

# Exemple de simulation pour explorer la prédition génomique

Timothée Flutre (INRAE), Charlotte Brault

16/02/2021

## Abstract

Ce document a pour but de montrer un exemple de prédition génomique à partir de données simulées.

## Contents

<b>1 Contexte</b>	<b>2</b>
<b>2 Chargement des fonctions</b>	<b>2</b>
<b>3 Modèle</b>	<b>2</b>
<b>4 Simulation des données</b>	<b>3</b>
4.1 Simulation des génotypes . . . . .	3
4.1.1 Génotypes aux SNP . . . . .	3
4.1.2 Echantillonnage des SNP . . . . .	5
4.1.3 Visualisation de la structure . . . . .	5
4.2 Simulation des phénotypes . . . . .	10
4.2.1 Effets additifs des SNP . . . . .	10
4.2.2 Valeurs génotypiques additives et variance génétique additive . . . . .	10
4.2.3 Erreurs . . . . .	11
4.2.4 Phénotypes . . . . .	11
<b>5 Description des données simulées</b>	<b>12</b>
5.1 Fréquences alléliques . . . . .	12
5.2 Relations génétiques additives . . . . .	14
5.3 Déséquilibre de liaison . . . . .	16
<b>6 Distinction des sous-population</b>	<b>20</b>
<b>7 Evaluation de la précision de prédiction</b>	<b>21</b>
7.1 Définition des ensembles d'entraînement et de test . . . . .	21
7.1.1 Entraînement . . . . .	21
7.1.2 Test . . . . .	24
7.2 Validation croisée . . . . .	25
7.2.1 Fonctions . . . . .	25
7.2.2 Partitions . . . . .	25
7.2.3 Validation . . . . .	26
<b>8 Annexe</b>	<b>26</b>

# 1 Contexte

Ce document fait partie de l'atelier “Prédiction Génomique” organisé et animé par Jacques David, Margaux Jullien, Vincent Ségura, Charlotte Brault, Michel Colombo et Friedrich Longin, à [Montpellier SupAgro](#) dans le cadre de l'option [APIMET](#) (Amélioration des Plantes et Ingénierie végétale Méditerranéennes et Tropicales) couplée à la spécialité [SEPMET](#) (Semences Et Plants Méditerranéens Et Tropicaux) du Master [3A](#) (Agronomie et Agroalimentaire), et de la spécialisation [PIST](#) du [Cursus Ingénieur d'Agroparistech](#).

Le copyright appartient à Montpellier SupAgro et à l’Institut National de la Recherche Agronomique. Le contenu du répertoire est sous license [Creative Commons Attribution-ShareAlike 4.0 International](#). Veuillez en prendre connaissance et vous y conformer (contactez les auteurs en cas de doute).

Les versions du contenu sont gérées avec le logiciel git, et le dépôt central est hébergé sur [GitHub](#).

Il est recommandé d'avoir déjà lu attentivement les documents “Premiers pas” et “Prédiction génomique” de l'atelier.

De plus, ce document nécessite de charger des paquets additionnels (ceux-ci doivent être installés au préalable sur votre machine, via `install.packages("pkg")`):

```
suppressPackageStartupMessages(library(rrBLUP))
suppressPackageStartupMessages(library(cvTools))
```

Un certain niveau de déséquilibre de liaison entre génotypes aux SNP est indispensable pour obtenir une précision de prédiction suffisamment élevée en validation croisée. Pour cela, on peut utiliser le processus du coalescent avec recombinaison. Une bonne approximation de celui-ci est implémenté dans le paquet [scrm](#). Par ailleurs, afin de tracer le déséquilibre de liaison en fonction de la distance physique, il vous faut aussi le paquet [GenomicRanges](#) de Bioconductor. Afin de faciliter l'utilisation de ces paquets dans ce document, il vous faut aussi avoir mon paquet de travail, [rutilstimflutre](#), disponible sur GitHub.

```
suppressPackageStartupMessages(library(scrm))
# if (!requireNamespace("BiocManager", quietly = TRUE))
#   install.packages("BiocManager")
#
# BiocManager::install("GenomicRanges")
suppressPackageStartupMessages(library(GenomicRanges))
#suppressPackageStartupMessages(library(rutilstimflutre))
suppressPackageStartupMessages(library(adegenet))
```

Il est également utile de savoir combien de temps est nécessaire pour exécuter tout le code R de ce document (voir l'annexe):

```
t0 <- proc.time()
```

# 2 Chargement des fonctions

Sourcer le fichier `rutils_like.R`. Option 1 : aller dans l'onglet `rutils_like.R` dans Rstudio et cliquer sur le bouton “Source”. Option 2 : appliquer la fonction “source” en mettant le chemin d'accès vers ce script R (chez moi c'est “`~/work/debug/rutils_like.R`”).

Vérifier que les fonctions sont bien chargées dans l'environnement.

```
source("~/work/debug/rutils_like.R")
```

# 3 Modèle

En se limitant à une architecture additive infinitésimale:

$$\begin{aligned}\mathbf{y} &= \mathbf{1} \mu + X \boldsymbol{\beta} + \boldsymbol{\epsilon} \\ &= \mathbf{1} \mu + \mathbf{a} + \boldsymbol{\epsilon}\end{aligned}$$

avec:

- $\boldsymbol{\epsilon} \sim \mathcal{N}_N(\mathbf{0}, \sigma^2 \text{Id})$ ;
- $\boldsymbol{\beta} \sim \mathcal{N}_P(\mathbf{0}, \sigma_\beta^2 \text{Id})$ ;
- $\mathbf{a} \sim \mathcal{N}_N(\mathbf{0}, \sigma_a^2 A_{\text{mark}})$  avec  $A_{\text{mark}} = \frac{XX^T}{2 \sum_p f_p(1-f_p)}$ .

Cet estimateur de  $A_{\text{mark}}$  est décrit dans [Habier et coll. \(2007\)](#), mais un meilleur estimateur, centré, est proposé dans [VanRaden \(2008\)](#): pour plus de détails, lire [Toro et coll. \(2011\)](#) et [Vitezica et coll. \(2013\)](#).

## 4 Simulation des données

```
set.seed(111)
```

### 4.1 Simulation des génotypes

Pour simuler une structure, on peut utiliser une fonction de Timothée Flutre (`rutilstimflutre`) : `SimulCoalescent`.

Voici un exemple de code pour appliquer cette fonction, disponible sur [Github](#)

#### 4.1.1 Génotypes aux SNP

Paramètres du Coalescent

```
nb.genos <- 500 # nombre de génotypes
nb.chroms <- 5
Ne <- 10^4 # taille efficace
chrom.len <- 5*10^5 # longueur de chaque chromosome
mu <- 10^{(-8)} # taux de mutation
c.rec <- 10^{(-8)} # taux de recombinaison
?simulCoalescent # pour avoir l'aide de la fonction

## No documentation for 'simulCoalescent' in specified packages and libraries:
## you could try '??simulCoalescent'
```

```
genomes <- simulCoalescent(nb.inds=nb.genos, nb.reps=nb.chroms,
                           pop.mut.rate=4 * Ne * mu * chrom.len,
                           pop.recomb.rate=4 * Ne * c.rec * chrom.len,
                           chrom.len=chrom.len,
                           #mig.rate=5, # => valeur par défaut sans structure
                           nb.pops=1,
                           verbose=1)
```

#### 4.1.1.1 Coalescent séquentiel avec recombinaison sans structure

```
## simulate according to the SCRM ...
## scrm 1000 5 -t 200 -r 200 5e+05 -SC abs -oSFS
## nb of SNPs: 7210
```

```

## chr1 chr2 chr3 chr4 chr5
## 1404 1366 1474 1546 1420
## make a data.frame with SNP coordinates ...
## randomize haplotypes to make diploid genotypes ...
## convert haplotypes into genotypes encoded as allele dose ...
## permute alleles in haplotypes ...
## permute alleles in genotypes ...

dim(genomes$genos)

## [1] 500 7210

genomes$genos[1:8,1:8]

##      snp0001 snp0002 snp0003 snp0004 snp0005 snp0006 snp0007 snp0008
## ind001      2       2       2       2       2       1       2       2
## ind002      2       2       2       2       2       1       2       2
## ind003      2       2       2       2       2       1       2       2
## ind004      2       2       2       2       2       1       2       2
## ind005      2       2       2       2       2       0       2       2
## ind006      2       2       2       2       2       0       2       2
## ind007      2       2       2       2       2       0       2       2
## ind008      2       2       2       2       2       0       2       2

```

**4.1.1.2 Coalescent séquentiel avec recombinaison et avec structure** La structure est ici simulée avec des taux de migration : plus le taux de migration est faible, plus les populations seront différencierées entre elles.

```
genomes.struct$genos[1:8,1:8]
```

```
##      snp00001 snp00002 snp00003 snp00004 snp00005 snp00006 snp00007 snp00008
## ind001      2      2      2      0      2      0      0      2
## ind002      2      2      2      0      2      0      0      2
## ind003      2      2      2      0      2      0      0      2
## ind004      2      2      2      0      2      0      0      2
## ind005      2      2      2      0      2      0      0      2
## ind006      2      2      2      0      2      0      0      2
## ind007      2      2      2      0      2      0      0      2
## ind008      2      2      2      0      2      0      0      2
```

#### 4.1.2 Echantillonnage des SNP

On sous-échantillonne des SNP parmi les bases du génome simulées

```
dim(genomes$genos)
```

```
## [1] 500 7210
```

```
dim(genomes.struct$genos)
```

```
## [1] 300 12253
```

```
P <- 5000 # nombre de SNP à échantillonner
# on s'assure d'abord qu'il y a assez de marqueurs
stopifnot(ncol(genomes.struct$genos) >= P,
          ncol(genomes$genos) >= P)
# sans structure
idx.snps.tokeep <- sample.int(n=ncol(genomes$genos), size=P, replace=FALSE)
X <- genomes$genos[, idx.snps.tokeep]
# avec structure
idx.snps.tokeep <- sample.int(n=ncol(genomes.struct$genos), size=P, replace=FALSE)
X.struct <- genomes.struct$genos[, idx.snps.tokeep]
```

```
dim(X)
```

```
## [1] 500 5000
```

```
X[1:8, 1:8]
```

```
##      snp3935 snp1954 snp6060 snp1584 snp0506 snp3072 snp5288 snp5475
## ind001      2      2      2      0      1      0      2      0
## ind002      2      2      1      0      2      2      2      0
## ind003      2      1      2      0      2      2      1      0
## ind004      1      0      2      0      2      1      2      0
## ind005      2      0      2      0      1      1      2      0
## ind006      2      1      2      0      2      0      2      0
## ind007      2      2      1      0      2      0      1      0
## ind008      2      1      2      0      2      1      2      0
```

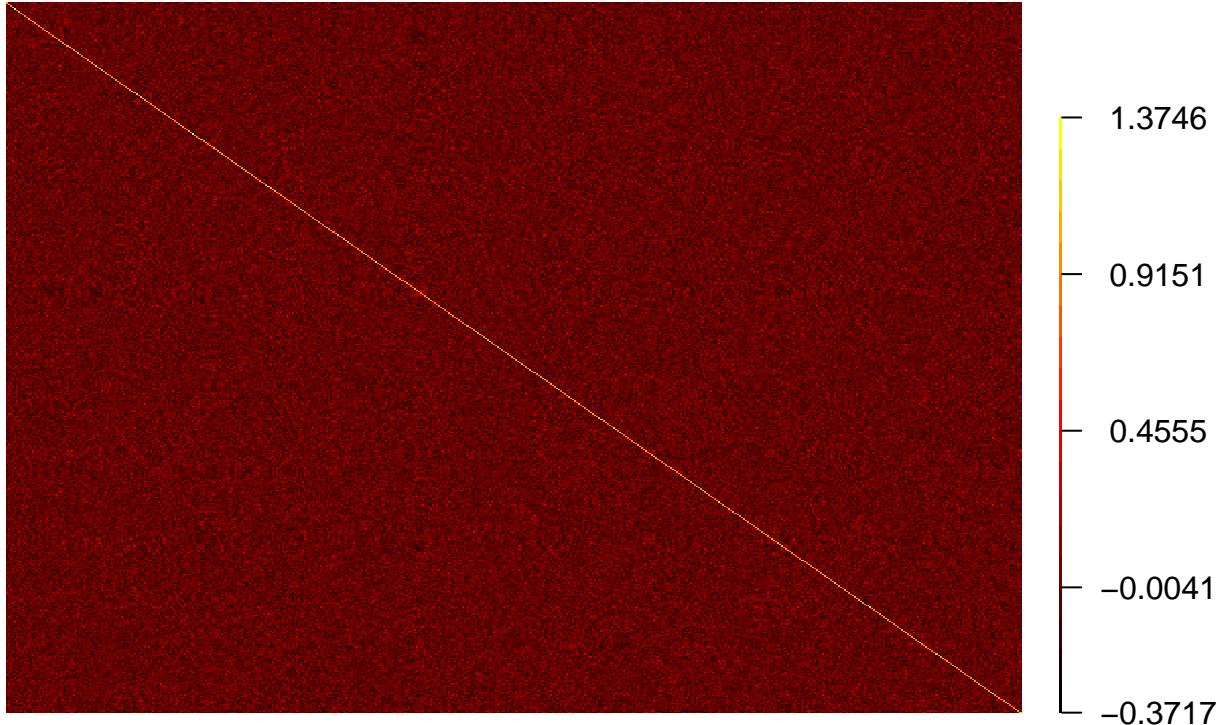
```
snp.coords <- genomes$snp.coords[colnames(X),]
#rm(genomes)
```

#### 4.1.3 Visualisation de la structure

Représentation graphique de la structure

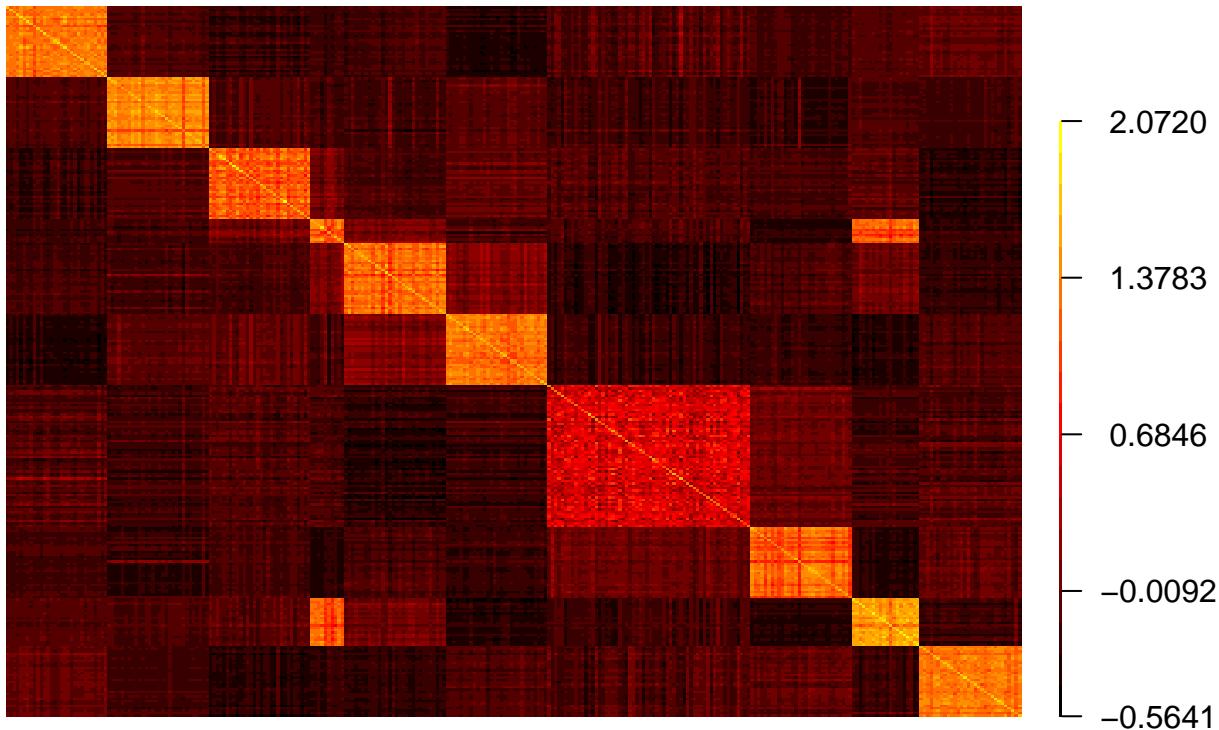
```
A.nostruct <- estimGenRel(X=X, verbose=0)
imageWithScale(A.nostruct, main="Additive genetic relationships with no structure")
```

## Additive genetic relationships with no structure



```
rm(A.nostruct)
A.struct <- estimGenRel(X=X.struct, verbose=0)
imageWithScale(A.struct, main="Additive genetic relationships with structure")
```

## Additive genetic relationships with structure



```
rm(A.struct)
```

Affichage des groupes de structure génétique avec une **Analyse en Composantes Principales**.

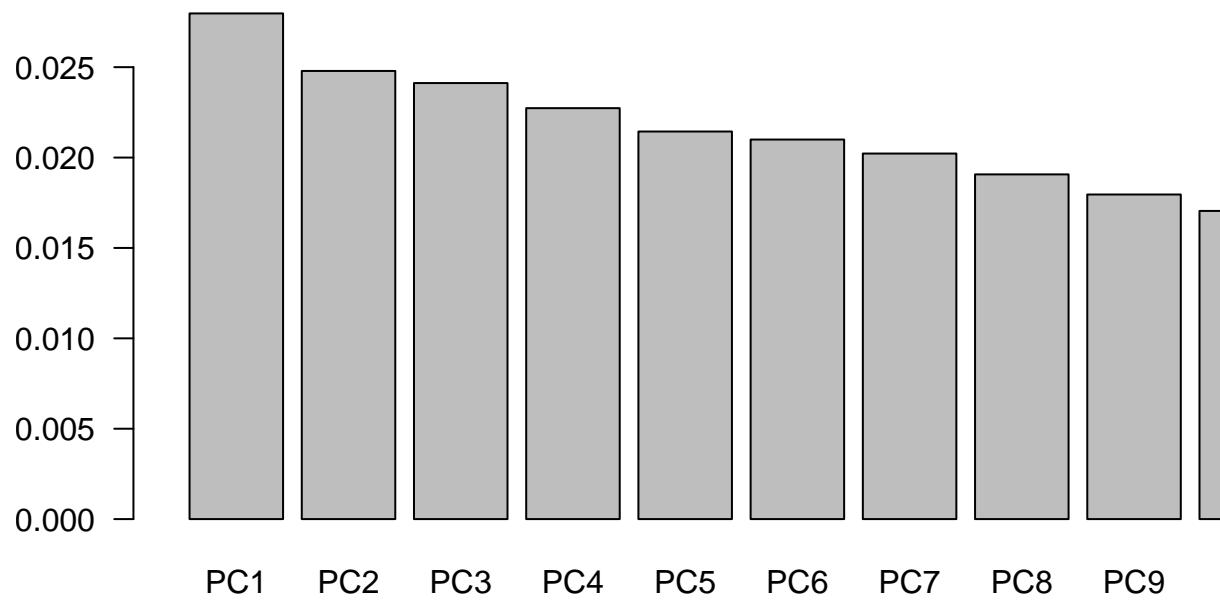
```
# Sans structure
```

```
out.pca.nostruct <- pca(X=X)
out.pca.nostruct$prop.vars[1:4]
```

```
##    PC1     PC2     PC3     PC4
## 0.0280 0.0248 0.0241 0.0227
```

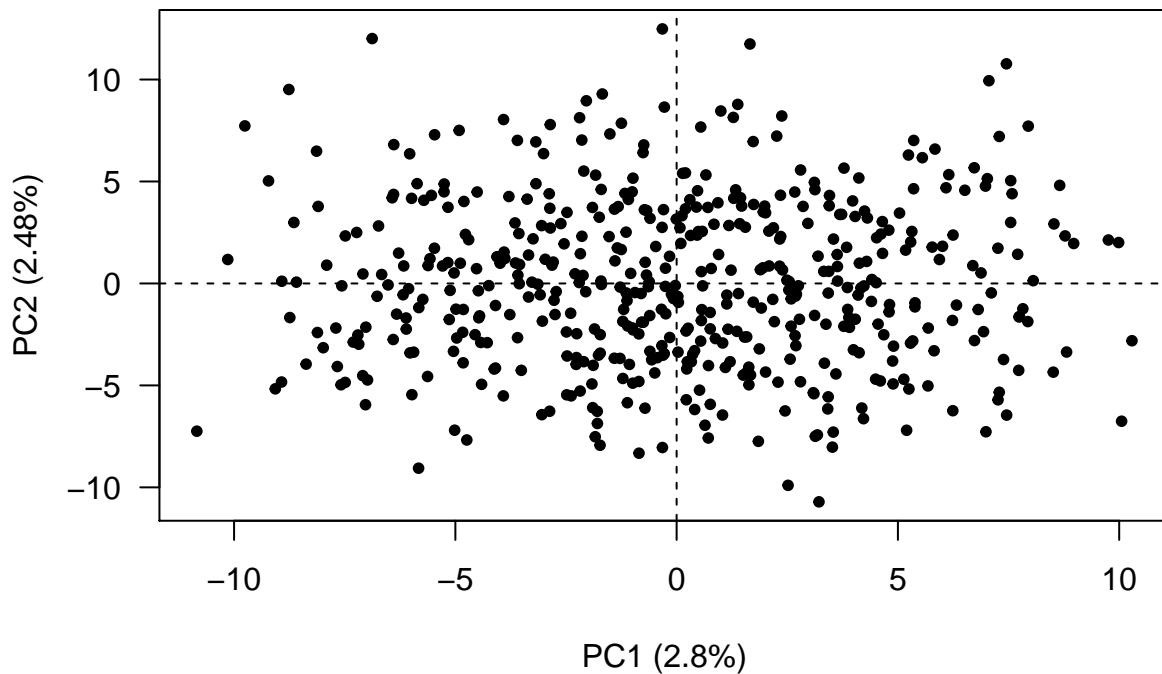
```
barplot(out.pca.nostruct$prop.vars,
        main="Proportion of variance explained by each PC, without structure",
        xlim=c(0,10), las=1)
```

## Proportion of variance explained by each PC, without structure



```
plotPca(rotation=out.pca.nostruct$rot.dat,  
        prop.vars=out.pca.nostruct$prop.vars,  
        main="PCA without structure")
```

PCA without structure



```
# Avec structure  
out.pca.struct <- pca(X=X.structure)  
out.pca.structure$prop.vars[1:4]
```

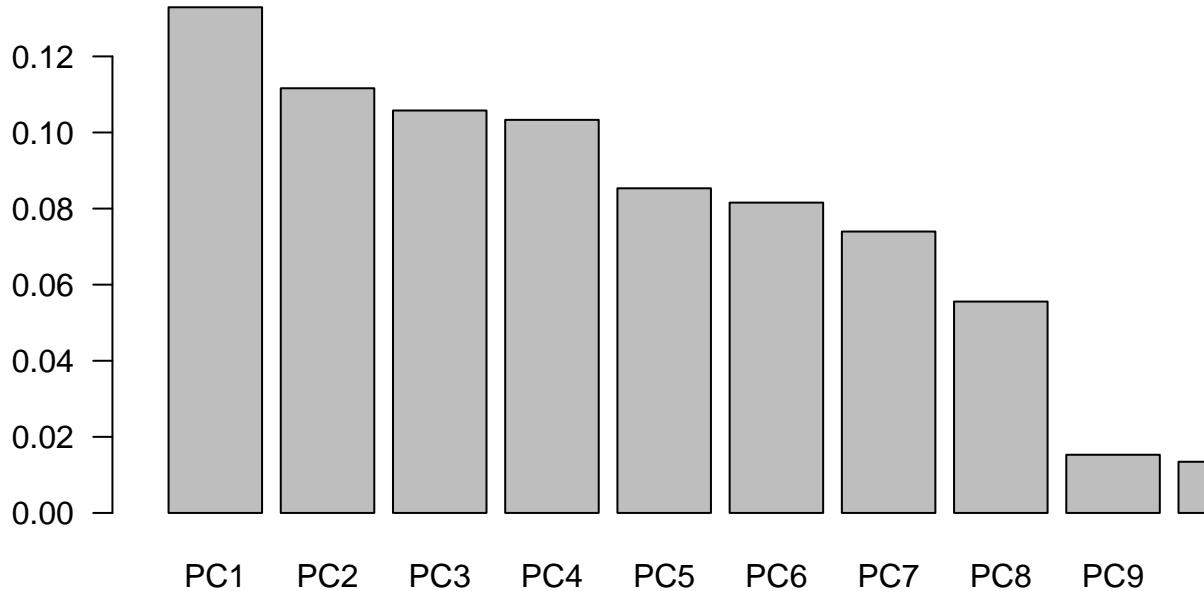
```

##   PC1   PC2   PC3   PC4
## 0.133 0.112 0.106 0.103

barplot(out.pca.struct$prop.vars,
        main="Proportion of variance explained by each PC, with structure",
        xlim=c(0,10), las=1)

```

**Proportion of variance explained by each PC, with structure**

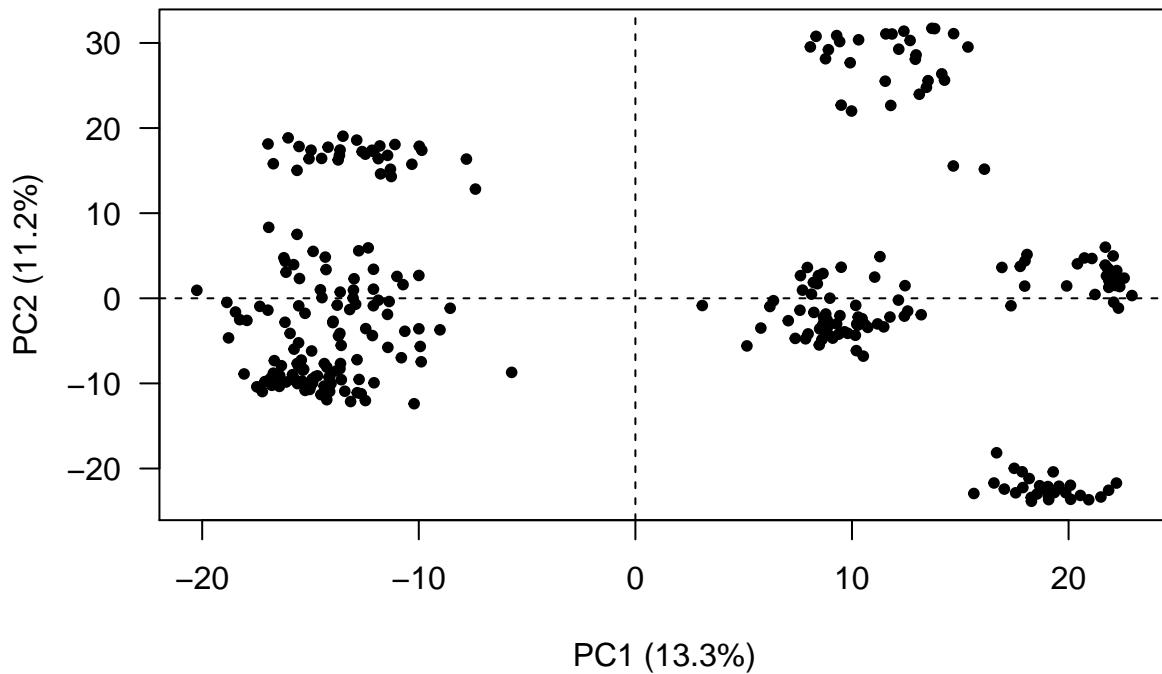


```

plotPca(rotation=out.pca.struct$rot.dat,
        prop.vars=out.pca.struct$prop.vars,
        main="PCA without structure")

```

## PCA without structure



## 4.2 Simulation des phénotypes

### 4.2.1 Effets additifs des SNP

On peut simuler 2 types d'architecture génétique :

- Modèle infinitésimal : chaque marqueur a un effet faible sur le phénotype.
- Modèle polygénique : certains marqueurs à effet nuls et certains à effet non nul (QTL).

```
is.infinitesimal <- TRUE
if(is.infinitesimal){
  # Modèle infinitésimal : chaque marqueur a un effet faible sur le phénotype
  sigma.beta2 <- 10^(-3) # chosen arbitrarily
  beta <- rnorm(n=P, mean=0, sd=sqrt(sigma.beta2)) # effet du marqueur, tiré dans une loi normale
} else {
  # Simulation avec certains marqueurs à effet nuls et certains à effet non nul (QTL).
  nb_QTL <- 10
  QTL <- sample(P, nb_QTL) # échantillonnage des marqueurs à effet non nul (QTL)
  beta <- rep(0, P) # effet associé au marqueur égal à 0
  beta[QTL] <- rnorm(n=nb_QTL, mean=0, sd=sqrt(sigma.beta2)) # sauf pour les QTL
}
# variance et moyenne comparables
beta <- scale(beta, center=TRUE, scale=TRUE)
```

On obtient un vecteur  $\beta$  des vrais effets des marqueurs.

### 4.2.2 Valeurs génotypiques additives et variance génétique additive

Ici on a gardé la matrice des génotypes aux marqueurs issue de la simulation **sans structure**. Notez que  $X$  est initialement codé en  $\{0, 1, 2\}$ , mais que dans la suite on peut le centrer à l'aide des fréquences alléliques comme dans l'estimateur de VanRaden:

```

N <- nb.genos # nombre d'individus
afs <- colMeans(X) / 2
tmp <- matrix(rep(1, N)) %*% (2 * afs)
X.center <- X - tmp
# Effet génétique additif
a <- X.center %*% beta
# Variance génétique additive
(sigma.a2 <- 1 * 2 * sum(afs * (1 - afs)))

```

```
## [1] 679
```

#### 4.2.3 Erreurs

```

h2 <- 0.7 # héritabilité au sens strict
(sigma2 <- ((1 - h2) / h2) * sigma.a2) # variance d'erreur calculée à partir de  $h^2$  et de sigma.a2

## [1] 291
epsilon <- rnorm(n=N, mean=0, sd=sqrt(sigma2)) # vecteur d'erreurs

```

#### 4.2.4 Phénotypes

```

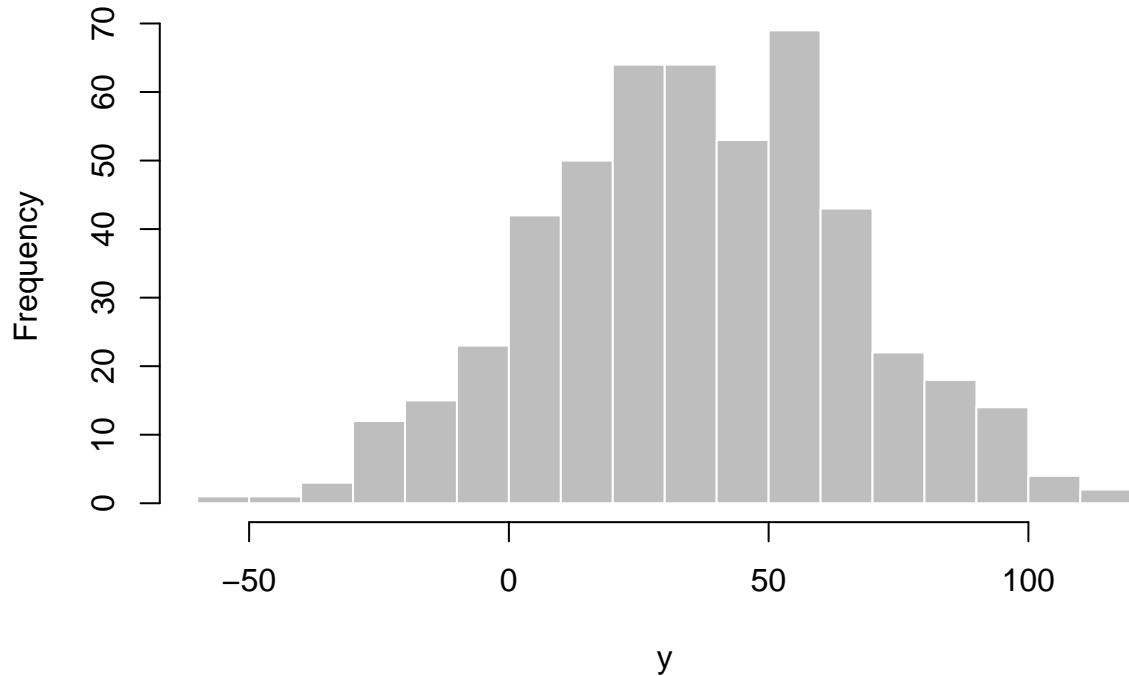
mu <- 36 # chosen arbitrarily
y <- mu + X.center %*% beta + epsilon
summary(y)

##          V1
##  Min.   :-50.1
##  1st Qu.: 15.8
##  Median : 35.2
##  Mean   : 36.1
##  3rd Qu.: 56.3
##  Max.   :114.5

hist(y, breaks="FD", main="Phenotypes", col="grey", border="white")

```

## Phenotypes

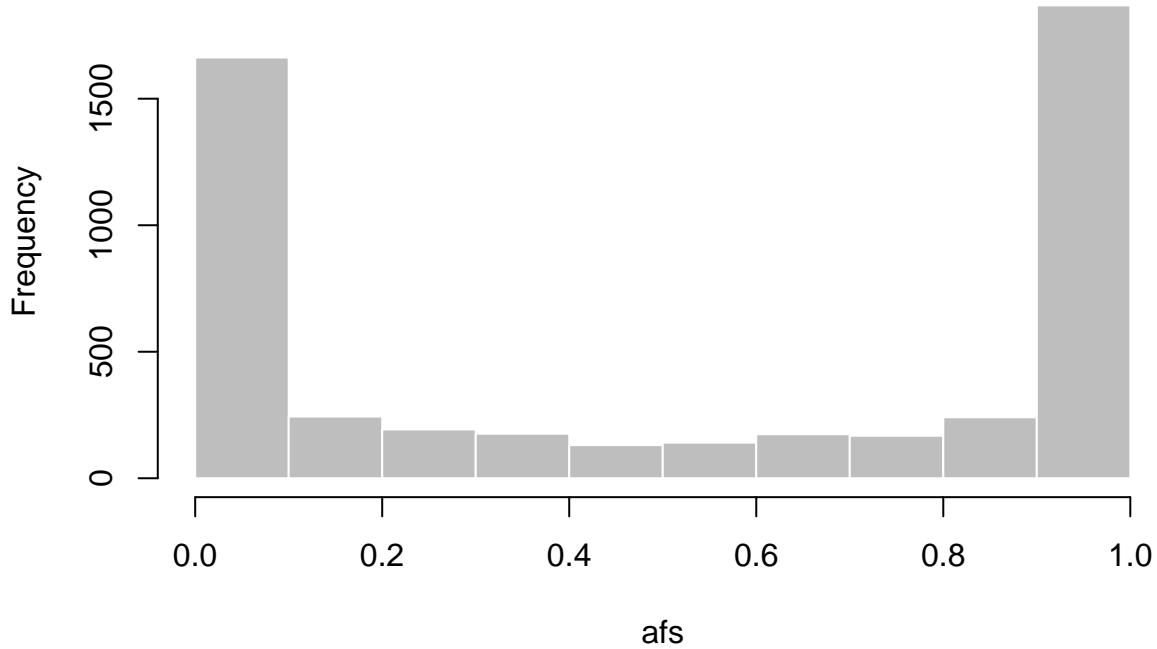


## 5 Description des données simulées

### 5.1 Fréquences alléliques

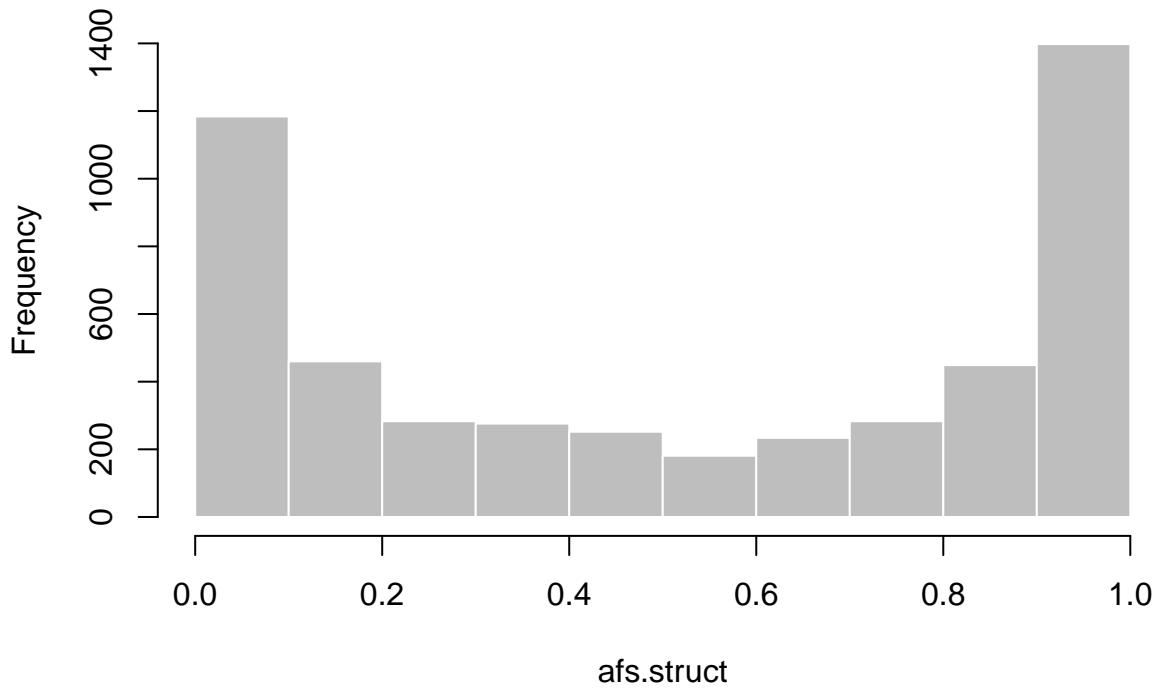
```
afs <- colMeans(X) / 2  
hist(afs, xlim=c(0, 1), main="Allele frequencies without structure", col="grey", border="white")
```

### Allele frequencies without structure



```
afs.struct <- colMeans(X.structure) / 2  
hist(afs.struct, xlim=c(0, 1), main="Allele frequencies with structure", col="grey", border="white")
```

### Allele frequencies with structure

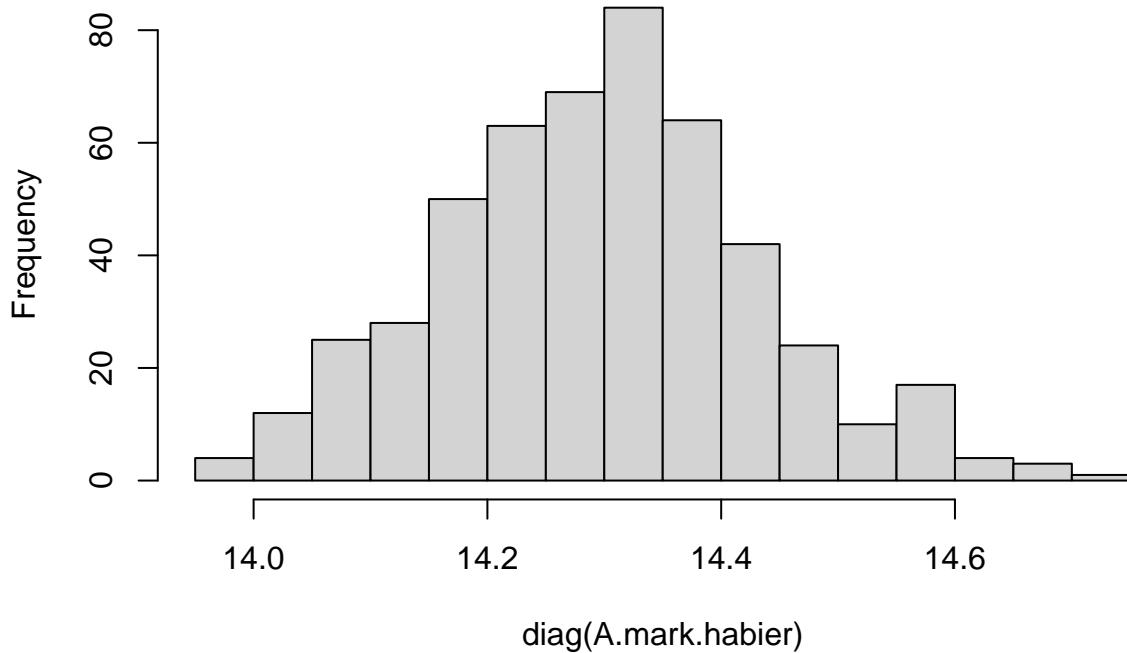


## 5.2 Relations génétiques additives

Estimateur de Habier et coll. (2007):

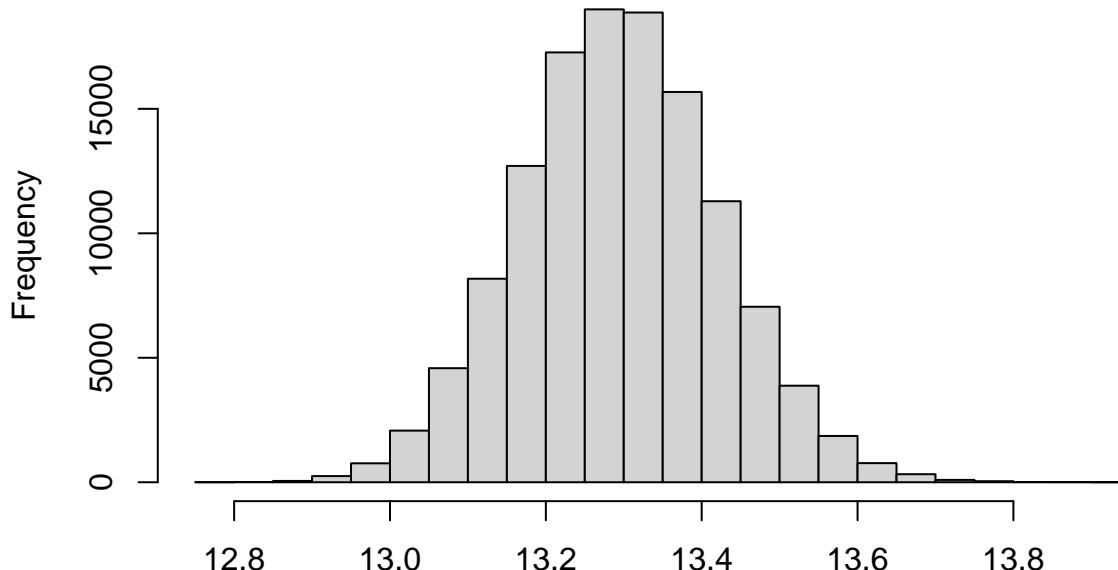
```
A.mark.habier <- (X %*% t(X)) / (2 * sum(afs * (1 - afs)))
hist(diag(A.mark.habier), breaks="FD")
```

**Histogram of diag(A.mark.habier)**



```
hist(A.mark.habier[upper.tri(A.mark.habier)])
```

## Histogram of A.mark.habier[upper.tri(A.mark.habier)]

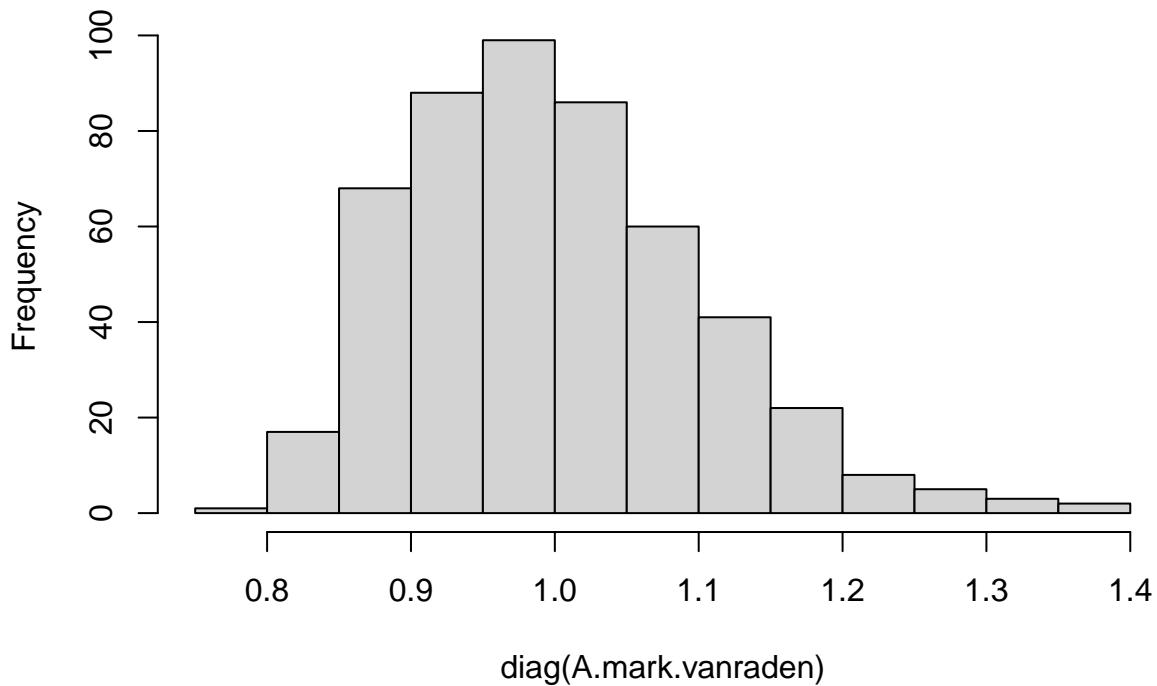


A.mark.habier[upper.tri(A.mark.habier)]

Estimateur de VanRaden (2008):

```
A.mark.vanraden <- (X.center %*% t(X.center)) / (2 * sum(afs * (1 - afs)))
hist(diag(A.mark.vanraden), breaks="FD")
```

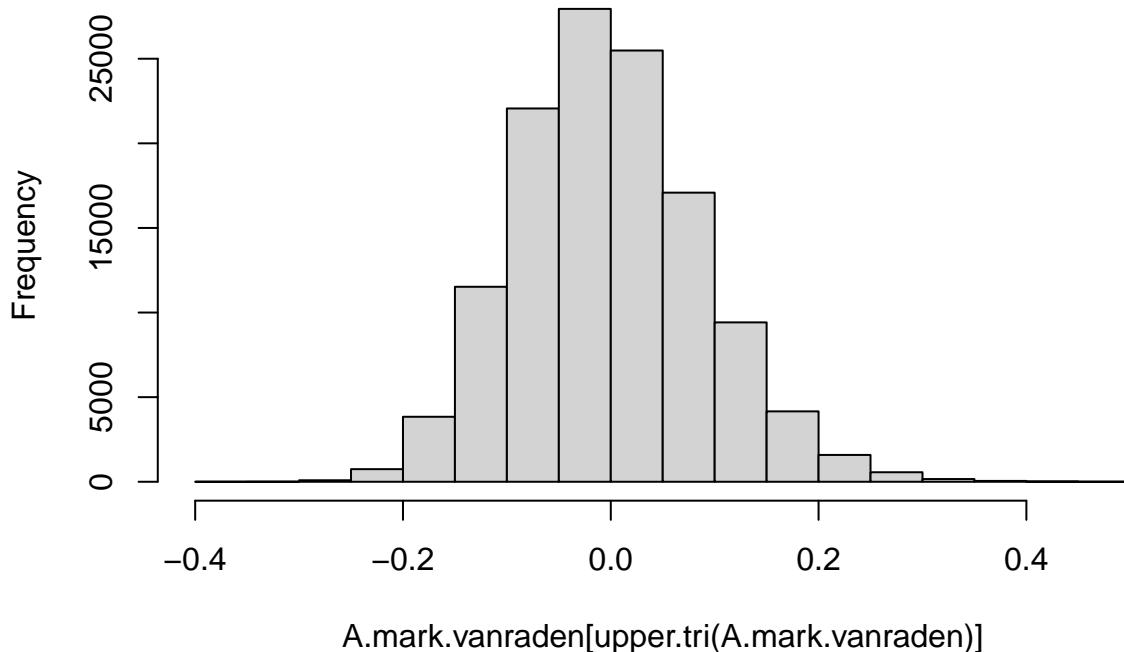
## Histogram of diag(A.mark.vanraden)



diag(A.mark.vanraden)

```
hist(A.mark.vanraden[upper.tri(A.mark.vanraden)])
```

Histogram of A.mark.vanraden[upper.tri(A.mark.vanraden)]



### 5.3 Déséquilibre de liaison

```
for(chr in unique(snp.coords$chr)){
  LD <- estimLd(X=X, snp.coords = snp.coords, only.chr =chr)
  LD$dist <- distSnpPairs(data.frame(loc1=LD$loc1, loc2=LD$loc2),
                           genomes$snp.coords,
                           nb.cores = 1, verbose = 1)

  plotLd(x=LD$dist, y=LD$cor2, xlab="Physical distance (in bp)",
         main=paste("Linkage disequilibrium for ", chr),
         add.ohta.kimura=TRUE, Ne=Ne, c=c.rec)
}

## extract relevant genotypes and SNPs...
## estimate pairwise LD...

## make GRanges ...

## calculate pairwise distances ...

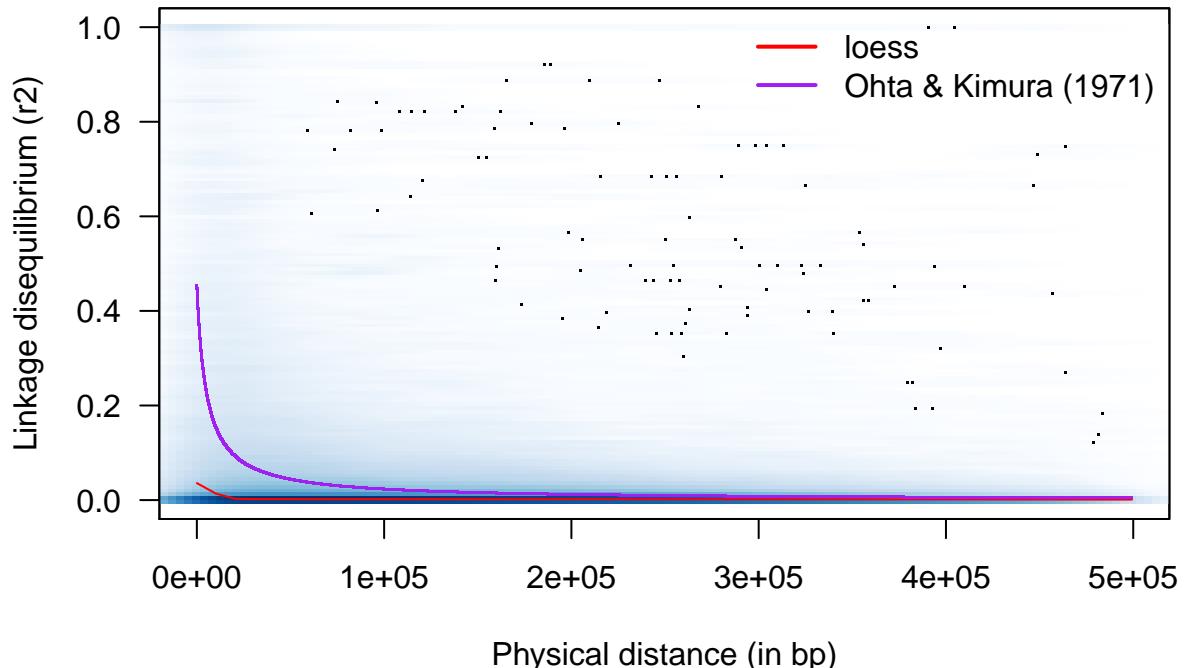
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :
## Binning grid too coarse for current (small) bandwidth: consider increasing
## 'gridsize'

## extract relevant genotypes and SNPs...
## estimate pairwise LD...

## make GRanges ...
## calculate pairwise distances ...
```

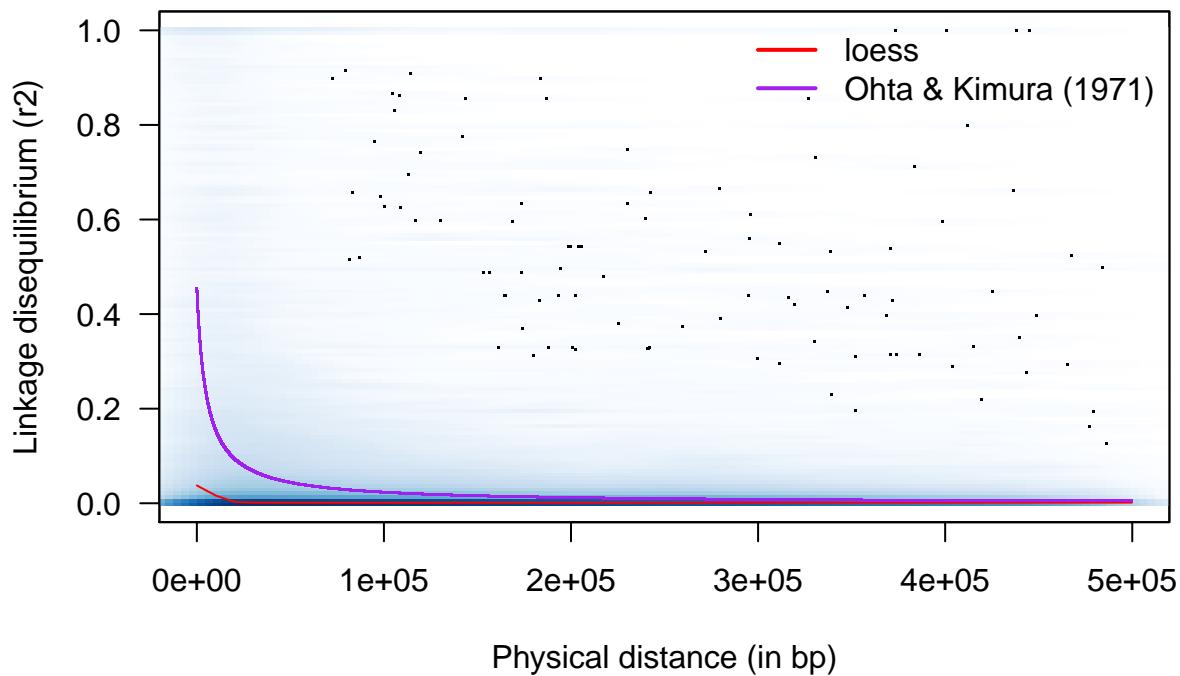
```
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'
```

### Linkage disequilibrium for chr3



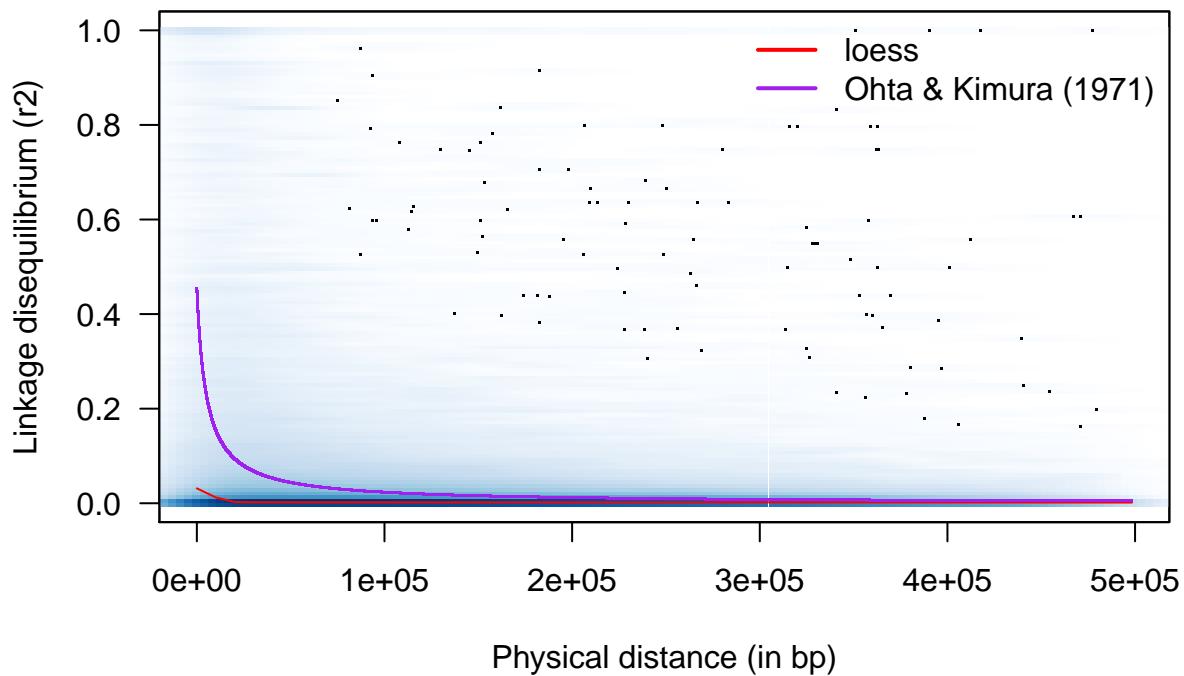
```
## extract relevant genotypes and SNPs...  
## estimate pairwise LD...  
  
## make GRanges ...  
## calculate pairwise distances ...  
  
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'
```

## Linkage disequilibrium for chr2



```
## extract relevant genotypes and SNPs...
## estimate pairwise LD...
## make GRanges ...
## calculate pairwise distances ...
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :
## Binning grid too coarse for current (small) bandwidth: consider increasing
## 'gridsize'
```

## Linkage disequilibrium for chr5

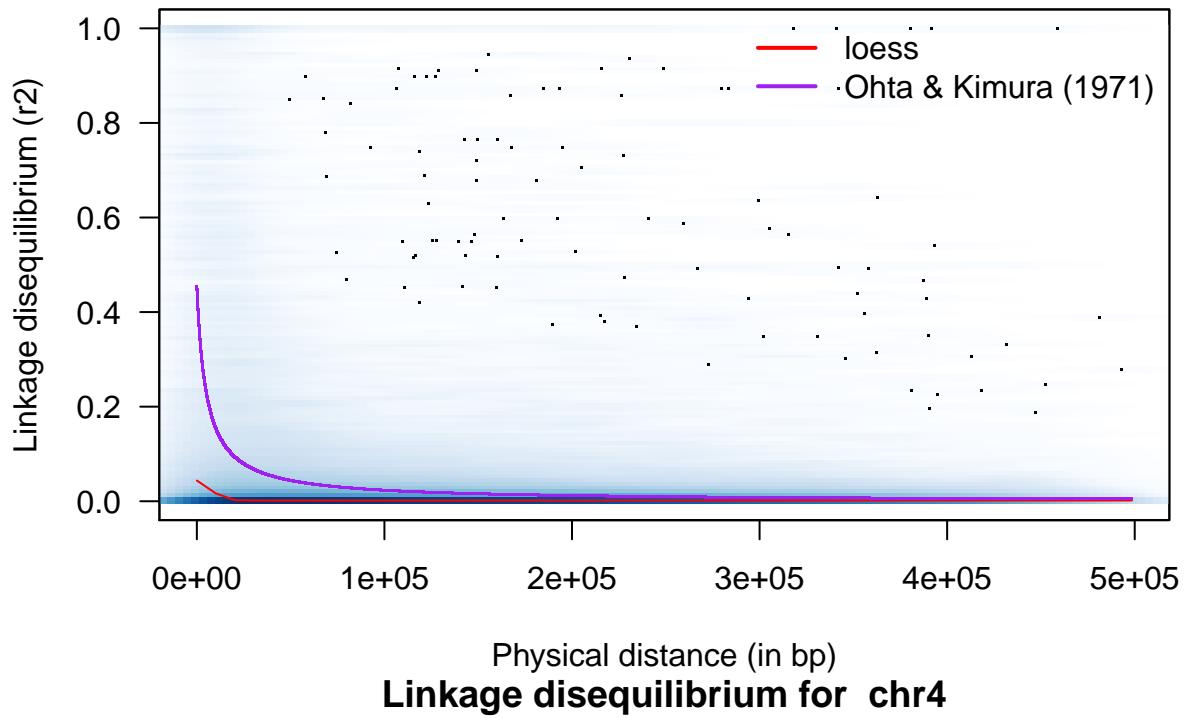


```
## extract relevant genotypes and SNPs...
## estimate pairwise LD...

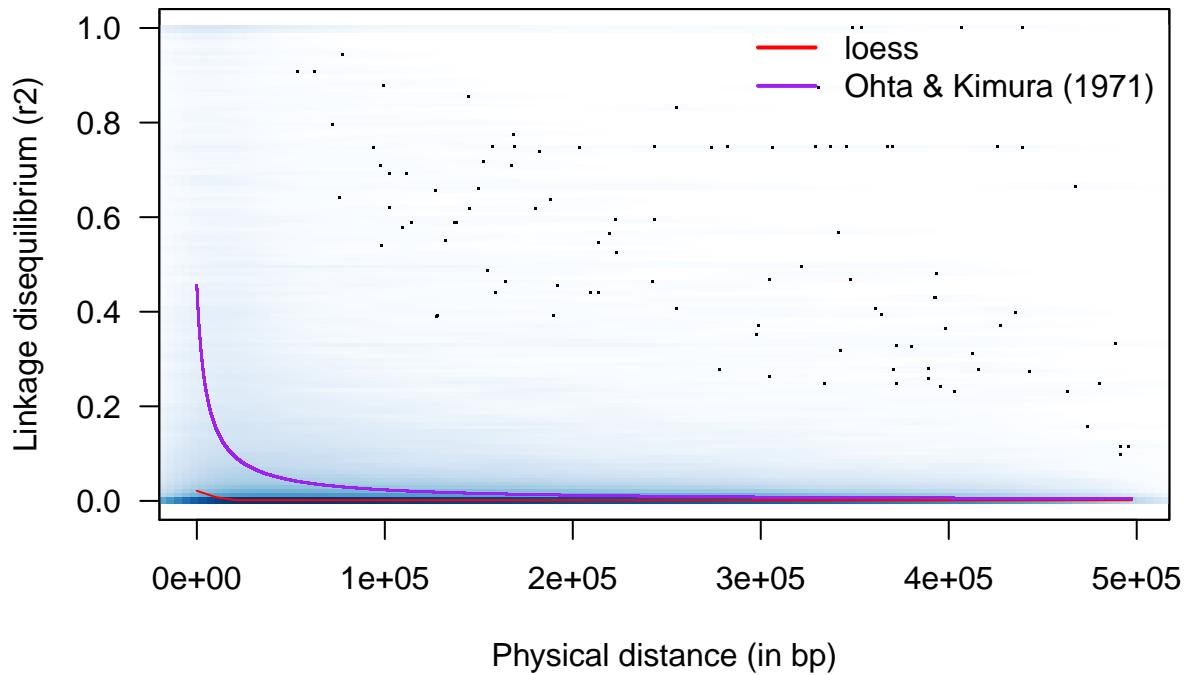
## make GRanges ...
## calculate pairwise distances ...

## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :
## Binning grid too coarse for current (small) bandwidth: consider increasing
## 'gridsize'
```

### Linkage disequilibrium for chr1



### Linkage disequilibrium for chr4



## 6 Distinction des sous-population

Une des problématiques en prédiction génomique est la distance génétique entre la population d'entraînement, qui a servi à ajuster le modèle (c'est-à-dire à estimer les effets associés aux marqueurs  $\hat{\beta}$ ). Si la population

est structurée, alors il y a de la distance génétique entre les différentes sous-populations. Vous pourrez voir comment évolue la précision de prédiction lorsque la population de validation fait partie d'une sous-population différente de la population d'entraînement.

```
sub_pop <- find.clusters(x=X$struct, n.pca=100, scale=TRUE, method="kmeans",
                           choose.n.clust=TRUE, n.clust=6)
levels(sub_pop$grp)

## [1] "1" "2" "3" "4" "5" "6"
sub_pop$size

## [1] 30 30 30 60 30 120
```

## 7 Evaluation de la précision de prédiction

Pour évaluer la qualité de notre modèle et l'influence des paramètres de simulation, nous allons utiliser la validation croisée. L'ensemble des génotypes ( $X$ ) et des phénotypes ( $y$ ) va être divisé en ensembles d'entraînement et de test:  $X_{train}, y_{train}$  et  $X_{test}, y_{test}$ .

Le modèle de prédiction va être ajusté sur l'ensemble d'entraînement pour estimer les effets associés aux marqueurs ( $\hat{\beta}$ ), puis ces effets des marqueurs vont être utilisés pour prédire des nouveaux génotypes, comme dans la vraie vie, en appliquant  $\hat{y}_{test} = X_{test}\hat{\beta}$ . Puisque nous avons accès aux phénotypes de l'ensemble de validation, nous pouvons vérifier la qualité de prédiction en faisant la corrélation entre la valeur prédictive ( $\hat{y}_{test}$ ) et la valeur "observée" ( $y_{test}$ ).

### 7.1 Définition des ensembles d'entraînement et de test

Création de vecteurs logiques (TRUE / FALSE) pour indiquer quels individus sont contenus dans la population d'entraînement et lesquels sont dans la population de validation. Pour l'exemple, on fait ici un ajustement du modèle sur les individus du jeu d'entraînement (`in.train`), on compare les effets estimés avec la simulation et on prédit sur les individus du jeu de validation (`in.test`).

Dans la pratique, on utilise des fonctions de **validation croisée** qui vont, pour chaque partition, estimer directement la qualité de la prédiction sur le jeu de validation.

```
prop <- 0.8
in.train <- sample(c(TRUE, FALSE), size=N, replace=TRUE, prob=c(prop, 1-prop))
sum(in.train)

## [1] 399

in.test <- (! in.train)
sum(in.test)

## [1] 101
stopifnot(xor(in.train, in.test)) # vérification que chaque individu est soit dans l'ensemble d'entraînement soit dans l'ensemble de validation
```

#### 7.1.1 Entraînement

Ajuster le modèle:

```
fit <- rrBLUP::mixed.solve(y=y[in.train], Z=X[in.train,])
str(fit)
```

```
## List of 5
## $ Vu  : num 0.758
## $ Ve  : num 342
```

```

## $ beta: num [1(1d)] 14.6
## $ u : num [1:5000(1d)] 0.0801 0.00272 0.27478 0 0.03737 ...
## ..- attr(*, "dimnames")=List of 1
## ...$ : chr [1:5000] "snp3935" "snp1954" "snp6060" "snp1584" ...
## $ LL : num -1849

Comparer les estimations des paramètres avec les valeurs utilisées pour simuler les données:

# Moyenne
mu.hat <- fit$beta
c(mu, mu.hat)

## [1] 36.0 14.6

# Variance génétique
sigma.beta2.hat <- fit$Vu
c(1, sigma.beta2.hat)

## [1] 1.000 0.758

# Variance d'erreur
sigma2.hat <- fit$Ve
c(sigma2, sigma2.hat)

## [1] 291 342

# Variance génétique additive
sigma.a2.hat <- sigma.beta2.hat * 2 * sum(afs * (1 - afs))
c(sigma.a2, sigma.a2.hat)

## [1] 679 515

# Héritabilité au sens strict
h2.hat <- sigma.a2.hat / (sigma.a2.hat + sigma2)
c(h2, h2.hat)

## [1] 0.700 0.639

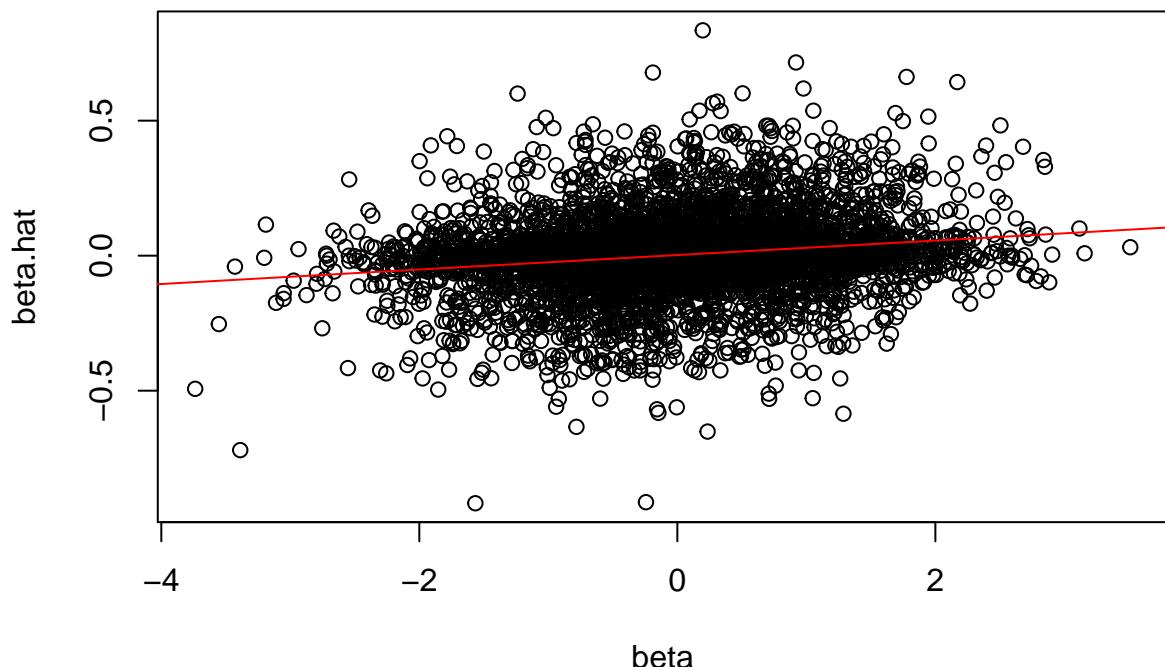
# Effet des marqueurs
beta.hat <- fit$u
# Ici on connaît la "vérité", on peut comparer les effets estimés avec les effets simulés
(tmp <- cor(beta, beta.hat))

##      [,1]
## [1,] 0.18

plot(beta, beta.hat, main=paste0("cor = ", round(tmp, 3)))
abline(lm(beta.hat ~ beta), col="red")

```

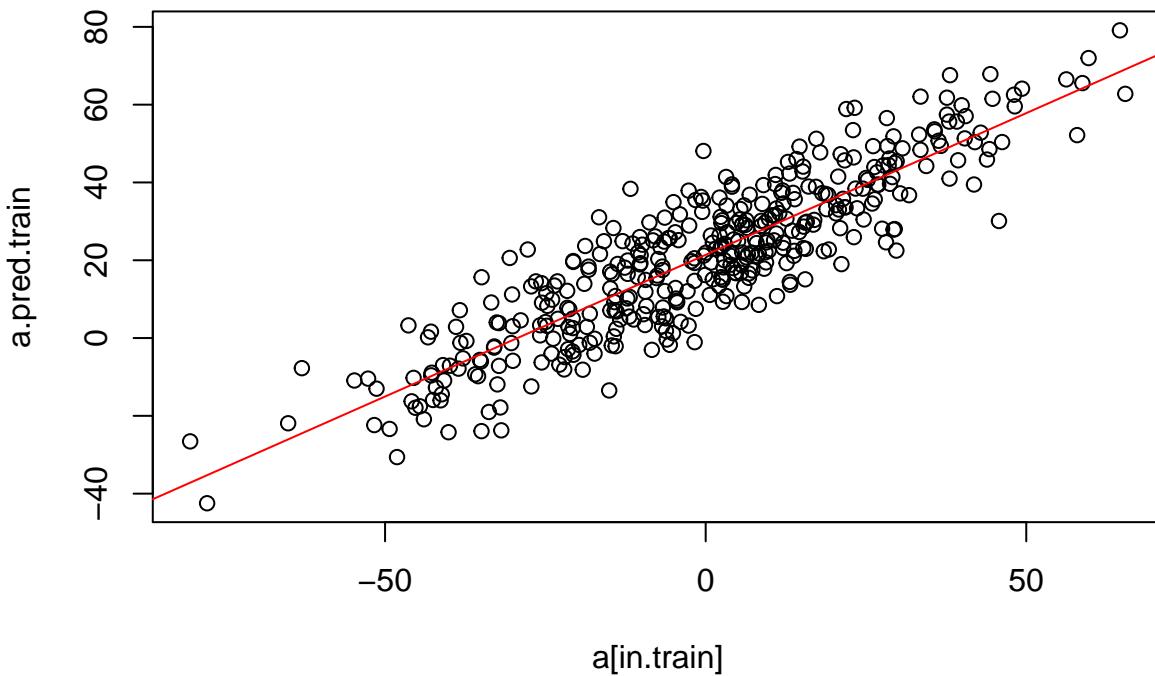
**cor = 0.18**



```
a.pred.train <- X[in.train, ] %*% beta.hat # breeding values pop entrainement = valeurs génotypiques add
(tmp <- cor(a[in.train], a.pred.train))

##      [,1]
## [1,] 0.885
plot(a[in.train], a.pred.train, main=paste0("cor = ", round(tmp, 3)))
abline(lm(a.pred.train ~ a[in.train]), col="red")
```

**cor = 0.885**



### 7.1.2 Test

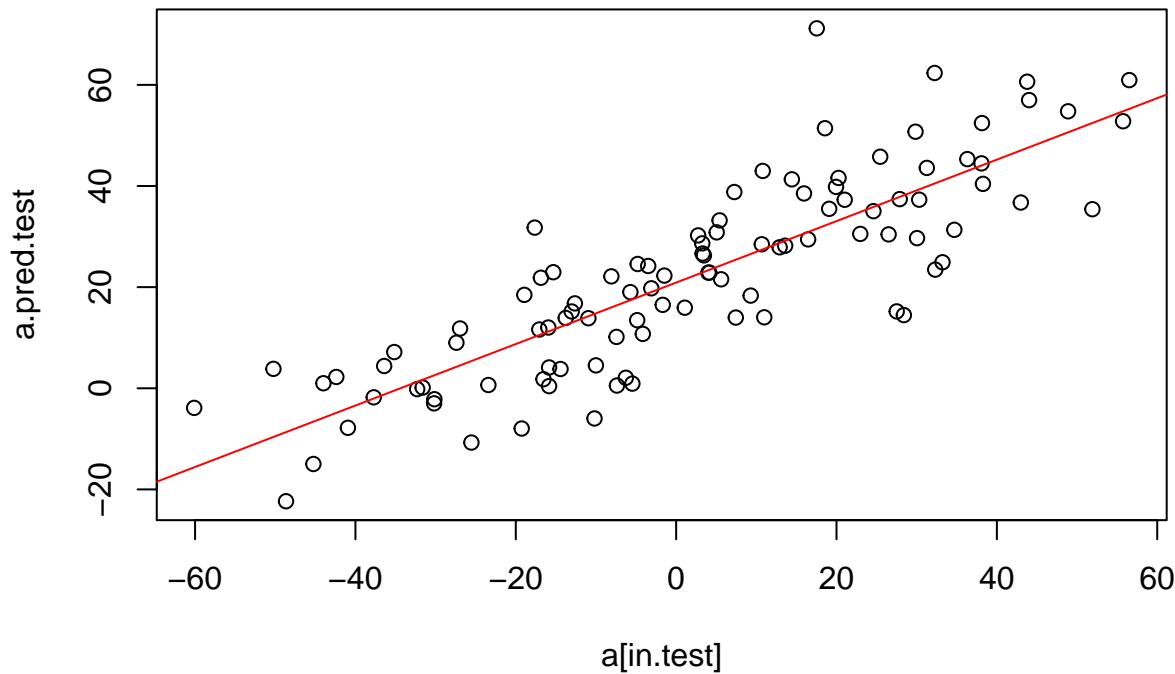
Prédire les valeurs génotypiques sur l'ensemble de test à partir des effets alléliques estimés sur l'ensemble d'entraînement:

```
a.pred.test <- X[in.test,] %*% beta.hat # breeding values pop test
(tmp <- cor(a[in.test], a.pred.test)) # précision de prédiction ou predictive ability

##      [,1]
## [1,] 0.838

plot(a[in.test], a.pred.test, main=paste0("cor = ", round(tmp, 3)))
abline(lm(a.pred.test ~ a[in.test]), col="red")
```

**cor = 0.838**



## 7.2 Validation croisée

En pratique, on définit de façon automatique toutes les partitions pour la validation croisée et on affiche directement les résultats des KxR précisions de prédiction.

### 7.2.1 Fonctions

Définir des fonctions supplémentaires est nécessaire pour utiliser le paquet `cvTools` avec la fonction `mixed.solve` du paquet `rrBLUP`:

```
rr <- function(y, Z, K=NULL, X=NULL, method="REML"){
  stopifnot(is.matrix(Z))
  out <- rrBLUP::mixed.solve(y=y, Z=Z, K=K, X=X, method=method)
  return(structure(out, class="rr"))
}
predict.rr <- function(object, newZ){
  stopifnot(is.matrix(newZ))
  out <- as.vector(newZ %*% object$u)
  if(! is.null(rownames(newZ)))
    names(out) <- rownames(newZ)
  return(out)
}
```

### 7.2.2 Partitions

La fonction `cvFolds` va permettre de définir les jeux d'entraînement et de validation de la validation croisée.

- $K = 5$  : nombre de partitions voulues, le jeu d'entraînement comportera alors  $4/5$ e des individus
- $R = 10$  : nombre de fois que la validation croisée est répétée, à chaque fois on rééchantillonne des individus différents dans les populations d'entraînement et de validation.

```

folds <- cvFolds(n=nrow(X), K=5, R=10)
dim(folds$subsets)

```

```
## [1] 500 10
```

A la fin nous aurons donc 50 valeurs de précision de prédiction, correspondant à 50 ajustements du modèle, c'est ce que fait la fonction suivante, `cvTool`. La fonction ne renvoie que la moyenne des précision de prédiction par réplicat.

### 7.2.3 Validation

```

callRR <- call("rr", y=y, Z=X)
system.time(
  out.cv <- cvTool(call=callRR, x=X, y=y, names=c("Z", "y"),
    cost=cor, folds=folds))

```

```
##      user  system elapsed
##  35.123   0.165 35.550
```

```
out.cv # one row per replicate
```

```

##          CV
## [1,] 0.604
## [2,] 0.591
## [3,] 0.629
## [4,] 0.606
## [5,] 0.646
## [6,] 0.631
## [7,] 0.643
## [8,] 0.670
## [9,] 0.646
## [10,] 0.631
mean(out.cv[, "CV"])

```

```
## [1] 0.63
```

```
sd(out.cv[, "CV"])
```

```
## [1] 0.0238
```

A vous de jouer ! Modifiez les paramètres par défauts et mesurez leur impact sur la précision de prédiction. Vous choisirez une représentation graphique qui illustrera vos conclusions.

## 8 Annexe

```
t1 <- proc.time(); t1 - t0
```

```
##      user  system elapsed
## 188.7     4.5 206.7
```

```
print(sessionInfo(), locale=FALSE)
```

```

## R version 4.0.3 (2020-10-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
```

```

## Matrix products: default
## BLAS:    /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
## LAPACK:  /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3
##
## attached base packages:
## [1] parallel   stats4    stats     graphics  grDevices utils     datasets
## [8] methods    base
##
## other attached packages:
## [1] adegenet_2.1.3      ade4_1.7-16          GenomicRanges_1.42.0
## [4] GenomeInfoDb_1.26.2  IRanges_2.24.1       S4Vectors_0.28.1
## [7] BiocGenerics_0.36.0  scrm_1.7.3-1        cvTools_0.3.2
## [10] robustbase_0.93-7   lattice_0.20-41      rrBLUP_4.6.1
## [13] knitr_1.31
##
## loaded via a namespace (and not attached):
## [1] splines_4.0.3        gtools_3.8.2         shiny_1.6.0
## [4] assertthat_0.2.1     expm_0.999-6        highr_0.8
## [7] sp_1.4-5             GenomeInfoDbData_1.2.4 LearnBayes_2.15.1
## [10] yaml_2.2.1           progress_1.2.2       pillar_1.4.7
## [13] glue_1.4.2           digest_0.6.27       promises_1.2.0.1
## [16] XVector_0.30.0       colorspace_2.0-0     Matrix_1.3-2
## [19] htmltools_0.5.1.1    httpuv_1.5.5        plyr_1.8.6
## [22] pkgconfig_2.0.3      raster_3.4-5        gmodels_2.18.1
## [25] zlibbioc_1.36.0      purrr_0.3.4         xtable_1.8-4
## [28] scales_1.1.1         gdata_2.18.0        later_1.1.0.1
## [31] tibble_3.0.6          mgcv_1.8-33        generics_0.1.0
## [34] ggplot2_3.3.3         ellipsis_0.3.1     deldir_0.2-9
## [37] magrittr_2.0.1        crayon_1.4.1       mime_0.10
## [40] evaluate_0.14         nlme_3.1-152       MASS_7.3-53.1
## [43] class_7.3-18          vegan_2.5-7        tools_4.0.3
## [46] prettyunits_1.1.1     hms_1.0.0          lifecycle_1.0.0
## [49] stringr_1.4.0         munsell_0.5.0      cluster_2.1.1
## [52] compiler_4.0.3        e1071_1.7-4        rlang_0.4.10
## [55] classInt_0.4-3        units_0.6-7        grid_4.0.3
## [58] RCurl_1.98-1.2        igraph_1.2.6       bitops_1.0-6
## [61] rmarkdown_2.6           boot_1.3-27       gtable_0.3.0
## [64] codetools_0.2-18      DBI_1.1.1          LDcorSV_1.3.3
## [67] reshape2_1.4.4         R6_2.5.0           dplyr_1.0.4
## [70] fastmap_1.1.0         seqinr_4.2-5       spdep_1.1-5
## [73] permute_0.9-5         KernSmooth_2.23-18 ape_5.4-1
## [76] stringi_1.5.3         Rcpp_1.0.6          vctrs_0.3.6
## [79] sf_0.9-7              coda_0.19-4        DEoptimR_1.0-8
## [82] spData_0.3.8          tidyselect_1.1.0    xfun_0.21

```