Exemple de simulation pour explorer la prédiction génomique

Timothée Flutre (INRA)

22/03/2018

Abstract

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

Contents

1 Contexte

Ce document fait partie de l'atelier "Prédiction Génomique" organisé et animé par Jacques David et Timothée Flutre depuis 2015, avec l'aide de Julie Fiévet et Philippe Brabant, 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.

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))
suppressPackageStartupMessages(library(GenomicRanges))
suppressPackageStartupMessages(library(rutilstimflutre))
```

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 Modèle

En se limitant à une architecture additive infinitésimale:

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

avec:

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

Cet estimateur de A_{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).

3 Simulation des données

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

3.1 Effets additifs des SNP

```
P <- 5000         # number of SNPs
sigma.beta2 <- 10^(-3) # chosen arbitrarily
beta <- rnorm(n=P, mean=0, sd=sqrt(sigma.beta2))
```

3.2 Génotypes aux SNP

3.2.1 Coalescent séquentiel avec recombinaison

```
N <- 500          # number of individuals
nb.chroms <- 10
L <- 10^6         # chromosome length, in base pairs
mu <- 10^(-8)     # neutral mutation rate in events / base / generation
u <- mu * L       # neutral mutation rate in events / chrom / gen
c.rec <- 10^(-8)  # recomb rate in events / base / gen
r <- c.rec * L    # recomb rate in events / chrom / gen
Ne <- 10^4        # effective population size
(theta <- 4 * Ne * u)  # scaled neutral mutation rate in events / chrom
```

```
## [1] 400
```

```
(rho <- 4 * Ne * r)    # scaled recomb rate in events / chrom
```

```
## [1] 400
```

```
genomes <- simulCoalescent(nb.inds=N, nb.reps=nb.chroms,
                           pop.mut.rate=theta, pop.recomb.rate=rho,
                           chrom.len=L, nb.pops=1, permute.alleles=TRUE)
```

```
## simulate according to the SCRM ...
## scrm 1000 10 -t 400 -r 400 1e+06 -SC abs -oSFS
## nb of SNPs: 30135
##  chr1  chr2  chr3  chr4  chr5  chr6  chr7  chr8  chr9 chr10
##  3071  3166  3063  3047  2997  2974  2971  3128  2679  3039
## 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 ...
```

```
stopifnot(ncol(genomes$genos) >= P)
idx.snps.tokeep <- sample.int(n=ncol(genomes$genos), size=P, replace=FALSE)
X <- genomes$genos[, idx.snps.tokeep]
dim(X)
```

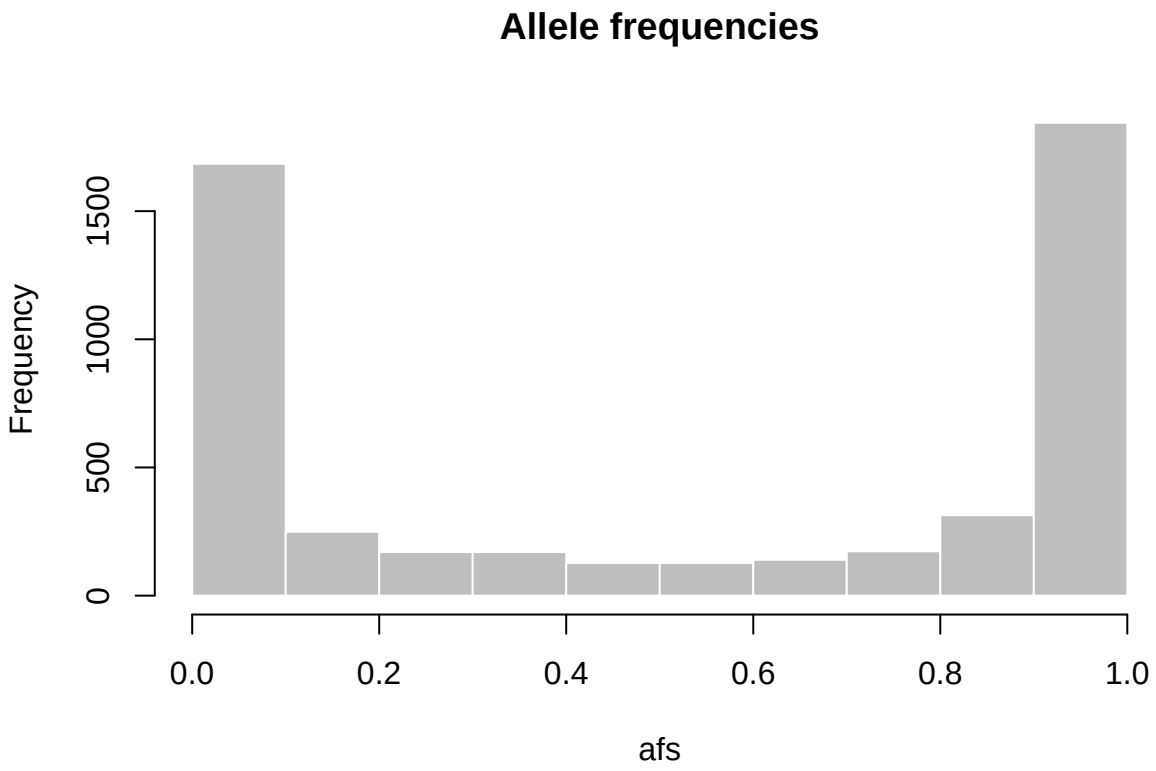
```
## [1]  500 5000
```

```
X[1:3, 1:5]
```

```
##        snp18386 snp02087 snp13786 snp02699 snp20745
## ind001        2        1        2        2        0
## ind002        2        2        2        2        0
## ind003        2        2        2        2        0
```

3.2.2 Fréquences alléliques

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



3.2.3 Déséquilibre de liaison

Sur un seul chromosome, entre un sous-ensemble de SNP, pour aller plus vite:

```
chr <- "chr1"
min.maf <- 0.15
mafs <- apply(rbind(afs, 1 - afs), 2, min)
tmp <- genomes$snp.coords[colnames(X),]
(length(snps.tokeep <- rownames(tmp[tmp$chr == chr &
                                    mafs >= min.maf,])))
```

```
## [1] 112
```

```
ld <- estimLd(X=X[,snps.tokeep],
              snp.coords=genomes$snp.coords[snps.tokeep,],
              use.ldcorsv=FALSE)
```

```
## estimate pairwise LD ...
```

```
nrow(ld)
```

```
## [1] 6216
```

```
summary(ld$cor2)
```

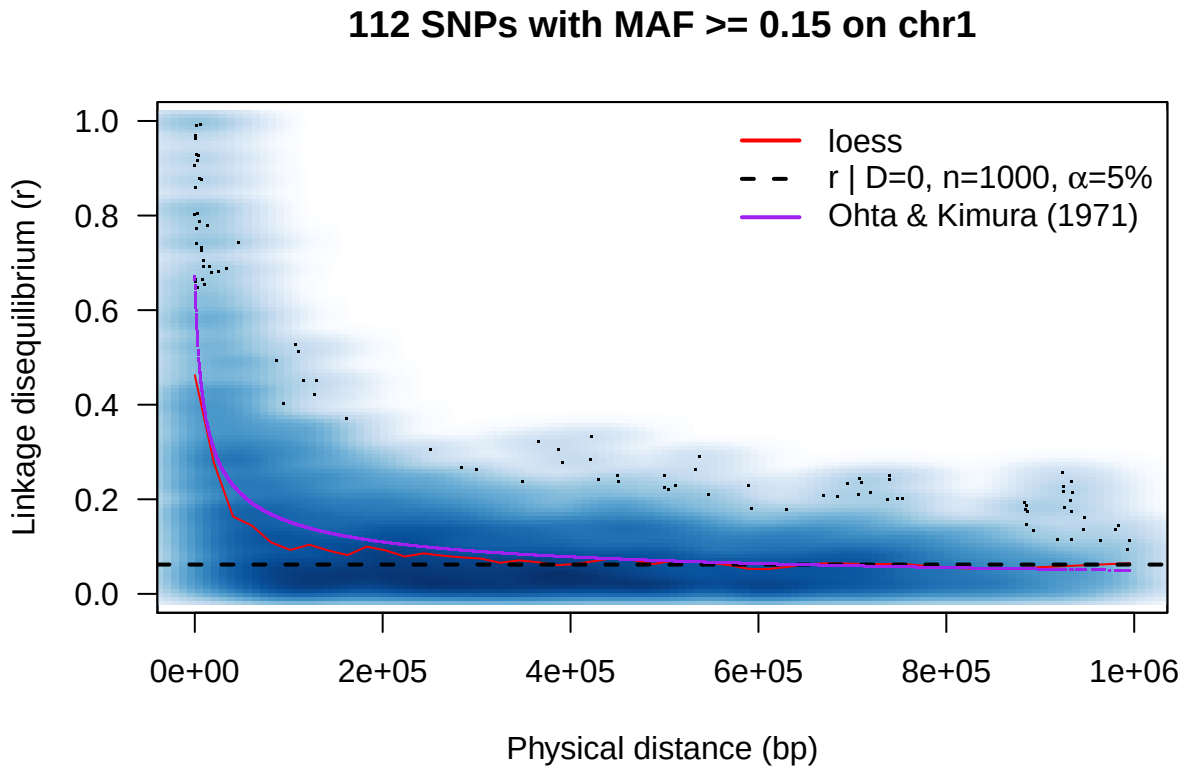
```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.000   0.001   0.006   0.026   0.019   1.000
```

```
snp.dist <- distSnpPairs(snp.pairs=ld[, c("loc1","loc2")],
                         snp.coords=genomes$snp.coords[snps.tokeep,])
```

```
## make GRanges ...
```

```
## calculate pairwise distances ...
```

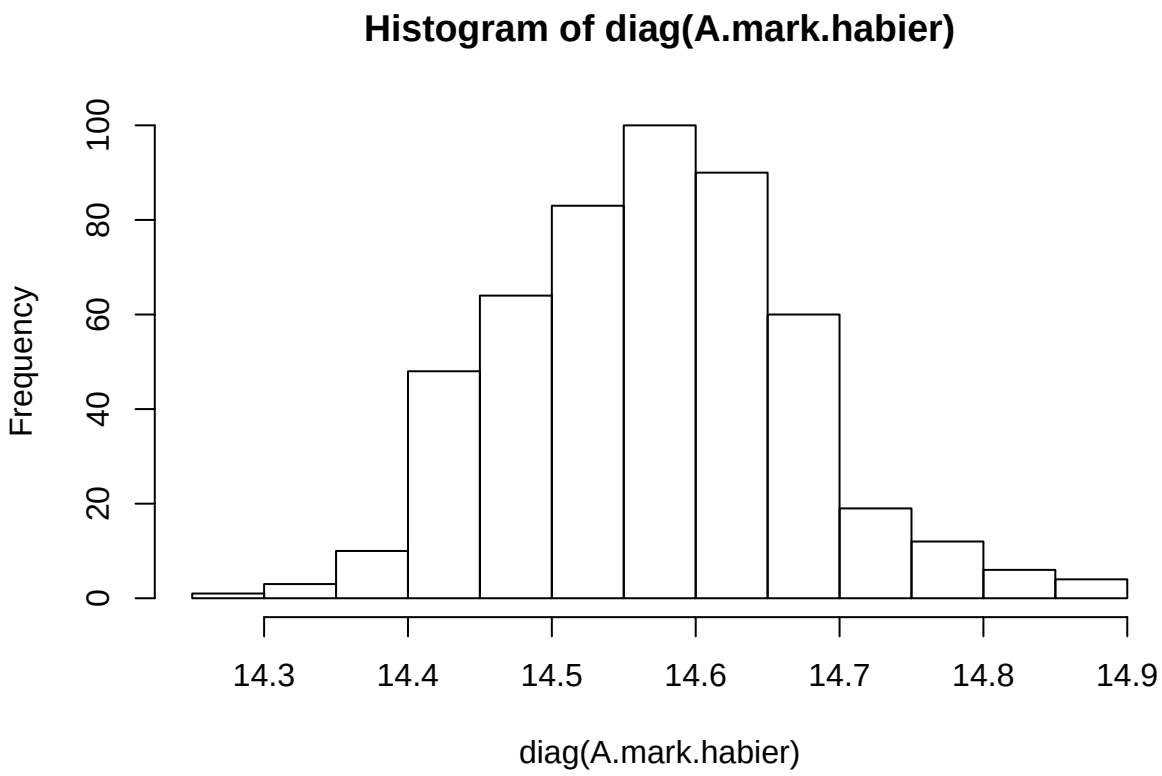
```
plotLd(snp.dist,
       sqrt(ld$cor2), estim="r",
       main=paste0(length(snps.tokeep), " SNPs with MAF >= ", min.maf,
                   " on ", chr),
       use.density=TRUE,
       span=1/20,
       sample.size=2*N,
       Ne=Ne, c=c.rec,
       add.ohta.kimura=TRUE)
```



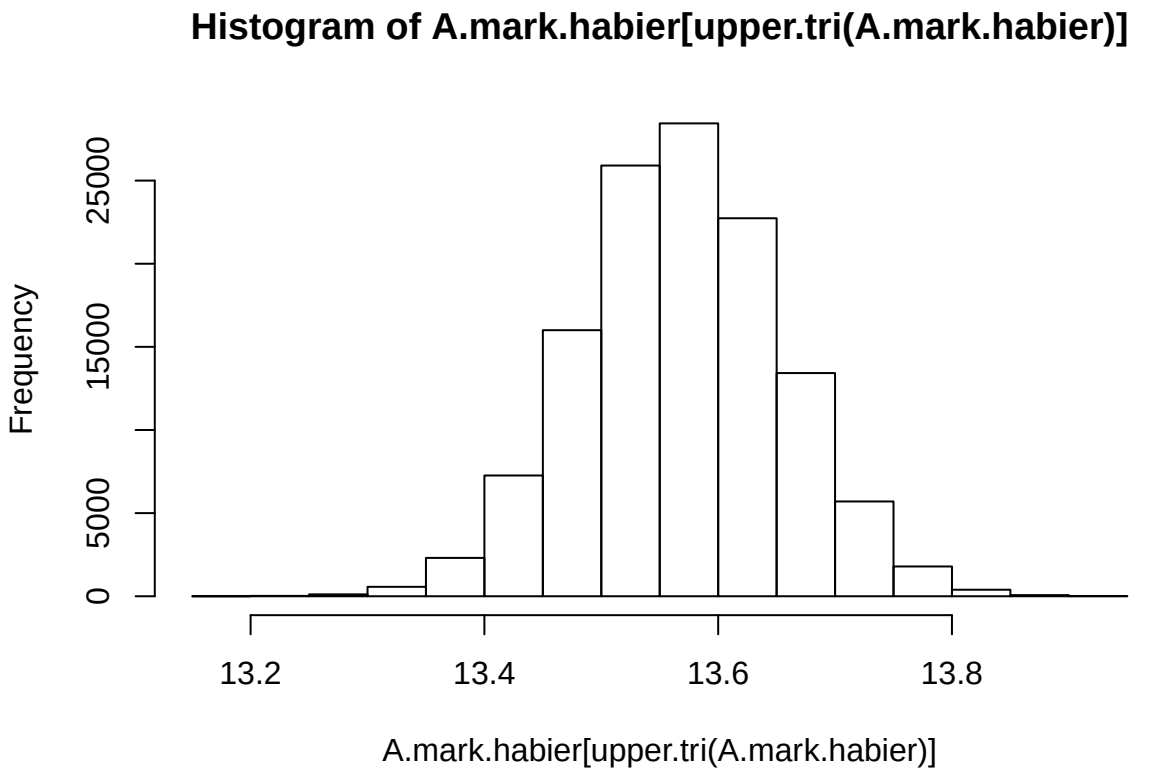
3.2.4 Relations génétiques additives

Estimateur de Habier et coll. (2007):

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



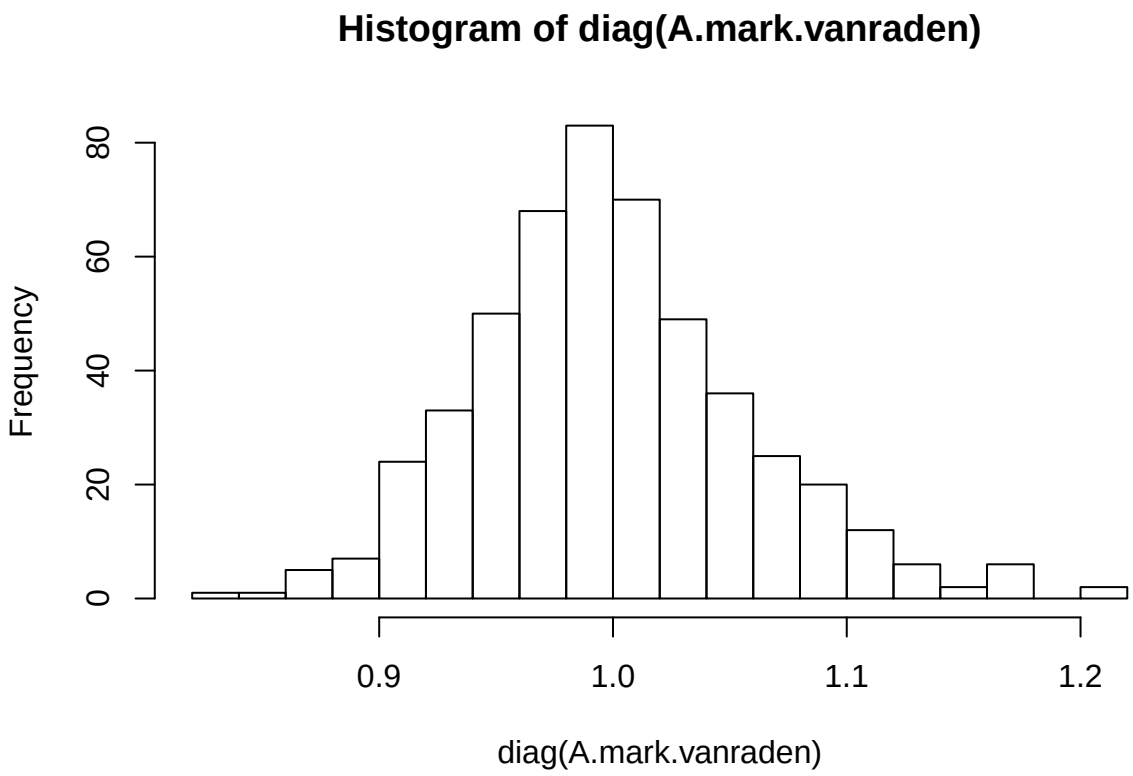

```
hist(A.mark.habier[upper.tri(A.mark.habier)])
```



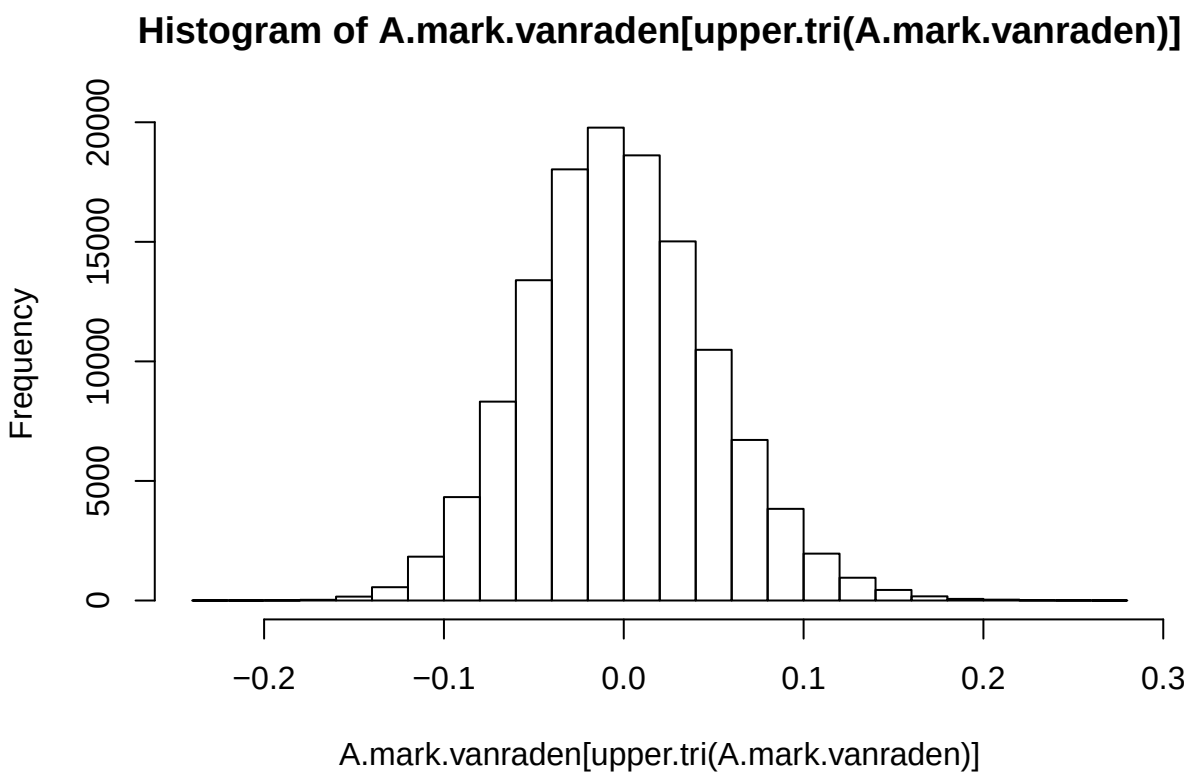
Estimateur de VanRaden (2008):

```
tmp <- matrix(rep(1, N)) %*% (2 * afs)
X.center <- X - tmp
```

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




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



3.2.5 Valeurs génotypiques additives et variance génétique additive

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:

```
X <- X.center
```

```
a <- X %*% beta
(sigma.a2 <- sigma.beta2 * 2 * sum(afs * (1 - afs)))
```

```
## [1] 0.668
```

3.3 Erreurs

```
h2 <- 0.7 # chosen arbitrarily
(sigma2 <- ((1 - h2) / h2) * sigma.a2)
```

```
## [1] 0.286
```

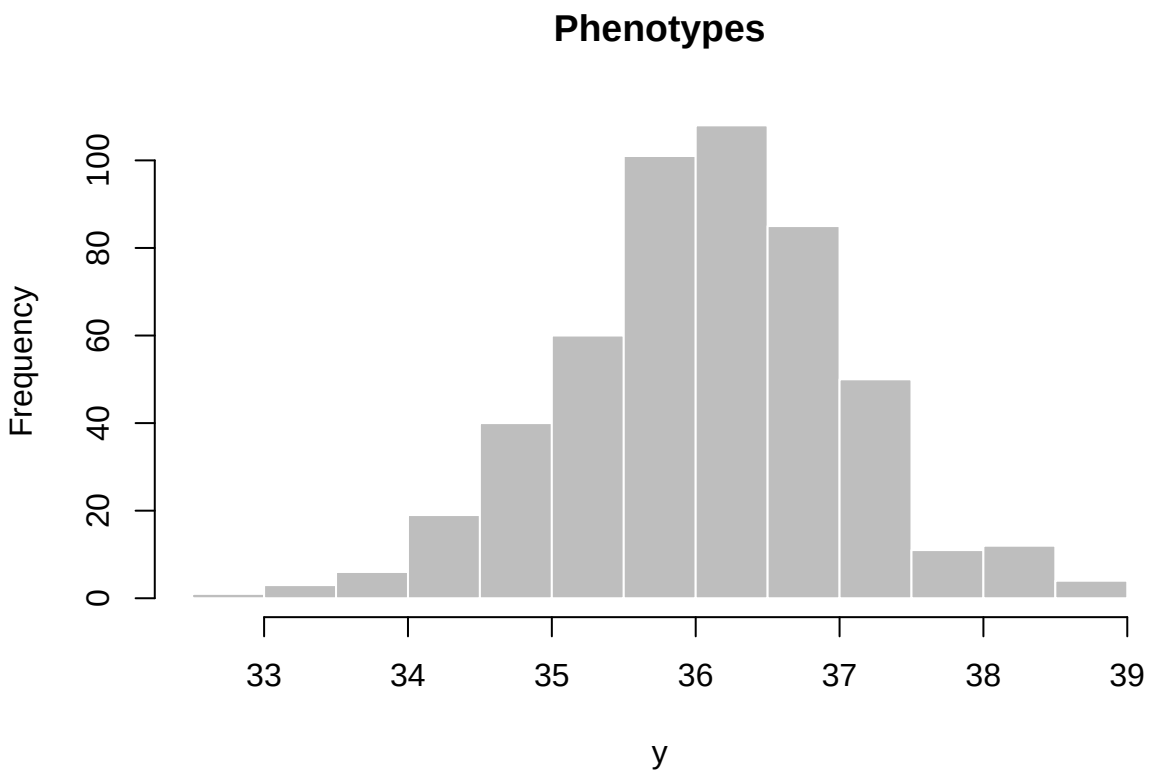
```
epsilon <- rnorm(n=N, mean=0, sd=sqrt(sigma2))
```

3.4 Phénotypes

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

```
##        V1
##  Min.   :32.8
##  1st Qu.:35.5
##  Median :36.1
##  Mean   :36.1
##  3rd Qu.:36.7
##  Max.   :38.9
```

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



4 Evaluation de la précision de prédiction

4.1 80/20

4.1.1 Définition des ensembles d'entraînement et de test

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

```
## [1] 393
```

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

```
## [1] 107
```

```
stopifnot(xor(in.train, in.test))
```

4.1.2 Entraînement

Ajuster le modèle:

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

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

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

```
## [1] 36.0 36.1
```

```
sigma.beta2.hat <- fit$Vu
c(sigma.beta2, sigma.beta2.hat)
```

```
## [1] 0.00100 0.00115
```

```
sigma2.hat <- fit$Ve
c(sigma2, sigma2.hat)
```

```
## [1] 0.286 0.205
```

```
sigma.a2.hat <- sigma.beta2.hat * 2 * sum(afs * (1 - afs))
c(sigma.a2, sigma.a2.hat)
```

```
## [1] 0.668 0.765
```

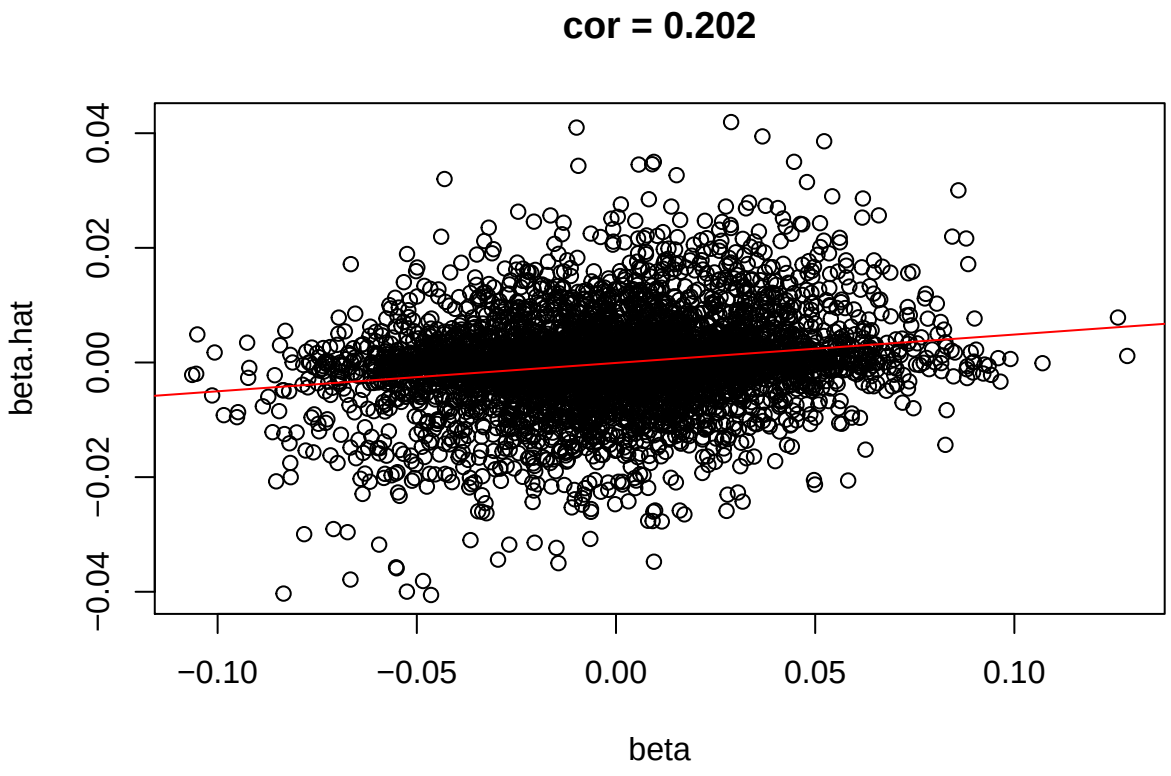
```
h2.hat <- sigma.a2.hat / (sigma.a2.hat + sigma2)
c(h2, h2.hat)
```

```
## [1] 0.700 0.728
```

```
beta.hat <- fit$u
(tmp <- cor(beta, beta.hat))
```

```
## [1] 0.202
```

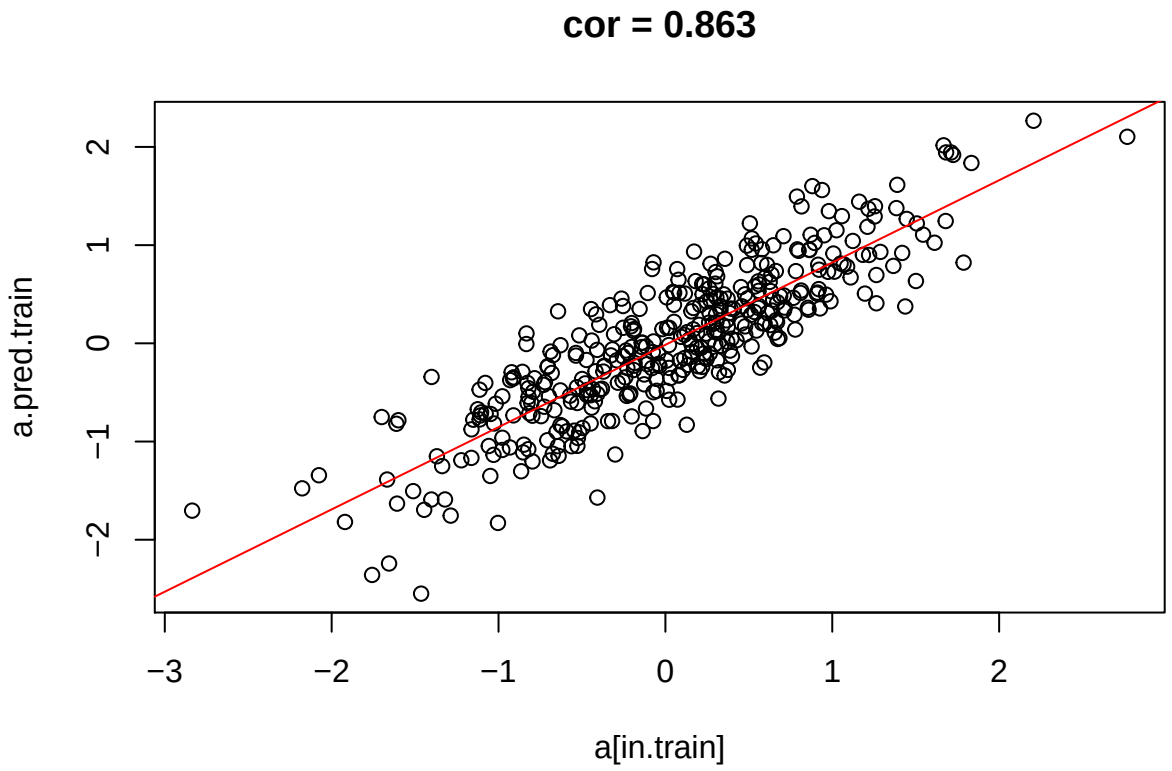
```
plot(beta, beta.hat, main=paste0("cor = ", round(tmp, 3)))
abline(lm(beta.hat ~ beta), col="red")
```




```
a.pred.train <- X[in.train,] %*% beta.hat
(tmp <- cor(a[in.train], a.pred.train))
```

```
##       [,1]
## [1,] 0.863
```

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



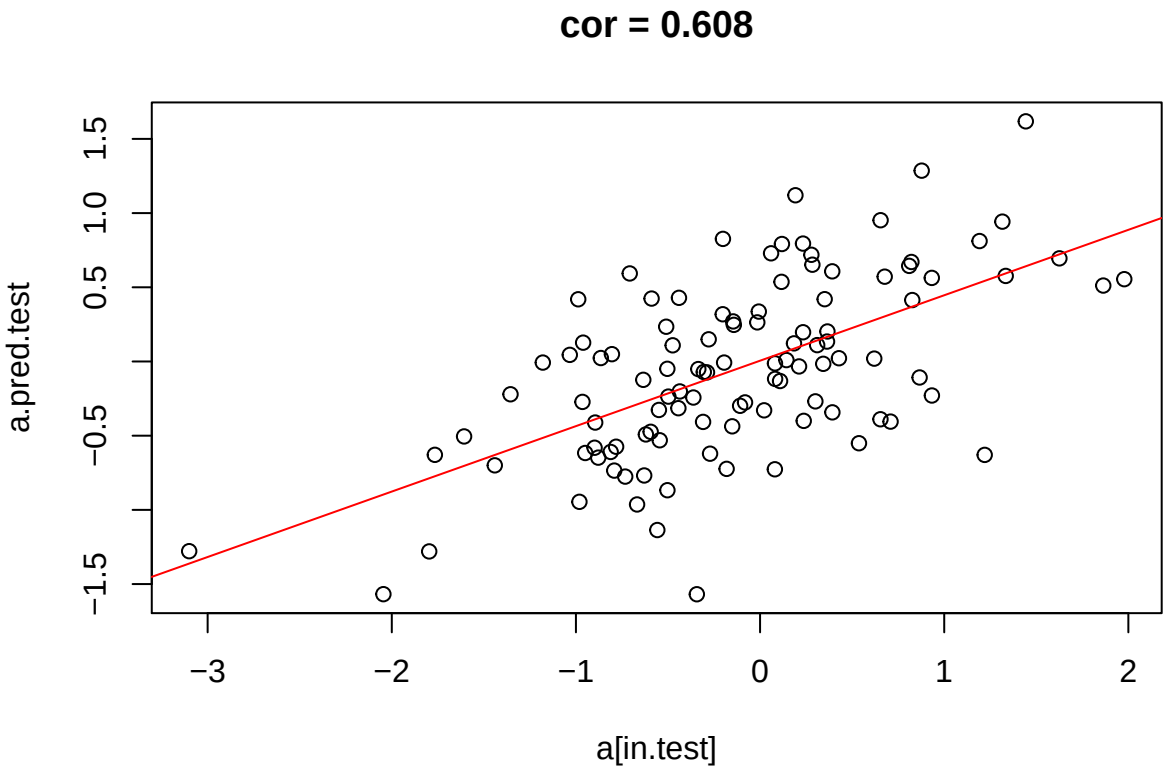
4.1.3 Test

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

```r
a.pred.test <- X[in.test,] %*% beta.hat
(tmp <- cor(a[in.test], a.pred.test))
```

```
##       [,1]
## [1,] 0.608
```

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



4.2 Validation croisée

4.2.1 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)
}
```

4.2.2 Partitions

```
folds <- cvFolds(n=nrow(X), K=5, R=10)
```

4.2.3 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
##    42.1    25.9    19.1
```

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

```
##          CV
##  [1,] 0.538
##  [2,] 0.537
##  [3,] 0.525
##  [4,] 0.519
##  [5,] 0.564
##  [6,] 0.544
##  [7,] 0.511
##  [8,] 0.509
##  [9,] 0.528
## [10,] 0.572
```

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

```
## [1] 0.535
```

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

```
## [1] 0.021
```

4.3 Formule analytique

TODO: voir Rabier et coll. (2016)

5 Annexe

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

```
##    user  system elapsed
##    61.8    29.6    40.3
```

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

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.4 LTS
##
## Matrix products: default
## BLAS: /usr/lib/openblas-base/libblas.so.3
## LAPACK: /usr/lib/libopenblasp-r0.2.18.so
##
## attached base packages:
```

```
## [1] parallel  stats4    methods   stats     graphics  grDevices utils
## [8] datasets  base
##
## other attached packages:
##  [1] rutilstimflutre_0.159.0 Rcpp_0.12.15
##  [3] lme4_1.1-14             Matrix_1.2-11
##  [5] data.table_1.10.4-3     GenomicRanges_1.30.0
##  [7] GenomeInfoDb_1.14.0     IRanges_2.12.0
##  [9] S4Vectors_0.16.0        BiocGenerics_0.24.0
## [11] scrm_1.7.2-0            cvTools_0.3.2
## [13] robustbase_0.92-8       lattice_0.20-35
## [15] rrBLUP_4.5              knitr_1.17
## [17] rmarkdown_1.8
##
## loaded via a namespace (and not attached):
##  [1] DEoptimR_1.0-8          compiler_3.4.4
##  [3] nloptr_1.0.4            XVector_0.18.0
##  [5] bitops_1.0-6            tools_3.4.4
##  [7] zlibbioc_1.24.0         digest_0.6.12
##  [9] evaluate_0.10.1         nlme_3.1-131.1
## [11] yaml_2.1.14             GenomeInfoDbData_0.99.1
## [13] stringr_1.2.0           rprojroot_1.2
## [15] grid_3.4.4              LDcorSV_1.3.2
## [17] minqa_1.2.4             magrittr_1.5
## [19] backports_1.1.1         htmltools_0.3.6
## [21] splines_3.4.4           MASS_7.3-49
## [23] KernSmooth_2.23-15      stringi_1.1.6
## [25] RCurl_1.95-4.8
```