Simulation via AlphaSimR

MaoHuang

4/27/2022

Objectives

Class1:
We will learn different `AlphaSimR` functions
We will write chunks of codes to simulate different scenarios

Class2:
We will write whole breeding pipeline
We will evaluate different results as we change simulation parameters

Practice reproducibility

Reading materials for
* Writing scripts via R Markdown (https://rmarkdown.rstudio.com/authoring_quick_tour.html)
* Create project in Rstudio (https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects)
* Github (https://r-pkgs.org/git.html#git-rstudio) commit and push

General steps to use `AlphaSimR`

1. Plan out your breeding program
2. Set up founder haplotypes
3. Set up simulation parameters
4. Modeling the breeding program
5. Evalute your simulation results

Script setup

Install packages, set the random seed

*** Note: `AlphaSimR` generates many random numbers. Set up random seed and your will always get the same results. Please change it when you run.

```
## echo=FALSE in R markdown, if you do not want to show this chunk of code
random_seed <- 12345
set.seed(random_seed)

## Install the package if haven't yet
packages_used <- c("AlphaSimR")
packages_installed <- installed.packages()  # what's been installed
for (package in packages_used){
  if (!(package %in% packages_installed[,"Package"])){
    install.packages(package)
  } else {
  library(package,character.only=T)
      }
}
```

```
## Loading required package: R6
```

Set up `AlphaSimR` parameters

AlphaSimR requires initial simulation parameters to be set up for haplotypes

```
nFounders <- 10
nChr <- 2
segSites <- 500   # Number of segrerating sites per chormosome this =nLoci???
nQTL<- 100   #Number of QTL per chormosome this may be passed down randomly????
```

The initial number of founders `nFounders` =10
The number of chromosomes for the species is `nChr` = 2
The number of segregating sites on each chromosome is `segSites` = 500
The number of QTLs (causal variant) on each chromosome is `nQTL` = 100

Some common functions in `AlphaSimR`

How to create founders

Note: Example modified from AlphaSimR (https://cran.r-project.org/web/packages/AlphaSimR/AlphaSimR.pdf), Chris Gynor example (https://cran.r-project.org/web/packages/AlphaSimR/vignettes/intro.html) and Huang (https://academic.oup.com/g3journal/article/12/3/jkac003/6511442?login=true) et al. 2022

- 1st, you produce founder haplotypes from a coalescent simulation as part of `AlphaSimR`. We use `runMacs2()` (Markovian Coalescent Simulator, Chen (https://genome.cshlp.org/content/19/1/136) et al. 2008). It simulates bi-allelic genome sequences according to a population demographic history. This function allows the user to specify one of several predefined population histories or supply their own population history. A list of currently available population histories can be found in the runMacs help document. You can also import your own haplotypes being generated in another software package or taken directly from real marker data rwith the newMapPop function. Note: you can also use `quickHaplo()` function to generate the initial haplotypes. It generates haplotypes by randomly sampling 1s and 0s. It is equivalent to modeling a population in Hardy-Weinberg equilibrium with allele frequencies of 0.5. But it is only recommended for prototype coding.

- 2nd, you define simulation parameters that connects the founder haplotype genotypic and phenotypic variation
- 3rd, your make the diploid founders from their haplotypes

Example:

Create founder haplotypes using runMacs2(); Default effective population size Ne is 100

```
founderHap<-runMacs2(nInd=nFounders,nChr=nChr,segSites=segSites)
```

Create New global simulation parameters from founder haplotypes using `SimParam$new()'

```
SP<-SimParam$new(founderHap)
```

Careful with your naming, avoid using "SP" in any other places: `AlphaSimR` other functions searches your R global environment for the object named "SP" by default .

Add additive trait architecture. This trait is conroled by nQTL=100 QTLs per chromosome. Genetic mean is 0 and genetic variance is 1.

`addTraitA()` radomly assign eligible QTLs for additive effects ONLY. You can simulation other effects dominance ("D"), epistasis ("E"), and genotype-by-environment ("G"). You can also simulate more than one trait —If simulating more than one trait, all traits will be pleiotrophic with correlated additive effects

```
SP$addTraitA(nQtlPerChr=nQTL,mean=0,var=1)
```

Design a SNP chip that can be used later (randomly assign eligible SNPs to a SNPchip)

```
SP$addSnpChip(nSnpPerChr=200)  # Number of SNP markers per chormosome  ### Note, This c

# Changes how sexes are determined in the simulation. The value "yes_sys" will systemat
#SP$setSexes("yes_sys")

# Track the population records
SP$setTrackRec(TRUE)
```

```
#3. Create a new populations of 5 individuals
founders<- newPop(founderHap, simParam=SP)
```

Let's take a look at the founders

```
str(founders)
```

```
## Formal class 'Pop' [package "AlphaSimR"] with 16 slots
##   ..@ id     : chr [1:10] "1" "2" "3" "4" ...
##   ..@ mother : chr [1:10] "0" "0" "0" "0" ...
##   ..@ father : chr [1:10] "0" "0" "0" "0" ...
##   ..@ gender : chr [1:10] "" "" "" "" ...
##   ..@ nTraits: int 1
##   ..@ gv     : num [1:10, 1] -0.0307 0.027 -1.512 1.9414 -0.1447 ...
##   ..@ pheno  : num [1:10, 1] NA NA NA NA NA NA NA NA NA NA
##   ..@ ebv    : num[1:10, 0 ]
##   ..@ gxe    :List of 1
##   .. ..$ : NULL
##   ..@ fixEff : int [1:10] 1 1 1 1 1 1 1 1 1 1
##   ..@ reps   : num [1:10] 1 1 1 1 1 1 1 1 1 1
##   ..@ nInd   : int 10
##   ..@ nChr   : int 2
##   ..@ ploidy : int 2
##   ..@ nLoci  : int [1:2] 500 500
##   ..@ geno   :List of 2
##   .. ..$ : raw [1:63, 1:2, 1:10] 64 94 f3 05 ...
##   .. ..$ : raw [1:63, 1:2, 1:10] 1b 11 19 03 ...
##   .. ..- attr(*, "dim")= int [1:2] 2 1
```

Each individual has ID in `@id`, they are in numeric form. The `@mother` and `@father` IDs are referring to diploid parents, which are not available and are all in 0s, because this population is made from haplotypes. It gives `@gv` which is *genetic value* or the *true breeding value* calculated for the trait simulated from `addTraitA()`. You can access it by `gv(founders)` or `founders@gv`

How to access the population information

The `founders` is now your founder population object. Treat it like a special vector. Now you can extract information. For example, you can sort the individuals based on the order of their *genetic values* and pick the top *n* of them.

```
# The gv is in a vector
GV<-founders@gv

# Sort individuals. Here SortID is also a population object (the sorted founders).
Sortfounders<-founders[order(-GV)]

# Pick top n of them
n<-3
TopID<-Sortfounders[1:n]
### str(SortID)

# OR If you want to look at all of the individuals in a dataframe format
IDGV<-as.data.frame(cbind(Sortfounders@id,Sortfounders@gv))
names(IDGV)<-c("ID","GV")
```

How to make some progenies

Outcrossing with `makeCross` and Selfing with `self`

- For example, we can make some inbreds. Selfing each individual in the population with `self()` . You can control the total number of selfed progenies per parent with the parameter `nProgeny` . You can do single seed descent using this function with `nProgeny=1` . This function only works when sexes is “no” in the population.

Selfing the founder individuals for `nSelf` generations

```
nSelf<-6
Inbred<-founders
for (i in 1:nSelf){
  Inbred<-self(Inbred, nProgeny=2)
   #print(Inbred@id)
}
```

How to phenotype these individuals.

```
### Phenotype 640 individuals of them
varE<-1
PhenoInbred<-setPheno(Inbred,varE=varE,simParam=SP)  # h2=0.5, Becasue varG is default
```

- Practice: Can you select the Top10 best Inbreds based on phenotypic data?

- We can now try randomly cross n individuals selected from the inbreds (*** Note: no sexes indicated) using `makeCross` . Each being crossed once.

```
nSel<-10
SampleInbred<-sample(Inbred@id,nSel)   # Randomly select nSel individuals
CrossingPlan<-matrix(SampleInbred,nrow=nSel/2,ncol=2)   # A matrix with column for Fema
F1<-makeCross(Inbred,crossPlan=CrossingPlan,simParam=SP) # This is the F1 generation cr
F2pop<-self(F1,nProgeny=100)  # Generated 100 F2s
```

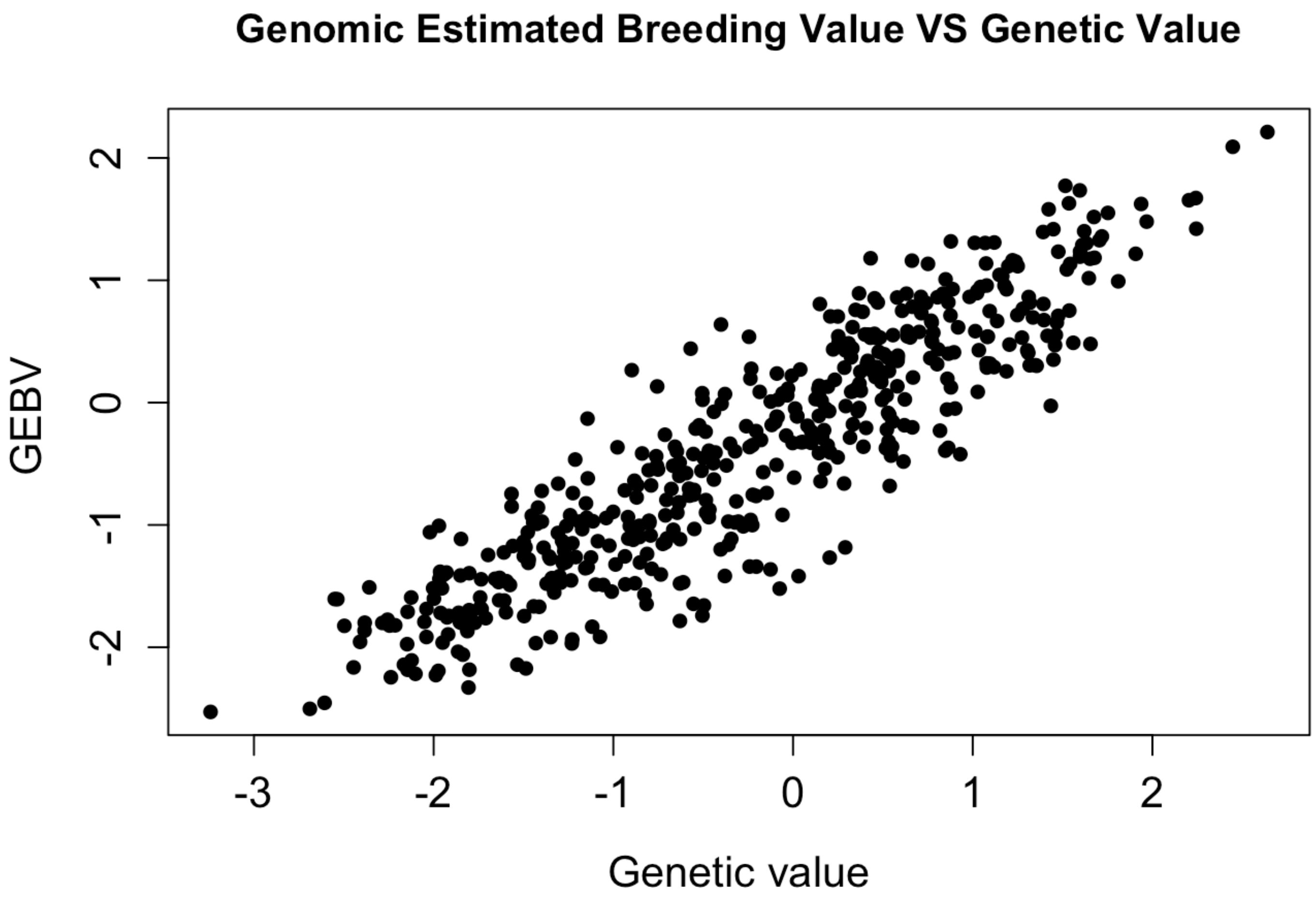
How to build a Genomic Selection model

- Note: Inbred pop was phenotyped above and can be genotyped. Its marker data by default is the "1st" SNP chip previously designed as part of the simulation parameters SP. If you have different SNP chips, you have to specify which SNp chip to use using intergers for the `snpChip` . So now we can train a genomic selection model using these Inbreds' phenotypic and genotypic data. We will then apply this GS model on F2s.
- Note: You can use `pullQtlGeno()` function to pull out genotypic data of your population(s).

```
GSmodel<-RRBLUP(PhenoInbred,snpChip=1,simParam=SP)  ## Set up GS model
GEBVF2<-setEBV(F2pop,GSmodel,simParam=SP) ## Estimate the GEBVs of F2s
```

How to compare genetic values (True BV) and breeding values

```
plot(gv(GEBVF2), ebv(GEBVF2), pch=16, xlab="Genetic value", ylab=" GEBV", main="Genomic
```



- We can estimate the GS accuracy by correlating the GEBV to true BV

```
cor(gv(GEBVF2),ebv(GEBVF2))
```

```
##         [,1]
## [1,] 0.90609
```

> - Practice: Can you try to set up scripts to compare GS accuracy and Phenotypic selection accuray on the inbreds? For example, we set phenotypic data for these GEBVF2s. We then select best 5% individuals based on phenotypic data vs based on their GEBVs, we then phenotype these individuals again for another year, and compare their selection accuracies.

Set up a scheme to compare phenotypic accuracy and genotypic accuracy

```
PhenoF2<-setPheno(GEBVF2,varE=1,simParam=SP)
nSelect<-0.1*nInd(GEBVF2)  # select 5% individuals

PSTop10<-PhenoF2[order(-PhenoF2@pheno)][1:nSelect]
GSTop10<-PhenoF2[order(-PhenoF2@ebv)][1:nSelect]

PSTop10_yr2<-setPheno(PSTop10,varE=1,simParam=SP)
GSTop10_yr2<-setPheno(GSTop10,varE=1,simParam=SP)

cor(PSTop10_yr2@pheno,PSTop10@pheno)
```

```
##              [,1]
## [1,] -0.02973494
```

```
cor(GSTop10_yr2@pheno,GSTop10@ebv)
```

```
##           [,1]
## [1,] 0.2228255
```

How to estimate selection intensity

Selection intensity is estimated as the difference of `mean(selected population)-mean(reference population)`

```
## Pull out selectInd, estimate the selection intensity

Refpop<-data.frame(PhenoF2@id,PhenoF2@pheno,PhenoF2@mother,PhenoF2@father)
colnames(Refpop)[1:2]<-c("id","trait")
Refpop$mean<-Refpop$trait/sd(Refpop$trait)

selectpop<-GSTop10@id

selectPopmean<-mean(Refpop[Refpop$id%in%selectpop,]$mean)
selectInt<-selectPopmean-mean(Refpop$mean)
print(selectInt)
```

```
## [1] 1.096349
```

- Practice, Change the number of individuals being selected from 5% to 10% and report the selection intensity