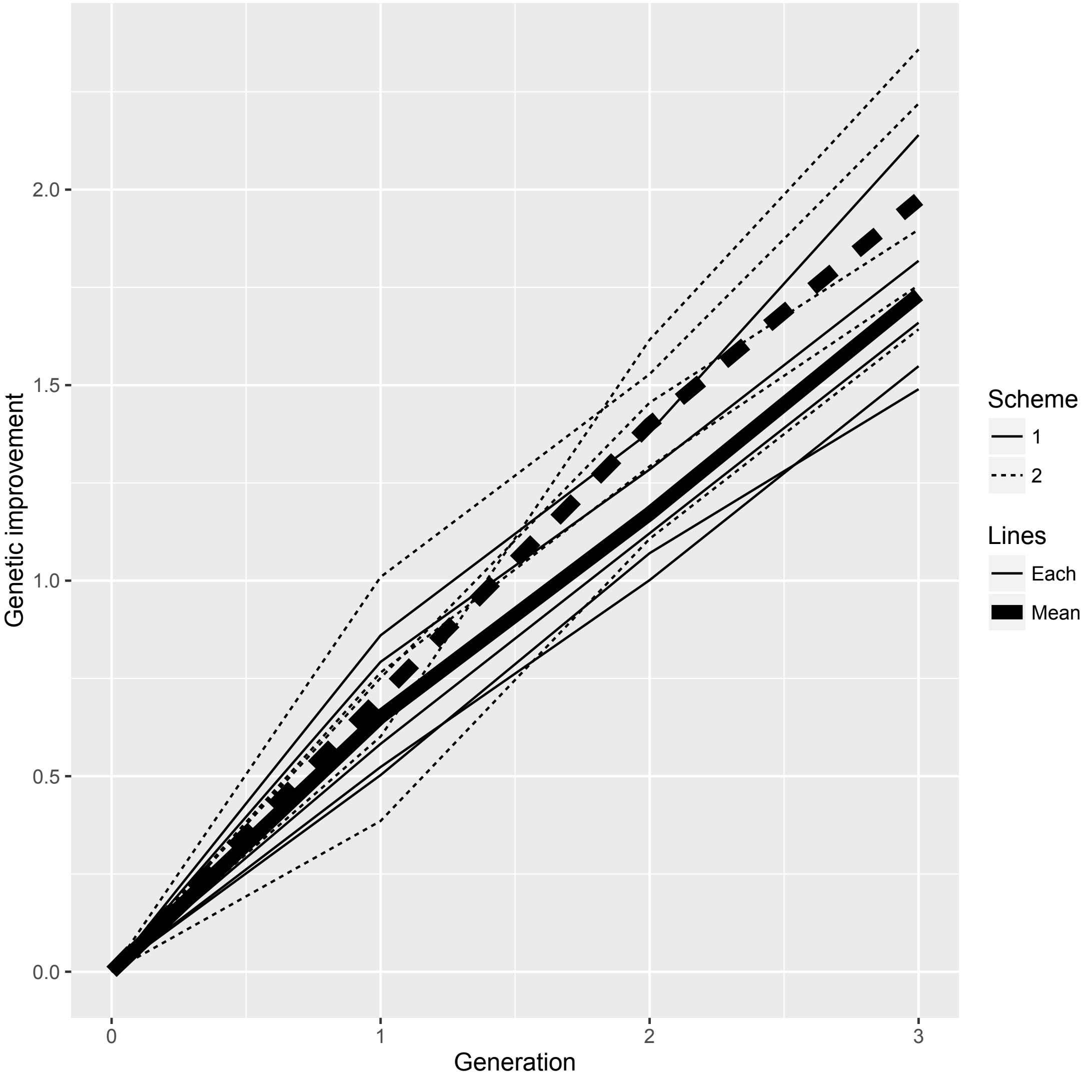
```
predictBreedVal()
select()
cross()
plotData(add=T)
```


(a)
(b)
Initial population N=100
N=40
Generation 1 N=100
N=40
Generation2 N=100
N=40
Generation3 N=100
Initial population N=100
Model
N=40
Generation 1 N=100
Model
N=40
Generation 2 N=100
Model
N=40
Generation 3 N=100
Model

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 times of simulations), and each thin line is the result of a simulation trial.


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

Example 2
“Phenotypic selection with different population size”

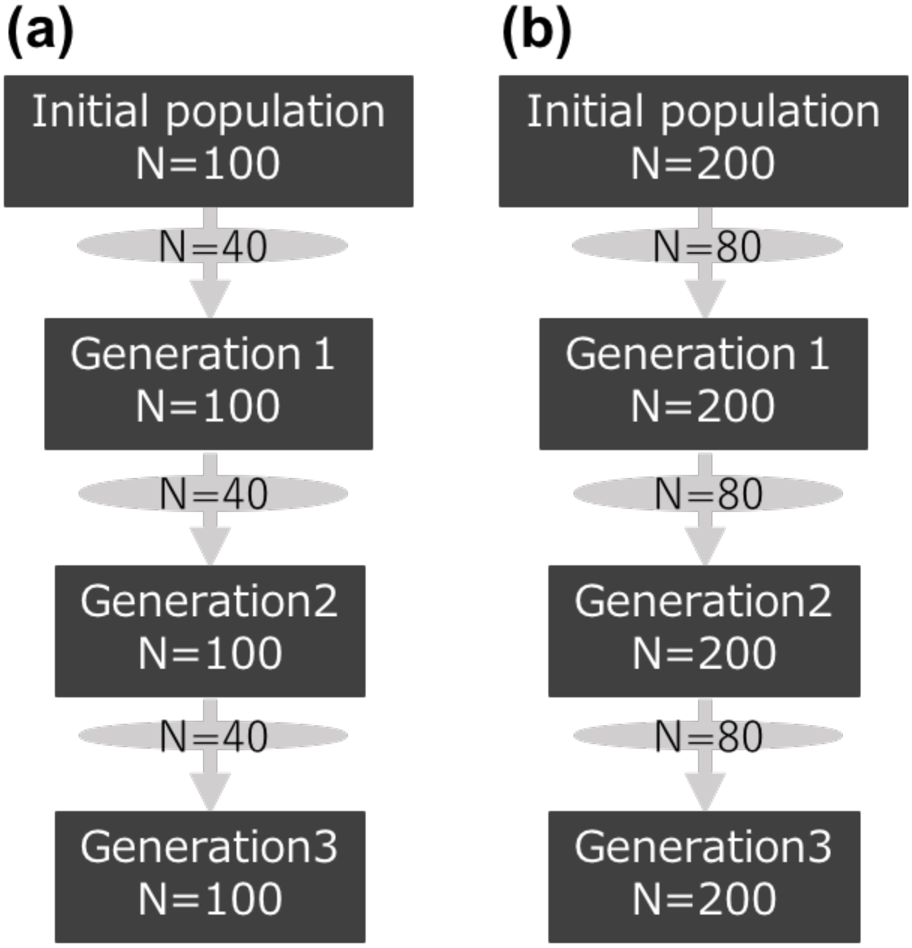
Code:

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

```
defineSpecies(saveDataFileName = "PSpopsize", nSim = 5)
initializePopulation()
phenotype()
select()
cross()
phenotype()
select()
cross()
phenotype()
select()
cross()
plotData()
```

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

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


(a)
Initial population
N=100
N=40
Generation 1
N=100
N=40
Generation2
N=100
N=40
Generation3
N=100
(b)
Initial population
N=200
N=80
Generation 1
N=200
N=80
Generation2
N=200
N=80
Generation3
N=200

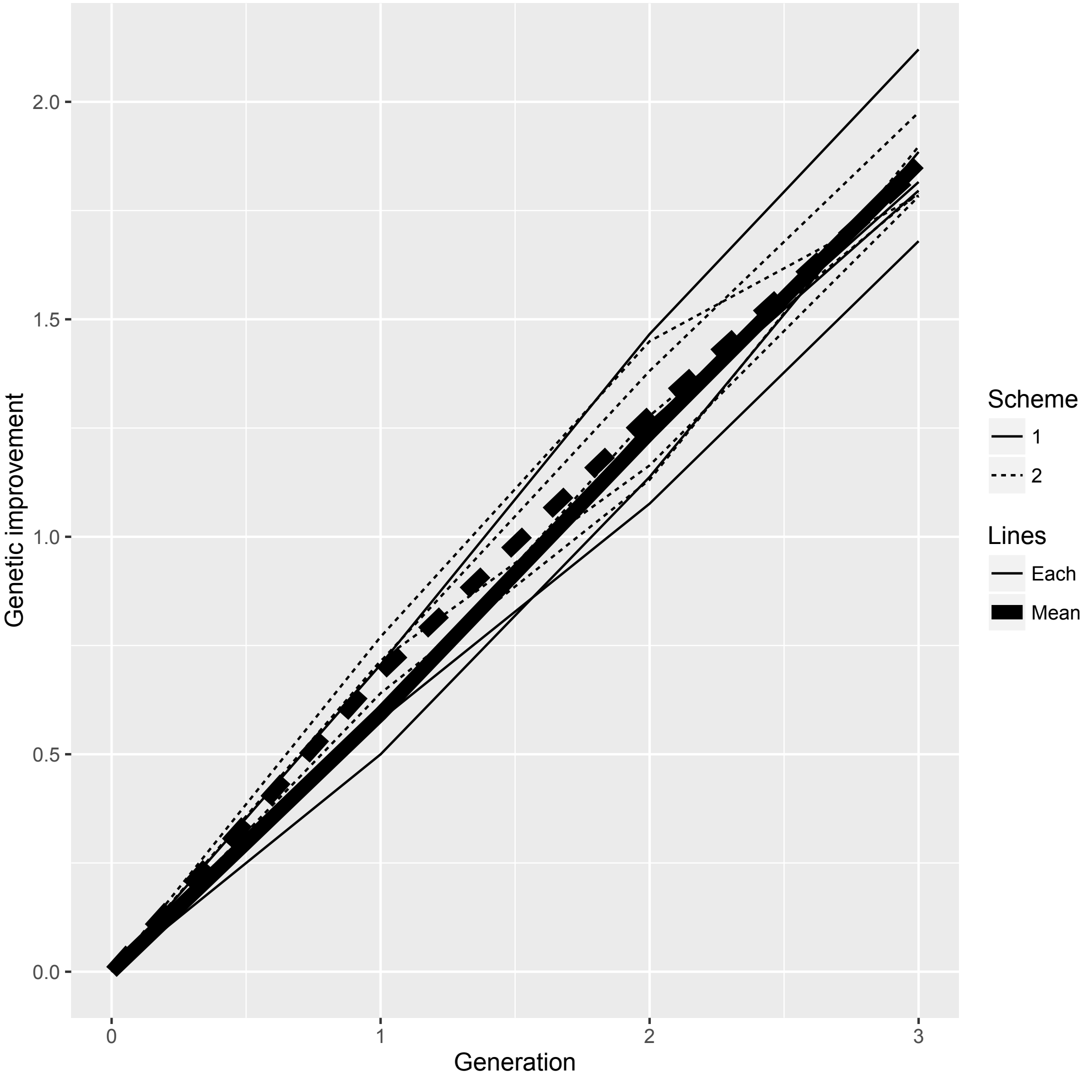
Note that we change the input if the function “select” also, because we would like the selection intensity to be set as 40% in both simulation settings.

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%.

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



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

Example 3
“Phenotypic selection with different selection intensity”

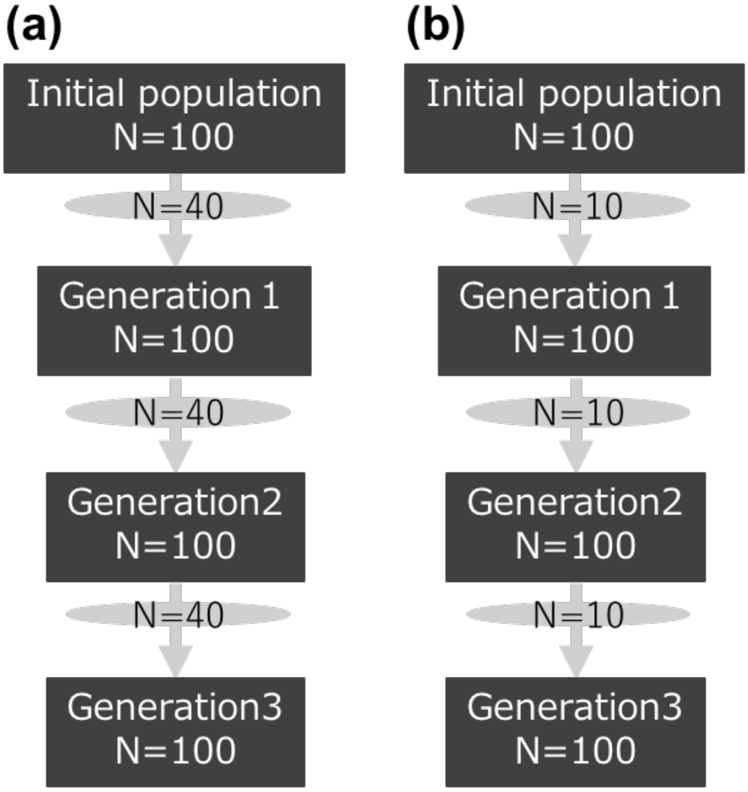
Code:

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

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

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

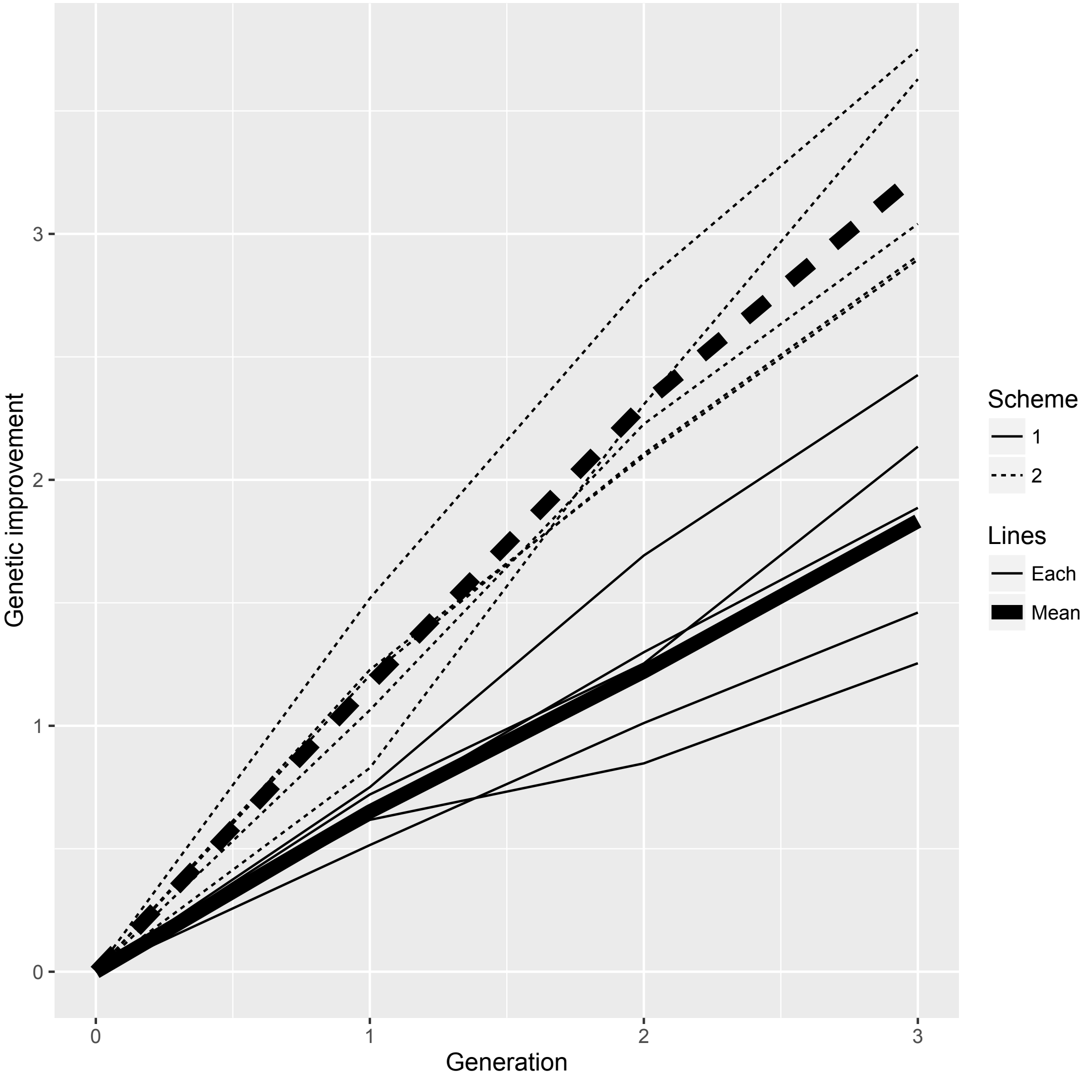
```
defineSpecies(load = "PSselectintensity")
initializePopulation()
phenotype()
select(nSelect = 10)
cross()
phenotype()
select(nSelect = 10)
cross()
phenotype()
select(nSelect = 10)
cross()
plotData(add = T)
```


(a)
Initial population N=100
N=40
Generation 1 N=100
N=40
Generation2 N=100
N=40
Generation3 N=100
(b)
Initial population N=100
N=10
Generation 1 N=100
N=10
Generation2 N=100
N=10
Generation3 N=100

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.

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


Genetic improvement
0
1
2
3
Generation
0
1
2
3
Scheme
1
2
Lines
Each
Mean

Example 4
“Genomic selection with and without model updating”

Code:

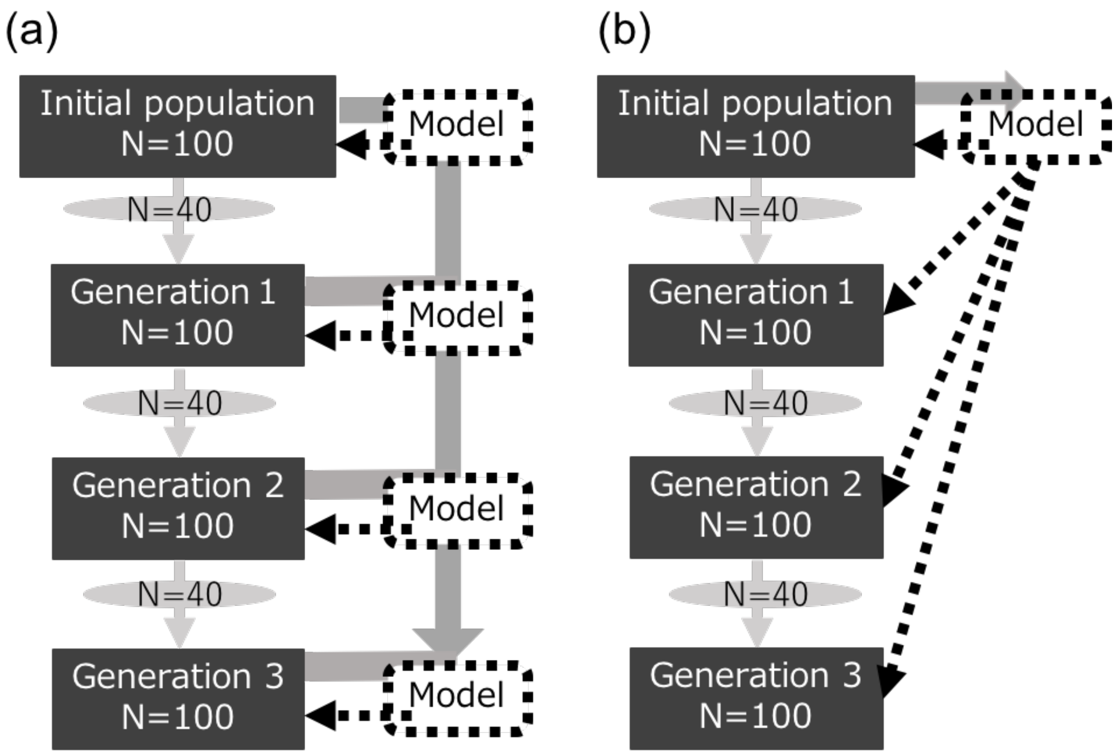
1. genomic selection with model updating each cycle (Fig. a)

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

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

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

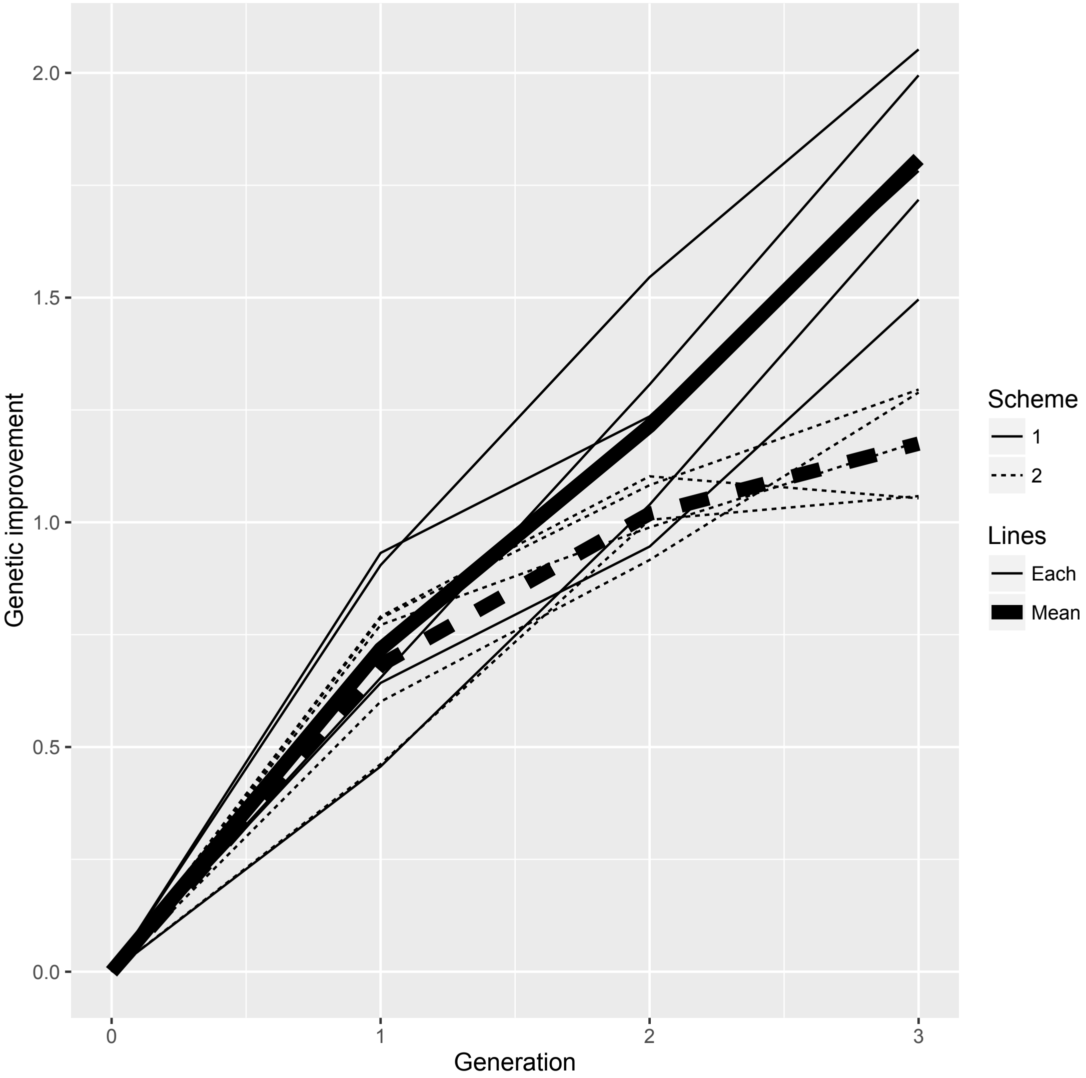
```
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
cross()
plotData(add=T)
```


(a)
Initial population N=100
Model
N=40
Generation 1 N=100
Model
N=40
Generation 2 N=100
Model
N=40
Generation 3 N=100
Model
(b)
Initial population N=100
Model
N=40
Generation 1 N=100
N=40
Generation 2 N=100
N=40
Generation 3 N=100

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.

Bold lines represent the mean values of simulation trials (in this case, 5 times of simulations), 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

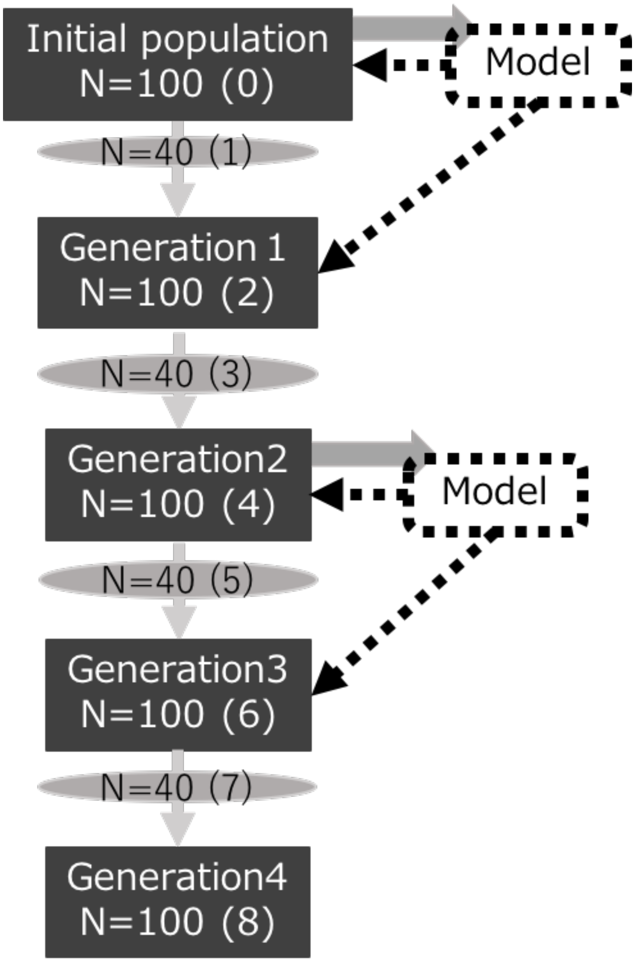
Example 5
“Genomic selection with and without model updating”

Code:

genomic selection with model updating

```
defineSpecies(load = "GSmodelupdate")
initializePopulation()
phenotype()
genotype()
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
cross()
phenotype()
genotype()
predictBreedVal(trainingPopID = 4)
select()
cross()
genotype()
predictBreedVal(trainingPopID = c(4, 5))
select()
cross()
plotData()
```

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

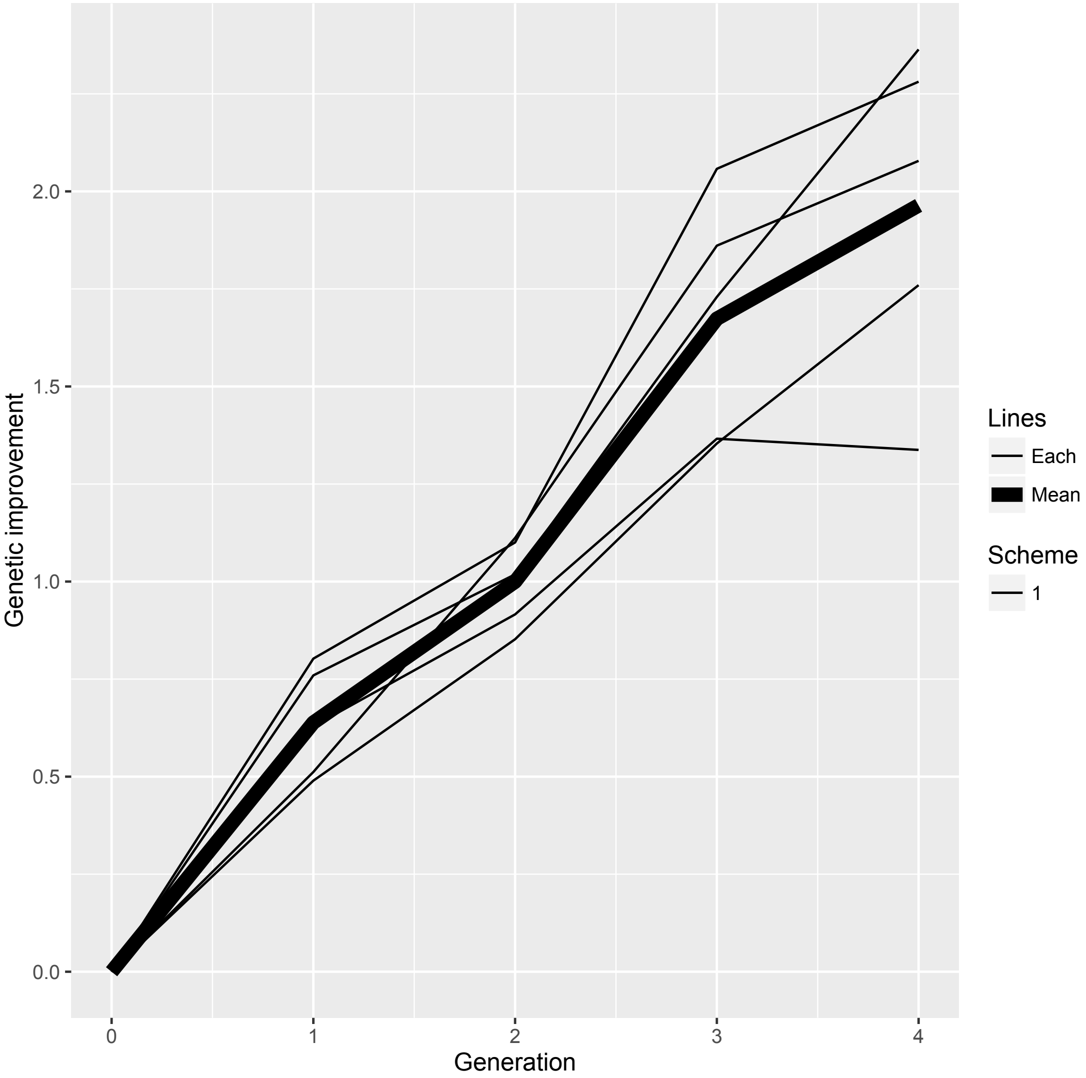

Initial population
N=100 (0)
Model
N=40 (1)
Generation 1
N=100 (2)
N=40 (3)
Generation2
N=100 (4)
Model
N=40 (5)
Generation3
N=100 (6)
N=40 (7)
Generation4
N=100 (8)

Numbers in parentheses represent the population ID.

Simulation result:

Genomic selection without model updating once per two cycles.

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


Genetic improvement
0.0
0.5
1.0
1.5
2.0
0
1
2
3
4
Generation
Lines
Each
Mean
Scheme
1

Example 6
“Genomic selection using different number of markers”

Code:

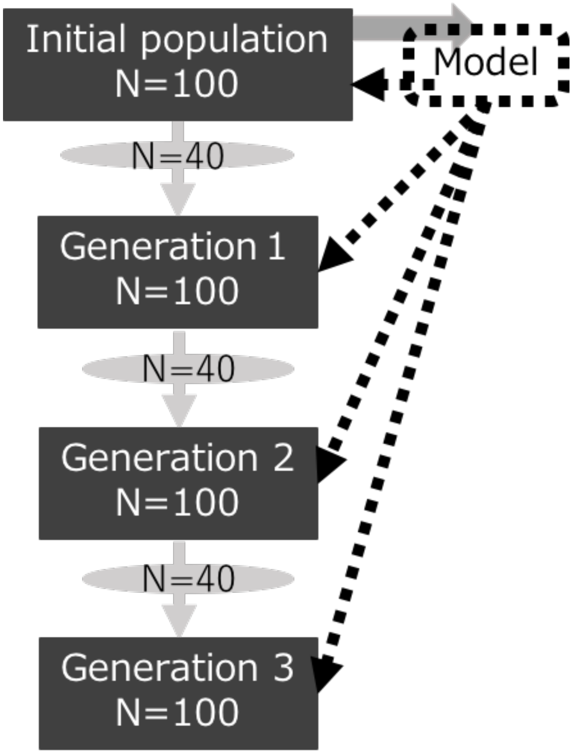
1. genomic selection with 1000 markers

```
defineSpecies(nSim = 5)
initializePopulation()
phenotype()
genotype()
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
cross()
plotData()
```

2. genomic selection with 3000 markers

```
defineSpecies(nSim = 5, nMarkers = 3000)
initializePopulation()
phenotype()
genotype()
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
```

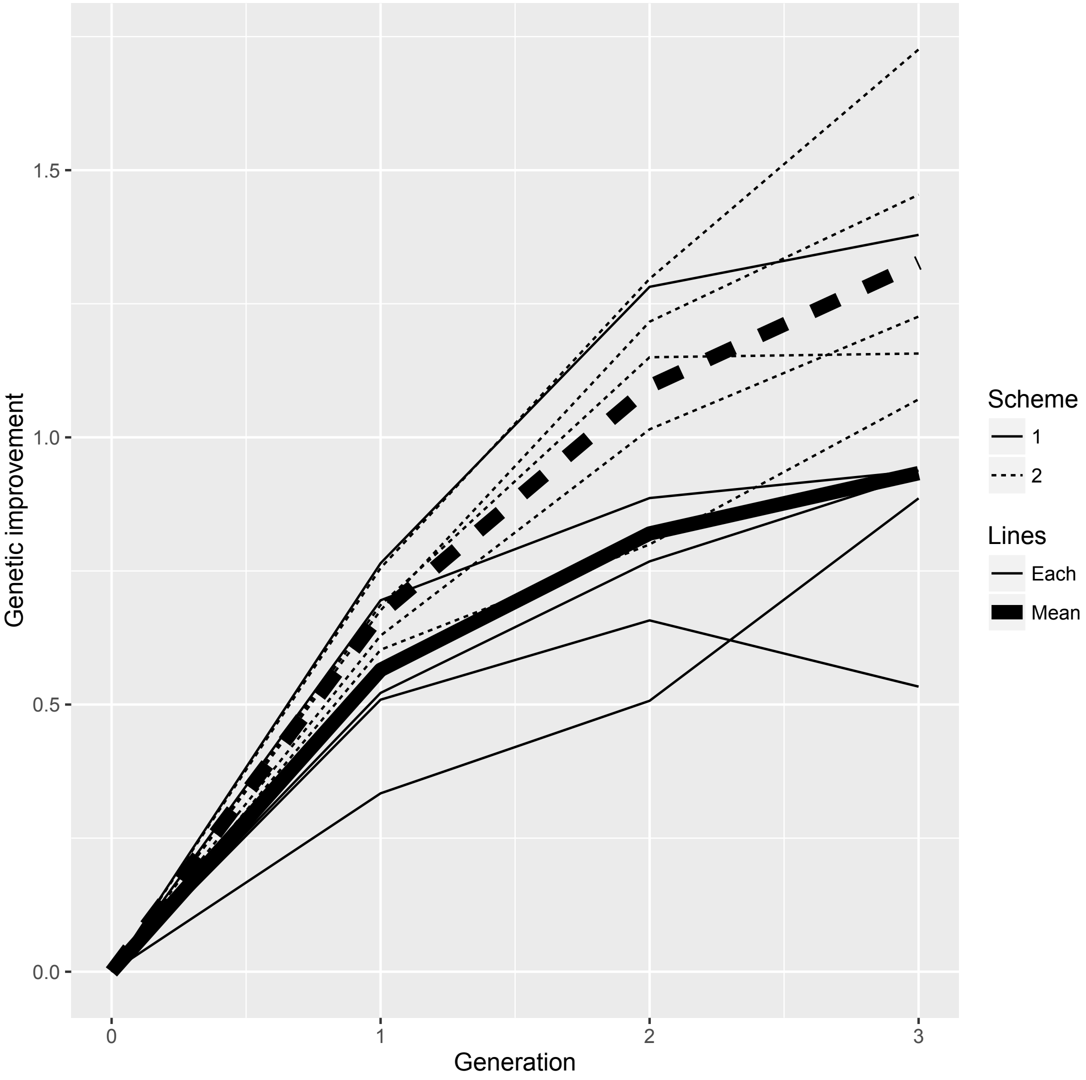
```
cross()
genotype()
predictBreedVal()
select()
cross()
plotData(add=T)
```


Initial population
N=100
Model
N=40
Generation 1
N=100
N=40
Generation 2
N=100
N=40
Generation 3
N=100

Simulation result:

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

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


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

Example 7

“Genomic selection with different levels of linkage disequilibrium (LD)”

Code:

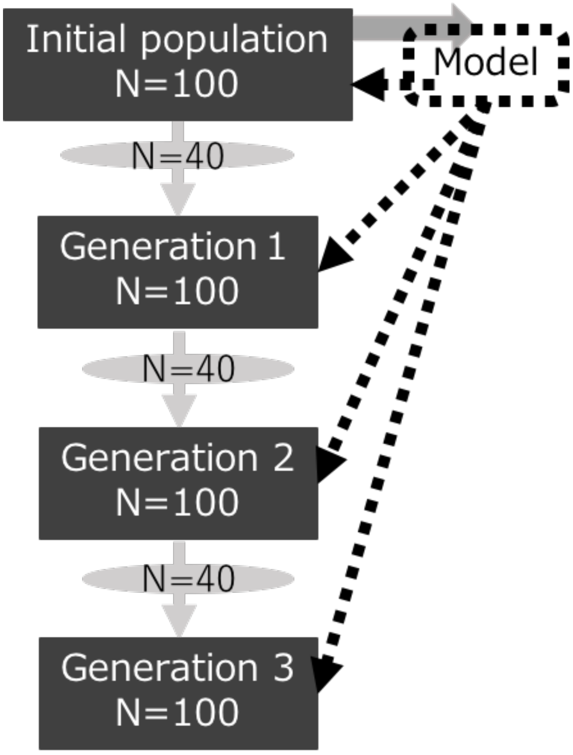
1. genomic selection for a population with high LD (effective population size is set as 100 in the base population)

```
defineSpecies(nSim = 5, effPopSize = 50)
initializePopulation()
phenotype()
genotype()
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
cross()
plotData()
```

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

```
defineSpecies(nSim = 5, effPopSize = 200)
initializePopulation()
phenotype()
genotype()
predictBreedVal()
select()
cross()
```

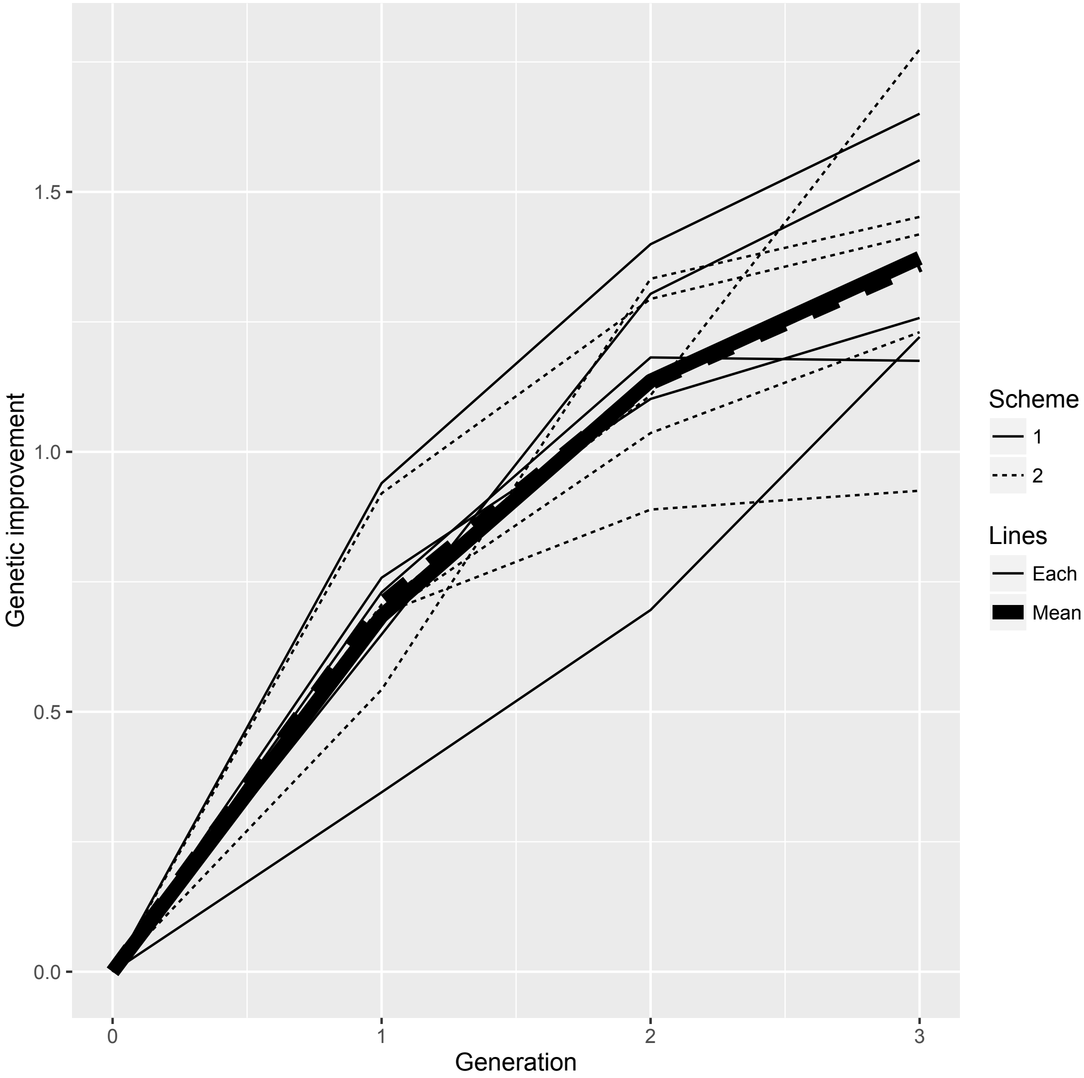
```
genotype()
predictBreedVal()
select()
cross()
genotype()
predictBreedVal()
select()
cross()
plotData(add=T)
```


Initial population
N=100
Model
N=40
Generation 1
N=100
N=40
Generation 2
N=100
N=40
Generation 3
N=100

Simulation result:

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

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


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

Now, we are preparing this manual…