Post-traitement des données démographiques et génotypiques simulées via SLiM et pyslim/msprime.

Delord, C.

Juin 2023

1. Objectifs:

Ce document de travail a été rédigé dans le cadre du projet POPSIZE (IRD UMR Marbec, CRPMEM de La Réunion, projet FEAMP Mesure 28).

Il détaille les commandes utilisées dans le logiciel R pour le post-traitement des données issues de simulations générées à l'aide du logiciel SLiM et des librairies Python *pyslim*/*msprime* dans le cadre du projet POPSIZE. Les scripts permettant de générer ces données simulées sont consultables via https://github.com/ChrystelleDelord/POPSIZE-Project-SLiM_Scripts). Pour plus d'informations ou en cas de question, ne pas hésiter à envoyer un e-mail à l'adresse suivante: chrys.delord@gmail.com.

Les commandes listées ci-dessous permettent d'accomplir les actions suivantes:

- Importer dans R le fichier .vcf global de référence, contenant l'information génétique de tous les individus à tous les loci simulés puis "échantillonnés" à l'aide du logiciel SLiM et des librairies Python *pyslim*/*msprime*.
- Importer également une partie des informations des autres fichiers de sorties de la simulation, et notamment `SLiM_samples.tsv` qui renferme les métadonnées relatives à tous les individus présents dans le fichier .vcf global.
- Procéder à des sous-échantillonnages d'individus et de loci parmi l'ensemble sauvegardé en fin de simulation. Dans le cas présent, on génère trois jeux de données contenant tous les individus et respectivement 30000 (i.e., tous les loci), puis 10000 et 1000 loci sous-échantillonnés au hasard. Puis, à partir de chacun de ces 3 jeux de données, on génère 4 sous-jeux de données contenant un nombre croissant d'individus préalablement sous-échantillonnés au hasard depuis le fichier `SLiM_samples.tsv`.
- Convertir le format des 12 jeux de données ainsi obtenus au total. Ces jeux de données seront convertis en format genepop (fichiers d'extension .gen) pour être lisibles par le logiciel *NeEstimator v2* *Do et al. 2014*, et en format PLINK (fichiers d'extension .bed et .map) pour être lisibles par le logiciel *GONE* *Santiago et al. 2020*. Un fichier définissant la population d'appartenance de chaque individu est également généré pour chacun des 4 sous-échantillonnages d'individus pour être lisible, en parallèle du fichier .vcf global, par le logiciel *GADMA* *Noskova et al. 2020*.
- Visualiser de manière préliminaire les informations liées aux différents sous-échantillonnages d'individus et notamment, la quantité de couples apparentés que l'on peut y retrouver (en lien avec les applications de type *close-kin mark recapture*).

2. Post-traitement et conversion de données génotypiques pour le test de différents logiciels d'estimation de la taille efficace:

Chargement des librairies et des données:

```
library(dartR)
library(CKMRpop)
library(psych)
library(tidyverse)

str1 = "5000"
str2 = "m005"
migrate = 0.05
```

```
# ------------------------------------------------------------------
# CHARGEMENT DES DONNEES:
# ------------------------------------------------------------------
# -- Données génotypiques fichier .vcf global de référence (sortie de SLiM et pysl
im/msprime):
start_time = Sys.time()
dat <- gl.read.vcf(paste0("POPSIZE_pyslim_output_Cohort", str1, "_", str2, ".vcf"
))
end_time = Sys.time()
# end_time - start_time # Affichage de la durée de chargement du jeu de données .v
cf global.
```

```
# -- Métadonnées relatives aux individus échantillonnés durant la simulation:
sample_file <- file.path(getwd(), "SLiM_samples.tsv")
samples <- vroom::vroom(file = sample_file, delim = "\t", col_types = "ccccccc") %
>%
  mutate(samp_years_list_post = str_split(syears_post, "  *"), samp_years_list_pos
t = map(.x = samp_years_list_post,
  .f = function(x) as.integer(x)), sampling_pop = str_split(pop_post, "  *"), samp
ling_pop = map(.x = sampling_pop,
  .f = function(x) as.integer(x))) %>% select(-syears_pre, -syears_post, -syears_d
ur, -pop_pre, -pop_post, -pop_dur) %>%
  extract(ID, into = c("sex", "born_year", "born_pop"), regex = "^([MF])([0-9]+)_
([0-9]+)", remove = FALSE, convert = TRUE)
samples <- samples %>% unnest(samp_years_list_post, sampling_pop)
```

```
## Warning: unnest() has a new interface. See ?unnest for details.
## Try `df %>% unnest(c(samp_years_list_post, sampling_pop))`, with `mutate()` if
needed
```

```
samples <- samples %>% mutate(samples, age_at_sampling = samp_years_list_post - bo
rn_year)

print("Notre jeu de données génotypique complet en format genlight:")
```

```
## [1] "Notre jeu de données génotypique complet en format genlight:"
```

```
show(dat)
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 14,856 genotypes,  30,000 binary SNPs, size: 134.4 Mb
##  0 (0 %) missing data
##
##  // Basic content
##    @gen: list of 14856 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  14856 individual labels
##    @loc.names:  30000 locus labels
##    @loc.all:  30000 alleles
##    @chromosome: factor storing chromosomes of the SNPs
##    @position: integer storing positions of the SNPs
##    @pop: population of each individual (group size range: 14856-14856)
##    @other: a list containing: loc.metrics  loc.metrics.flags  verbose  history
ind.metrics
```

```
print("Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnage:"
)
```

```
## [1] "Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnag
e:"
```

```
table(samples$samp_years_list_post)
```

```
##
##   90   91   92   93   94   95   96   97   98   99  100
## 1534 1591 1546 1500 1480 1580 1560 1506 1540 1520 1591
```

```
print("Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnage e
t par sous-population:")
```

```
## [1] "Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnage
et par sous-population:"
```

```
table(samples$sampling_pop, samples$samp_years_list_post)
```

```
##
##      90  91  92  93  94  95  96  97  98  99 100
##   1 768 767 780 754 711 802 761 775 774 753 806
##   2 766 824 766 746 769 778 799 731 766 767 785
```

```
print("Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnage e
t par classe d'âge:")
```

```
## [1] "Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnage
et par classe d'âge:"
```

```
table(samples$age_at_sampling, samples$samp_years_list_post)
```

```
##
##       90  91  92  93  94  95  96  97  98  99 100
##   1  548 546 560 509 473 580 547 525 530 521 579
##   2  333 320 322 324 340 328 326 312 340 330 333
##   3  218 230 224 220 211 224 236 211 220 227 235
##   4  160 171 147 166 159 156 155 136 165 162 161
##   5   83 118 112  97 122 106 116 120  88  98 103
##   6   81  87  74  88  72  73  64  84  74  55  56
##   7   48  53  45  44  48  54  55  54  58  51  51
##   8   28  42  40  21  29  31  29  32  33  36  34
##   9   22  16  17  19  12  11  17  17  17  24  21
##   10   9   7   3   7   9  10  11   8   9   7   9
##   11   3   1   1   2   2   4   3   6   3   3   6
##   12   1   0   1   2   2   2   1   0   1   3   3
##   13   0   0   0   1   1   0   0   1   2   3   0
##   14   0   0   0   0   0   1   0   0   0   0   0
```

Sous-échantillonnage successif des loci:

Notre objet `dat` contient 14856 individus et 30000 loci issus de SLiM et *pyslim*/*msprime*. A partir de `dat`, nous créons `dat_L10000` qui contient 14856 individus et 10000 loci. A partir de `dat_L10000`, nous créons `dat_L1000` qui contient 14856 individus et 1000 loci.

Cette procédure permet par exemple de tester, avec les mêmes individus, l'influence du nombre de loci sur la qualité des estimations de taille efficace.

```
# ------------------------------------------------------------------
# SOUS-ECHANTILLONNAGE DES LOCI:
# ------------------------------------------------------------------
dat_L10000 <- gl.keep.loc(dat, loc.list = sample(dat$loc.names, 10000, replace = F
ALSE, prob = NULL), first = NULL, last = NULL, verbose = NULL)
dat_L1000 <- gl.keep.loc(dat_L10000, loc.list = sample(dat_L10000$loc.names, 1000,
replace = FALSE, prob = NULL), first = NULL, last = NULL, verbose = NULL)
```

Sous-échantillonnage des individus:

Notre fichier .vcf de référence contient 14856 individus échantillonnés entre les pas de temps 90 à 100, dans chacune des classes d'âge 1 à 15 ans et dans chacune des sous-populations. Afin de tester l'influence de stratégies d'échantillonnage réalistes sur la qualité des estimations de taille efficace, il va nous falloir procéder à des sous-échantillonnage afin de conserver uniquement l'information génotypique d'individus reflétant une véritable campagne d'échantillonnage sur le terrain. Par exemple, il est peu probable de pouvoir échantillonner des individus sur 11 pas de temps consécutifs dans la réalité. On peut, en revanche, se concentrer sur un pas de temps donné pour réfléter un échantillonnage localisé dans le temps.

Nous allons donc ici, à titre d'exemple, procéder à plusieurs sous-échantillonnages de taille croissante en nombre d'individus, mais focalisés sur le dernier pas de temps simulé dans SLiM: le pas de temps 100, qui correspond au plus récent (=temps présent). Par contre, nous sous-échantillonnerons le même nombre d'individus dans chacune des deux sous-populations.

En outre, nous souhaitons que les mêmes individus soient sous-échantillonnés dans chacun jeu de données à 30000, 10000 et 1000 loci. Ceci permettra, in fine, de comparer l'influence du nombre de loci sur la qualité des estimations de taille efficace sans intégrer de "bruit" lié à l'utilisation d'individus différents. Réciproquement, tous les individus seront génotypés sur les mêmes 30000, 10000 ou 1000 loci.

A l'aide de l'objet `samples` issu du fichier `SLiM_samples.tsv`, on collecte un nombre d'individus variant entre 0.5%, 2.0% et 5.0% de la taille efficace locale simulée dans le logiciel SLiM, dans chacune des sous-populations. Ici, ces tailles de sous-échantillonnages correspondent à 14, 55 et 139 individus par sous-population, respectivement. On collecte également un sous-échantillon "typique" de 50 individus, là encore dans chacune des sous-populations.

```
# ----------------------------------------------------------------
# SOUS-ECHANTILLONNAGE DES INDIVIDUS:
# ----------------------------------------------------------------

# Chargement des valeurs de taille efficace démographique calculées durant la phas
e de simulation à l'aide du logiciel SLiM.
if(file.exists("SLiM_demo_table.tsv")) {
  SLiM_demo <- read.table("SLiM_demo_table.tsv", sep ='\t', header = T)
  Ne_demo = (harmonic.mean(SLiM_demo$ne_demo))/2
} else {
  Ne_demo = round(0.559*strtoi(str1))
}

# On effectue 4 sous-échantillonnages au temps présent, avec un nombre variable
 d'individus par sous-population:
Ssize_005percent <- round(0.005*Ne_demo) # Une taille d'échantillon correspondant
 à 0.5% de la taille efficace locale: 14 individus.
Ssize_02percent <- round(0.02*Ne_demo) # Une taille d'échantillon correspondant à
 2.0% de la taille efficace locale: 55 individus.
Ssize_05percent <- round(0.05*Ne_demo) # Une taille d'échantillon correspondant à
 5.0% de la taille efficace locale: 139 individus.
Ssize_typical <- 50 # Une taille d'échantillon "typique" de 50 individus.

# Sous-échantillonnage des noms (ID) d'un nombre d'individus par sous-population c
orrespondant à 0.5% de la taille efficace locale.
subsamp_ID_005 <- samples %>% filter(samp_years_list_post %in% c(100), sampling_po
p %in% c(1,2)) %>%
  group_by(samp_years_list_post, sampling_pop) %>% sample_n(if(n() < Ssize_005perc
ent) n() else Ssize_005percent)
table(subsamp_ID_005$age_at_sampling, subsamp_ID_005$samp_years_list_post) # Nb
 d'échantillons par classe d'âge.
table(subsamp_ID_005$sampling_pop, subsamp_ID_005$samp_years_list_post) # Nb d'éch
antillons par population.
subsamp_ID_005 %>% group_by(ID) %>% filter(n() > 1) %>% print(n = 20)

# Sous-échantillonnage des noms (ID) d'un nombre d'individus correspondant à 2.0%
 de la taille efficace locale (par sous-population).
subsamp_ID_02 <- samples %>% filter(samp_years_list_post %in% c(100), sampling_pop
%in% c(1,2)) %>%
  group_by(samp_years_list_post, sampling_pop) %>% sample_n(if(n() < Ssize_02perce
nt) n() else Ssize_02percent)
#table(subsamp_ID_02$age_at_sampling, subsamp_ID_02$samp_years_list_post)
subsamp_ID_02 %>% group_by(ID) %>% filter(n() > 1) %>% print(n = 20)

# Sous-échantillonnage des noms (ID) d'un nombre d'individus correspondant à 5.0%
```

```
 de la taille efficace locale (par sous-population).
subsamp_ID_05 <- samples %>% filter(samp_years_list_post %in% c(100), sampling_pop
%in% c(1,2)) %>%
  group_by(samp_years_list_post, sampling_pop) %>% sample_n(if(n() < Ssize_05perce
nt) n() else Ssize_05percent)
#table(subsamp_ID_05$age_at_sampling, subsamp_ID_05$samp_years_list_post)
subsamp_ID_05 %>% group_by(ID) %>% filter(n() > 1) %>% print(n = 20)

# Sous-échantillonnage des noms (ID) de 50 individus.
subsamp_ID_typical <- samples %>% filter(samp_years_list_post %in% c(100), samplin
g_pop %in% c(1,2)) %>%
  group_by(samp_years_list_post, sampling_pop) %>% sample_n(if(n() < Ssize_typica
l) n() else Ssize_typical)
#table(subsamp_ID_typical$age_at_sampling, subsamp_ID_typical$samp_years_list_pos
t)
subsamp_ID_typical %>% group_by(ID) %>% filter(n() > 1) %>% print(n = 20)
```

Extraction des génotypes et informations des individus sous-échantillonnés:

Nous avons à disposition d'une part, nos jeux de données génotypiques de 30000, 10000 ou 1000 loci pour l'ensemble des 14856 individus, d'autre part des listes (e.g., `subsamp_ID_005`) comportant les identifiants des individus que l'on souhaite extraire de chacun de ces jeux de données. Les lignes de commandes ci-dessous procèdent à plusieurs manipulations pour assurer une extraction correcte de ces échantillons et de leur information, notamment la population dans laquelle ils ont été échantillonnés.

```
# ------------------------------------------------------------------
# EXPORT D'UN FICHIER (strata file) CONTENANT LA POPULATION DANS LAQUELLE CHAQUE I
NDIVIDU A ETE SOUS-ECHANTILLONNE
# (à partir de l'identifiant unique de haque individu et des informations contenue
s dans l'objet 'samples')
# ------------------------------------------------------------------

stratat_005 <- as.data.frame(subsamp_ID_005 %>% ungroup() %>% select(ID, sampling_
pop))
names(stratat_005) <- c("sample", "pop")
write.table(stratat_005, paste0("pop_file_", str1, "_", str2,".txt") ,sep="\t", ro
w.names=FALSE, quote=FALSE)

stratat_02 <- as.data.frame(subsamp_ID_02 %>% ungroup() %>% select(ID, sampling_po
p))
names(stratat_02) <- c("sample", "pop")
write.table(stratat_02, paste0("pop_file_", str1, "_", str2, "_s02.txt") ,sep="\t"
, row.names=FALSE, quote=FALSE)

stratat_05 <- as.data.frame(subsamp_ID_05 %>% ungroup() %>% select(ID, sampling_po
p))
names(stratat_05) <- c("sample", "pop")
write.table(stratat_05, paste0("pop_file_", str1, "_", str2, "_s05.txt") ,sep="\t"
, row.names=FALSE, quote=FALSE)

stratat_typical <- as.data.frame(subsamp_ID_typical %>% ungroup() %>% select(ID, s
ampling_pop))
names(stratat_typical) <- c("sample", "pop")
```

```
write.table(stratat_typical, paste0("pop_file_", str1, "_", str2, "_stypical.txt")
,sep="\t", row.names=FALSE, quote=FALSE)
```

On extrait à présent les groupes individus sous-échantillonnés de chacun des jeux de données `dat` (30000 loci), `dat_L10000` (10000 loci) et `dat_L1000` (1000 loci). Pour chacun d'entre eux, 4 nouveaux jeux de données sont ainsi générés.

Puis, à chacun des 12 jeux de données finaux, on réassigne l'information de population d'origine à chacun des individus qu'il contient, en vue des conversions de format de données à venir.

```
# ------------------------------------------------------------
# EXTRACTION DEPUIS LE JEU DE DONNEES A 30000 LOCI:
# ------------------------------------------------------------

ind.list_005 <- dat$ind.names[dat$ind.names %in% subsamp_ID_005$ID] # On liste les
identifiants uniques des individus sous-échantillonnés.
dat_s005 <- gl.keep.ind(dat, ind.list=ind.list_005, recalc = FALSE, mono.rm = FALS
E, verbose = NULL)
dat_s005@other$ind.metrics$pop <- pop(dat_s005)
dat_s005$other$ind.metrics$pop <- ifelse(stratat_005$sample %in% dat_s005$other$in
d.metrics$id, stratat_005$pop, dat_s005$other$ind.metrics$pop)
# Mise à jour de l'information de population:
dat_s005 <- gl.reassign.pop(dat_s005, as.pop='pop',verbose=3)
popNames(dat_s005)

# On répète le même processus pour les 11 jeux de données suivants.
ind.list_02 <- dat$ind.names[dat$ind.names %in% subsamp_ID_02$ID]
dat_s02 <- gl.keep.ind(dat, ind.list=ind.list_02, recalc = FALSE, mono.rm = FALSE,
verbose = NULL)
dat_s02@other$ind.metrics$pop <- pop(dat_s02)
dat_s02$other$ind.metrics$pop <- ifelse(stratat_02$sample %in% dat_s02$other$ind.m
etrics$id, stratat_02$pop, dat_s02$other$ind.metrics$pop)
dat_s02 <- gl.reassign.pop(dat_s02, as.pop='pop',verbose=3)
popNames(dat_s02)

ind.list_05 <- dat$ind.names[dat$ind.names %in% subsamp_ID_05$ID]
dat_s05 <- gl.keep.ind(dat, ind.list=ind.list_05, recalc = FALSE, mono.rm = FALSE,
verbose = NULL)
dat_s05@other$ind.metrics$pop <- pop(dat_s05)
dat_s05$other$ind.metrics$pop <- ifelse(stratat_05$sample %in% dat_s05$other$ind.m
etrics$id, stratat_05$pop, dat_s05$other$ind.metrics$pop)
dat_s05 <- gl.reassign.pop(dat_s05, as.pop='pop',verbose=3)
popNames(dat_s05)

ind.list_typ <- dat$ind.names[dat$ind.names %in% subsamp_ID_typical$ID]
dat_styp <- gl.keep.ind(dat, ind.list=ind.list_typ, recalc = FALSE, mono.rm = FALS
E, verbose = NULL)
dat_styp@other$ind.metrics$pop <- pop(dat_styp)
dat_styp$other$ind.metrics$pop <- ifelse(stratat_typical$sample %in% dat_styp$othe
r$ind.metrics$id, stratat_typical$pop, dat_styp$other$ind.metrics$pop)
dat_styp <- gl.reassign.pop(dat_styp, as.pop='pop',verbose=3)
popNames(dat_styp)

# ------------------------------------------------------------
# EXTRACTION DEPUIS LE JEU DE DONNEES A 10000 LOCI:
# ------------------------------------------------------------
```

```
ind.list_005 <- dat_L10000$ind.names[dat_L10000$ind.names %in% subsamp_ID_005$ID]
dat_L10000_s005 <- gl.keep.ind(dat_L10000, ind.list=ind.list_005, recalc = FALSE,
 mono.rm = FALSE, verbose = NULL)
dat_L10000_s005@other$ind.metrics$pop <- pop(dat_L10000_s005)
dat_L10000_s005$other$ind.metrics$pop <- ifelse(stratat_005$sample %in% dat_L10000
_s005$other$ind.metrics$id, stratat_005$pop, dat_L10000_s005$other$ind.metrics$po
p)
dat_L10000_s005 <- gl.reassign.pop(dat_L10000_s005, as.pop='pop',verbose=3)
popNames(dat_L10000_s005)

ind.list_02 <- dat_L10000$ind.names[dat_L10000$ind.names %in% subsamp_ID_02$ID]
dat_L10000_s02 <- gl.keep.ind(dat_L10000, ind.list=ind.list_02, recalc = FALSE, mo
no.rm = FALSE, verbose = NULL)
dat_L10000_s02@other$ind.metrics$pop <- pop(dat_L10000_s02)
dat_L10000_s02$other$ind.metrics$pop <- ifelse(stratat_02$sample %in% dat_L10000_s
02$other$ind.metrics$id, stratat_02$pop, dat_L10000_s02$other$ind.metrics$pop)
dat_L10000_s02 <- gl.reassign.pop(dat_L10000_s02, as.pop='pop',verbose=3)
popNames(dat_L10000_s02)

ind.list_05 <- dat_L10000$ind.names[dat_L10000$ind.names %in% subsamp_ID_05$ID]
dat_L10000_s05 <- gl.keep.ind(dat_L10000, ind.list=ind.list_05, recalc = FALSE, mo
no.rm = FALSE, verbose = NULL)
dat_L10000_s05@other$ind.metrics$pop <- pop(dat_L10000_s05)
dat_L10000_s05$other$ind.metrics$pop <- ifelse(stratat_05$sample %in% dat_L10000_s
05$other$ind.metrics$id, stratat_05$pop, dat_L10000_s05$other$ind.metrics$pop)
dat_L10000_s05 <- gl.reassign.pop(dat_L10000_s05, as.pop='pop',verbose=3)
popNames(dat_L10000_s05)

ind.list_typ <- dat_L10000$ind.names[dat_L10000$ind.names %in% subsamp_ID_typical
$ID] # lists individuals to keep.
dat_L10000_styp <- gl.keep.ind(dat_L10000, ind.list=ind.list_typ, recalc = FALSE,
 mono.rm = FALSE, verbose = NULL)
dat_L10000_styp@other$ind.metrics$pop <- pop(dat_L10000_styp)
dat_L10000_styp$other$ind.metrics$pop <- ifelse(stratat_typical$sample %in% dat_L1
0000_styp$other$ind.metrics$id, stratat_typical$pop, dat_L10000_styp$other$ind.met
rics$pop)
dat_L10000_styp <- gl.reassign.pop(dat_L10000_styp, as.pop='pop',verbose=3)
popNames(dat_L10000_styp)

# ------------------------------------------------------------------
# EXTRACTION DEPUIS LE JEU DE DONNEES A 1000 LOCI:
# ------------------------------------------------------------------

ind.list_005 <- dat_L1000$ind.names[dat_L1000$ind.names %in% subsamp_ID_005$ID]
dat_L1000_s005 <- gl.keep.ind(dat_L1000, ind.list=ind.list_005, recalc = FALSE, mo
no.rm = FALSE, verbose = NULL)
dat_L1000_s005@other$ind.metrics$pop <- pop(dat_L1000_s005)
dat_L1000_s005$other$ind.metrics$pop <- ifelse(stratat_005$sample %in% dat_L1000_s
005$other$ind.metrics$id, stratat_005$pop, dat_L1000_s005$other$ind.metrics$pop)
dat_L1000_s005 <- gl.reassign.pop(dat_L1000_s005, as.pop='pop',verbose=3)
popNames(dat_L1000_s005)

ind.list_02 <- dat_L1000$ind.names[dat_L1000$ind.names %in% subsamp_ID_02$ID]
dat_L1000_s02 <- gl.keep.ind(dat_L1000, ind.list=ind.list_02, recalc = FALSE, mon
o.rm = FALSE, verbose = NULL)
dat_L1000_s02@other$ind.metrics$pop <- pop(dat_L1000_s02)
```

```
dat_L1000_s02$other$ind.metrics$pop <- ifelse(stratat_02$sample %in% dat_L1000_s02
$other$ind.metrics$id, stratat_02$pop, dat_L1000_s02$other$ind.metrics$pop)
dat_L1000_s02 <- gl.reassign.pop(dat_L1000_s02, as.pop='pop',verbose=3)
popNames(dat_L1000_s02)

ind.list_05 <- dat_L1000$ind.names[dat_L1000$ind.names %in% subsamp_ID_05$ID]
dat_L1000_s05 <- gl.keep.ind(dat_L1000, ind.list=ind.list_05, recalc = FALSE, mon
o.rm = FALSE, verbose = NULL)
dat_L1000_s05@other$ind.metrics$pop <- pop(dat_L1000_s05)
dat_L1000_s05$other$ind.metrics$pop <- ifelse(stratat_05$sample %in% dat_L1000_s05
$other$ind.metrics$id, stratat_05$pop, dat_L1000_s05$other$ind.metrics$pop)
dat_L1000_s05 <- gl.reassign.pop(dat_L1000_s05, as.pop='pop',verbose=3)
popNames(dat_L1000_s05)

ind.list_typ <- dat_L1000$ind.names[dat_L1000$ind.names %in% subsamp_ID_typical$I
D]
dat_L1000_styp <- gl.keep.ind(dat_L1000, ind.list=ind.list_typ, recalc = FALSE, mo
no.rm = FALSE, verbose = NULL)
dat_L1000_styp@other$ind.metrics$pop <- pop(dat_L1000_styp)
dat_L1000_styp$other$ind.metrics$pop <- ifelse(stratat_typical$sample %in% dat_L10
00_styp$other$ind.metrics$id, stratat_typical$pop, dat_L1000_styp$other$ind.metric
s$pop)
dat_L1000_styp <- gl.reassign.pop(dat_L1000_styp, as.pop='pop',verbose=3)
popNames(dat_L1000_styp)
```

Conversion et export des données génotypiques de sous-échantillons

On exporte les données génotypiques pour chacun des 12 jeux de données en différents formats:

```
# ------------------------------------------------------------------
# CONVERSION EN FORMAT GENEPOP (.GEN):
# ------------------------------------------------------------------

gl2genepop(dat_s005, outfile = paste0("gen_output_", str1, "_", str2, "_L30000_s00
5.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_s02, outfile = paste0("gen_output_", str1, "_", str2, "_L30000_s02.
gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_s05, outfile = paste0("gen_output_", str1, "_", str2, "_L30000_s05.
gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_styp, outfile = paste0("gen_output_", str1, "_", str2, "_L30000_sty
p.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L10000_s005, outfile = paste0("gen_output_", str1, "_", str2, "_L10
000_s005.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L10000_s02, outfile = paste0("gen_output_", str1, "_", str2, "_L100
00_s02.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L10000_s05, outfile = paste0("gen_output_", str1, "_", str2, "_L100
00_s05.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L10000_styp, outfile = paste0("gen_output_", str1, "_", str2, "_L10
000_styp.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L1000_s005, outfile = paste0("gen_output_", str1, "_", str2, "_L100
0_s005.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L1000_s02, outfile = paste0("gen_output_", str1, "_", str2, "_L1000
_s02.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L1000_s05, outfile = paste0("gen_output_", str1, "_", str2, "_L1000
```

```
_s05.gen"), outpath = getwd(), verbose = NULL)
gl2genepop(dat_L1000_styp, outfile = paste0("gen_output_", str1, "_", str2, "_L100
0_styp.gen"), outpath = getwd(), verbose = NULL)

# ----------------------------------------------------------------
# CONVERSION EN FORMAT PLINK (.PED et .MAP):
# ----------------------------------------------------------------

# Ici, nous allons réintégrer à nos fichiers l'information de structure chromosomi
que. C'est la subdivision en 5 chromosomes recombinants qui avait été programmée d
ans le logiciel SLiM.
gen_list <- c(dat_L1000_s005, dat_L1000_s02, dat_L1000_s05, dat_L1000_styp, dat_L1
0000_s005, dat_L10000_s02, dat_L10000_s05, dat_L10000_styp, dat_s005, dat_s02, dat
_s05, dat_styp)
names(gen_list) <- c("dat_L1000_s005", "dat_L1000_s02", "dat_L1000_s05", "dat_L100
0_styp", "dat_L10000_s005", "dat_L10000_s02", "dat_L10000_s05", "dat_L10000_styp",
"dat_s005", "dat_s02", "dat_s05", "dat_styp")

for (dat0 in names(gen_list)) {
  dat <- gen_list[[dat0]]
  dat$other$loc.metrics$position <- dat$position
  dat$other$loc.metrics$chromosome <- dat$chromosome
  dat$other$loc.metrics$chromosome <- ifelse(dat$other$loc.metrics$position %in% c
(1:400000000), 1, dat$other$loc.metrics$chromosome)
  dat$other$loc.metrics$chromosome <- ifelse(dat$other$loc.metrics$position %in% c
(400000001:800000000), 2, dat$other$loc.metrics$chromosome)
  dat$other$loc.metrics$chromosome <- ifelse(dat$other$loc.metrics$position %in% c
(800000001:1200000000), 3, dat$other$loc.metrics$chromosome)
  dat$other$loc.metrics$chromosome <- ifelse(dat$other$loc.metrics$position %in% c
(1200000001:1600000000), 4, dat$other$loc.metrics$chromosome)
  dat$other$loc.metrics$chromosome <- ifelse(dat$other$loc.metrics$position %in% c
(1600000001:2000000000), 5, dat$other$loc.metrics$chromosome)
  dat$chromosome <- as.factor(dat$other$loc.metrics$chromosome)
  assign(paste0(dat0, "_p1"), gl.keep.pop(dat, "1"))
  assign(paste0(dat0, "_p2"), gl.keep.pop(dat, "2"))
}

# Nous pouvons procéder à la conversion et à l'export de fichiers, pour chaque sou
s-échantillonnage et chaque sous-population (p1 et p2):
gl2plink(dat_s005_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = pas
te0("ped_output_", str1, "_", str2, "_L30000_s005_p1"), outpath = getwd(), chr_for
mat = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown",
phen_value = "-9", verbose = NULL)
gl2plink(dat_s02_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = past
e0("ped_output_", str1, "_", str2, "_L30000_s02_p1"), outpath = getwd(), chr_forma
t = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown", p
hen_value = "-9", verbose = NULL)
gl2plink(dat_s05_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = past
e0("ped_output_", str1, "_", str2, "_L30000_s05_p1"), outpath = getwd(), chr_forma
t = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown", p
hen_value = "-9", verbose = NULL)
gl2plink(dat_styp_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = pas
te0("ped_output_", str1, "_", str2, "_L30000_styp_p1"), outpath = getwd(), chr_for
mat = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown",
phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_s005_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfil
e = paste0("ped_output_", str1, "_", str2, "_L10000_s005_p1"), outpath = getwd(),
```

```
 chr_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "u
nknown", phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_s02_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L10000_s02_p1"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_s05_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L10000_s05_p1"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_styp_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfil
e = paste0("ped_output_", str1, "_", str2, "_L10000_styp_p1"), outpath = getwd(),
 chr_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "u
nknown", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_s005_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L1000_s005_p1"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_s02_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
 = paste0("ped_output_", str1, "_", str2, "_L1000_s02_p1"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_s05_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
 = paste0("ped_output_", str1, "_", str2, "_L1000_s05_p1"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_styp_p1, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L1000_styp_p1"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)

gl2plink(dat_s005_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = pas
te0("ped_output_", str1, "_", str2, "_L30000_s005_p2"), outpath = getwd(), chr_for
mat = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown",
phen_value = "-9", verbose = NULL)
gl2plink(dat_s02_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = past
e0("ped_output_", str1, "_", str2, "_L30000_s02_p2"), outpath = getwd(), chr_forma
t = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown", p
hen_value = "-9", verbose = NULL)
gl2plink(dat_s05_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = past
e0("ped_output_", str1, "_", str2, "_L30000_s05_p2"), outpath = getwd(), chr_forma
t = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown", p
hen_value = "-9", verbose = NULL)
gl2plink(dat_styp_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile = pas
te0("ped_output_", str1, "_", str2, "_L30000_styp_p2"), outpath = getwd(), chr_for
mat = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unknown",
phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_s005_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfil
e = paste0("ped_output_", str1, "_", str2, "_L10000_s005_p2"), outpath = getwd(),
 chr_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "u
nknown", phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_s02_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L10000_s02_p2"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_s05_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L10000_s05_p2"), outpath = getwd(), chr
```

```
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L10000_styp_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfil
e = paste0("ped_output_", str1, "_", str2, "_L10000_styp_p2"), outpath = getwd(),
 chr_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "u
nknown", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_s005_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L1000_s005_p2"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_s02_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
 = paste0("ped_output_", str1, "_", str2, "_L1000_s02_p2"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_s05_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
 = paste0("ped_output_", str1, "_", str2, "_L1000_s05_p2"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
gl2plink(dat_L1000_styp_p2, plink_path = "../PLINK_linux", bed_file=FALSE, outfile
= paste0("ped_output_", str1, "_", str2, "_L1000_styp_p2"), outpath = getwd(), chr
_format = "character", pos_cM = "0", ID_dad = "0", ID_mom = "0", sex_code = "unkno
wn", phen_value = "-9", verbose = NULL)
```

A présent, ces fichiers exportés peuvent être utilisés comme entrées pour les 3 logiciels NeEstimator, GONE et GADMA. Ces logiciels ne sont pas liés au logiciel R et le traitement des 12 jeux de données à 1000, 10000 ou 30000 et 14, 50, 55 ou 139 individus par sous-population s'effectue de manière indépendante. Ce traitement ne sera pas montré ici mais est explicité dans le rapport scientifique final du projet POPSIZE (cf. section OP.I.4).

Nous pouvons par contre poursuivre, ici, en ne nous occupant plus des informations génotypiques des individus mais uniquement à leurs informations démographiques et notamment, leurs relations d'apparentement, point central de la méthode de *close kin mark recapture* (CKMR, Bravington et al. 2016).

3. Quelques informations sur les relations d'apparentement entre individus simulés puis "échantillonnés" dans SLiM et *pyslim*/*msprime*:

Pour commencer, nous pouvons reprendre les mêmes individus que ceux qui ont été sous-échantillonnés depuis notre fichier .vcf de référence en vue du test de méthodes d'estimation de taille efficace. Il peut en effet être intéressant, in fine, de comparer les méthodes génétiques d'estimation de taille efficace et d'abondance totale en utilisant les mêmes jeux de données simulés, tout comme nous pourrions être amenés à utiliser un seul et même jeu de données empirique dans le cadre d'applications réelles.

Les fonctions de la librairie R CKMRpop, à défaut de pouvoir simuler facilement la migration et un grand nombre de loci (cf., section I.3.1 du rapport scientifique final du projet POPSIZE), restent très utiles pour visualiser les données des individus échantillonnés. Consulter la vignette CKMRpop.

Nous commençons tout d'abord par récupérer l'information relative aux parents et grand-parents de chaque individu échantillonné dans SLiM, stockée dans le fichier de sortie `SLiM_ancestries.tsv`. Nous utilisons ensuite les fonctions `CKMRpop::relatives_from_ancestry_vectors` puis `CKMRpop::compile_related_pairs` qui vont nous permettre de regrouper l'ensemble des apparentements pour chaque échantillon (cousins, frères ou soeurs, oncles ou tantes, etc.)

```
print("Visualisation de nos tableaux de données d'individus échantillonnés:")
```

```
## [1] "Visualisation de nos tableaux de données d'individus échantillonnés:"
```

```
samples
```

```
## # A tibble: 16,948 x 7
##    ID     sex   born_year born_pop samp_years_list~ sampling_pop age_at_samplin
g
##    <chr>  <chr>     <int>    <int>            <int>        <int>           <int
>
##  1 F88_2~ F            88        2               90            1
2
##  2 F89_1~ F            89        1               90            1
1
##  3 F89_1~ F            89        1               90            1
1
##  4 F88_1~ F            88        1               90            1
2
##  5 F86_2~ F            86        2               90            1
4
##  6 M87_1~ M            87        1               90            1
3
##  7 F87_1~ F            87        1               90            1
3
##  8 F87_1~ F            87        1               90            1
3
##  9 M88_1~ M            88        1               90            1
2
## 10 M81_1~ M            81        1               90            1
9
## # ... with 16,938 more rows
```

```
subsamp_ID_005
```

```
## # A tibble: 28 x 7
## # Groups:   samp_years_list_post, sampling_pop [2]
##    ID     sex   born_year born_pop samp_years_list~ sampling_pop age_at_samplin
g
##    <chr>  <chr>     <int>    <int>            <int>        <int>           <int
>
##  1 M96_1~ M            96        1              100            1
4
##  2 F96_1~ F            96        1              100            1
4
##  3 F96_1~ F            96        1              100            1
4
##  4 M99_2~ M            99        2              100            1
1
##  5 F99_1~ F            99        1              100            1
1
##  6 F94_1~ F            94        1              100            1
```

```
6
##  7 F97_1~ F            97        1              100            1
3
##  8 M98_1~ M            98        1              100            1
2
##  9 F98_1~ F            98        1              100            1
2
## 10 M98_1~ M            98        1              100            1
2
## # ... with 18 more rows
```

```
subsamp_ID_02
```

```
## # A tibble: 110 x 7
## # Groups:   samp_years_list_post, sampling_pop [2]
##    ID     sex   born_year born_pop samp_years_list~ sampling_pop age_at_samplin
g
##    <chr>  <chr>     <int>    <int>            <int>        <int>           <int
>
##  1 F93_1~ F            93        1              100            1
7
##  2 M98_1~ M            98        1              100            1
2
##  3 F95_1~ F            95        1              100            1
5
##  4 F95_1~ F            95        1              100            1
5
##  5 F99_2~ F            99        2              100            1
1
##  6 F96_1~ F            96        1              100            1
4
##  7 M98_1~ M            98        1              100            1
2
##  8 M90_1~ M            90        1              100            1              1
0
##  9 M99_1~ M            99        1              100            1
1
## 10 M98_1~ M            98        1              100            1
2
## # ... with 100 more rows
```

```
subsamp_ID_05
```

```
## # A tibble: 278 x 7
## # Groups:   samp_years_list_post, sampling_pop [2]
##    ID     sex   born_year born_pop samp_years_list~ sampling_pop age_at_samplin
g
##    <chr>  <chr>     <int>    <int>            <int>        <int>           <int
>
##  1 F98_1~ F            98        1              100            1
2
##  2 F98_1~ F            98        1              100            1
2
##  3 F96_1~ F            96        1              100            1
```

```
4
##  4 F96_1~ F            96        1              100            1
4
##  5 F93_2~ F            93        2              100            1
7
##  6 M99_1~ M            99        1              100            1
1
##  7 M97_1~ M            97        1              100            1
3
##  8 F98_1~ F            98        1              100            1
2
##  9 M99_1~ M            99        1              100            1
1
## 10 M93_1~ M            93        1              100            1
7
## # ... with 268 more rows
```

```
subsamp_ID_typical
```

```
## # A tibble: 100 x 7
## # Groups:   samp_years_list_post, sampling_pop [2]
##    ID     sex   born_year born_pop samp_years_list~ sampling_pop age_at_samplin
g
##    <chr>  <chr>     <int>    <int>            <int>        <int>           <int
>
##  1 F99_1~ F            99        1              100            1
1
##  2 M96_1~ M            96        1              100            1
4
##  3 F99_2~ F            99        2              100            1
1
##  4 F98_1~ F            98        1              100            1
2
##  5 F96_1~ F            96        1              100            1
4
##  6 F99_1~ F            99        1              100            1
1
##  7 F99_2~ F            99        2              100            1
1
##  8 M98_1~ M            98        1              100            1
2
##  9 F99_1~ F            99        1              100            1
1
## 10 M99_1~ M            99        1              100            1
1
## # ... with 90 more rows
```

```
# On importe les données concernant l'identité des parents et grands-parents de ch
aque individu échantillonné dans SLiM:
anc <- read_tsv("SLiM_ancestries.tsv", col_types = "cc") %>%
    arrange(ID) %>%
    mutate(ancestors = str_split(ancestors, ",")) %>%
    filter(!duplicated(ID)) # Du fait de l'échantillonnage avec remise effectué su
r plusieurs pas de temps successifs dans SLiM, certains individus ont pu être écha
```

```
ntillonnés plusieurs fois. Ici, nous retirons ces doublons.

# Fonction pour attribuer, à chaque échantillon, l'ensemble de ses apparentés (Eri
c Anderson)
# (https://eriqande.github.io/CKMRpop/articles/using-other-simulation-programs.htm
l#simulating-with-slim-but-recording-ancestral-lineages-internally)
relatives_from_ancestry_vectors <- function (A)
{
    tmp <- A %>% unnest(ancestors) %>% group_by(ancestors) %>%
        summarise(descendants = list(unique(ID)))
    desc <- tmp$descendants
    names(desc) <- tmp$ancestors
    A %>% mutate(relatives = map(.x = ancestors, function(x) sort(unique(unlist(de
sc[as.character(x)])))))
}
anc_rel <- relatives_from_ancestry_vectors(anc)

# full_names <- samples$ID
# names(full_names) <- str_match(full_names, "_([0-9]+)$")[,2]
# anc_rel2 <- anc_rel %>%
#     mutate(relatives = map(.x = relatives, .f = function(x) full_names[as.charac
ter(x)]), ID = as.character(ID))
# (Les commandes désactivées ci-dessus sont utilisées par E. Anderson, mais se rév
èlent inutiles dans notre cas au vu du formattage de nos données.)

# Recherche des couples d'individus apparentés au sein de l'échantillon complet (s
amples) puis un de nos sous-échantillonnages (subsamp_ID_005):
samples_kin <- samples %>%
    left_join(anc_rel, by = c("ID" = "ID"))
crel_from_pedIDs <- compile_related_pairs(samples_kin)
print("Liste des couples d'individus apparentés pour l'échantillonnage complet de
 14856 individus des pas de temps 90 à 100:")
```

```
## [1] "Liste des couples d'individus apparentés pour l'échantillonnage complet de
14856 individus des pas de temps 90 à 100:"
```

```
crel_from_pedIDs
```

```
## # A tibble: 319,959 x 25
##    id_1  id_2  conn_comp dom_relat max_hit dr_hits upper_member times_encounter
~
##    <chr> <chr>     <dbl> <chr>       <int> <list>         <int>            <int
>
##  1 F78_~ F90_~         1 GP              1 <int>              1                1
9
##  2 F78_~ F91_~         1 PO              1 <int>              1                2
2
##  3 F78_~ F91_~         1 GP              1 <int>              1                1
9
##  4 F78_~ F92_~         1 GP              1 <int>              1                1
9
##  5 F78_~ F94_~         1 GP              1 <int>              1                1
9
##  6 F78_~ F94_~         1 GP              1 <int>              1                1
```

```
9
##  7 F78_~ F96_~         1 GP              1 <int>              1               1
9
##  8 F78_~ F96_~         1 GP              1 <int>              1               1
9
##  9 F78_~ F96_~         1 GP              1 <int>              1               1
9
## 10 F78_~ F97_~         1 GP              1 <int>              1               1
9
## # ... with 319,949 more rows, and 17 more variables:
## #   primary_shared_ancestors <list>, psa_tibs <list>, sex_1 <chr>, sex_2 <chr>,
## #   born_year_1 <int>, born_year_2 <int>, samp_years_list_post_1 <int>,
## #   samp_years_list_post_2 <int>, born_pop_1 <int>, sampling_pop_1 <int>,
## #   age_at_sampling_1 <int>, ancestors_1 <list>, born_pop_2 <int>,
## #   sampling_pop_2 <int>, age_at_sampling_2 <int>, ancestors_2 <list>,
## #   anc_match_matrix <list>
```

```
crel_from_pedIDs %>% group_by(id_1, id_2) %>% mutate(dupe = n()>1) %>% filter(dupe
=="TRUE") # On vérifie le nombre de doublons.
```

```
## # A tibble: 133,718 x 26
## # Groups:   id_1, id_2 [58,899]
##    id_1  id_2  conn_comp dom_relat max_hit dr_hits upper_member times_encounter
~
##    <chr> <chr>     <dbl> <chr>       <int> <list>         <int>             <int
>
##  1 F78_~ M83_~         1 FC              1 <int>             NA
3
##  2 F78_~ M83_~         1 FC              1 <int>             NA
3
##  3 F80_~ F93_~         1 GP              1 <int>              1                2
5
##  4 F80_~ F93_~         1 GP              1 <int>              1                2
5
##  5 F80_~ F96_~         1 GP              1 <int>              1                2
5
##  6 F80_~ F96_~         1 GP              1 <int>              1                2
5
##  7 F80_~ M89_~         1 A               2 <int>              1                1
5
##  8 F80_~ M89_~         1 A               2 <int>              1                1
5
##  9 F80_~ M91_~         1 FC              1 <int>             NA                1
3
## 10 F80_~ M91_~         1 FC              1 <int>             NA                1
3
## # ... with 133,708 more rows, and 18 more variables:
## #   primary_shared_ancestors <list>, psa_tibs <list>, sex_1 <chr>, sex_2 <chr>,
## #   born_year_1 <int>, born_year_2 <int>, samp_years_list_post_1 <int>,
## #   samp_years_list_post_2 <int>, born_pop_1 <int>, sampling_pop_1 <int>,
## #   age_at_sampling_1 <int>, ancestors_1 <list>, born_pop_2 <int>,
## #   sampling_pop_2 <int>, age_at_sampling_2 <int>, ancestors_2 <list>,
## #   anc_match_matrix <list>, dupe <lgl>
```

```
subsamp_ID_05_kin <- subsamp_ID_05 %>%
    left_join(anc_rel, by = c("ID" = "ID"))
crel_from_pedIDs_05 <- compile_related_pairs(subsamp_ID_05_kin)
print("Liste des couples d'individus apparentés pour un sous-échantillonnage de 13
9 individus par sous-population au pas de temps 100:")
```

```
## [1] "Liste des couples d'individus apparentés pour un sous-échantillonnage de 1
39 individus par sous-population au pas de temps 100:"
```

```
crel_from_pedIDs
```

```
## # A tibble: 319,959 x 25
##    id_1  id_2  conn_comp dom_relat max_hit dr_hits upper_member times_encounter
~
##    <chr> <chr>     <dbl> <chr>       <int> <list>         <int>             <int
>
##  1 F78_~ F90_~         1 GP              1 <int>              1                 1
9
##  2 F78_~ F91_~         1 PO              1 <int>              1                 2
2
##  3 F78_~ F91_~         1 GP              1 <int>              1                 1
9
##  4 F78_~ F92_~         1 GP              1 <int>              1                 1
9
##  5 F78_~ F94_~         1 GP              1 <int>              1                 1
9
##  6 F78_~ F94_~         1 GP              1 <int>              1                 1
9
##  7 F78_~ F96_~         1 GP              1 <int>              1                 1
9
##  8 F78_~ F96_~         1 GP              1 <int>              1                 1
9
##  9 F78_~ F96_~         1 GP              1 <int>              1                 1
9
## 10 F78_~ F97_~         1 GP              1 <int>              1                 1
9
## # ... with 319,949 more rows, and 17 more variables:
## #   primary_shared_ancestors <list>, psa_tibs <list>, sex_1 <chr>, sex_2 <chr>,
## #   born_year_1 <int>, born_year_2 <int>, samp_years_list_post_1 <int>,
## #   samp_years_list_post_2 <int>, born_pop_1 <int>, sampling_pop_1 <int>,
## #   age_at_sampling_1 <int>, ancestors_1 <list>, born_pop_2 <int>,
## #   sampling_pop_2 <int>, age_at_sampling_2 <int>, ancestors_2 <list>,
## #   anc_match_matrix <list>
```

```
crel_from_pedIDs_05 %>% group_by(id_1, id_2) %>% mutate(dupe = n()>1) %>% filter(d
upe=="TRUE") # Zéro doublons puisque échantillonnage sur 1 seule année.
```

```
## # A tibble: 0 x 26
## # Groups:   id_1, id_2 [0]
## # ... with 26 variables: id_1 <chr>, id_2 <chr>, conn_comp <dbl>,
## #   dom_relat <chr>, max_hit <int>, dr_hits <list>, upper_member <int>,
## #   times_encountered <int>, primary_shared_ancestors <list>, psa_tibs <list>,
```

```
## #   sex_1 <chr>, sex_2 <chr>, born_year_1 <int>, born_year_2 <int>,
## #   samp_years_list_post_1 <int>, samp_years_list_post_2 <int>,
## #   born_pop_1 <int>, sampling_pop_1 <int>, age_at_sampling_1 <int>,
## #   ancestors_1 <list>, born_pop_2 <int>, sampling_pop_2 <int>, ...
```

Comme nous pouvons le voir, nous obtenons de nombreux duos d'apparentés. Cela n'est pas surprenant car d'une part, CKMRpop peut répertorier de nombreuses catégories d'apparentement différentes sur la base des données généalogiques fournies par SLiM. D'autre part, notre simulation concernait une population de petite taille (abondance totale de ~17710 individus juvéniles et adultes). Les réseaux familiaux y sont donc denses. Lorsque les simulations seront étendues à de beaucoup plus larges populations, ces chiffres devraient diminuer au fur et à mesure que la proportion d'individus échantillonnés dans SLiM diminue.

Observons un peu les différentes catégories d'apparentement. CKMRpop en distingue plusieurs:

- Se: self (même individu)
- PO: parent-offspring (parent-enfant)
- Si: sibling (correspond à une catégorie "half-sibling" HSP si max_hit=1 (partage d'un parent), "full-sibling" FSP si max_hit=2 (partage de 2 parents)
- GP: grand-parent
- A: avuncular (relation type oncle/tante-neveu/nièce)
- FC: first-cousin (cousins au premier degré).

A ce jour, les applications du CKMR s'intéressent aux relations de type HSP/FSP (Si) et POP (PO). Observons leur occurrence dans notre échantillonnage:

```
print("SOUS-ECHANTILLONNAGE DE 139 INDIVIDUS PAR SOUS-POPULATION:")
```

```
## [1] "SOUS-ECHANTILLONNAGE DE 139 INDIVIDUS PAR SOUS-POPULATION:"
```

```
nb_kin_types <- crel_from_pedIDs_05 %>% count(dom_relat, max_hit)
nb_clusters <- crel_from_pedIDs_05 %>% count(conn_comp) %>% arrange(desc(n))
print(paste("Nous avons ", nrow(nb_clusters), " groupes d'individus apparentés, do
nt ", length(which(nb_clusters$n>1)), " comprennent plus d'une paire (triades ou p
lus).", sep=""))
```

```
## [1] "Nous avons 45 groupes d'individus apparentés, dont 15 comprennent plus d'u
ne paire (triades ou plus)."
```

```
print(paste("Nous avons ", as.numeric(nb_kin_types[nb_kin_types$dom_relat=="PO",3
]), " paires POP, ",
            as.numeric(nb_kin_types[nb_kin_types$dom_relat=="Si" & nb_kin_types$ma
x_hit=="1", 3]), " HSP et ",
            as.numeric(nb_kin_types[nb_kin_types$dom_relat=="Si" & nb_kin_types$ma
x_hit=="2", 3]), " FSP, toutes classes d'âge et populations confondues.", sep=""))
```

```
## [1] "Nous avons NA paires POP, 19 HSP et 1 FSP, toutes classes d'âge et populat
ions confondues."
```

```
print("ECHANTILLONNAGE COMPLET ISSU DE SLIM (en tenant compte des éventuels doublo
ns générant plusieurs couples apparentés identiques):")
```

```
## [1] "ECHANTILLONNAGE COMPLET ISSU DE SLIM (en tenant compte des éventuels doubl
ons générant plusieurs couples apparentés identiques):"
```

```
nb_kin_types <- crel_from_pedIDs  %>% distinct(id_1, id_2, .keep_all = TRUE) %>% c
ount(dom_relat, max_hit)
nb_clusters <- crel_from_pedIDs  %>% distinct(id_1, id_2, .keep_all = TRUE) %>% co
unt(conn_comp) %>% arrange(desc(n))
nb_kin_types
```

```
## # A tibble: 10 x 3
##    dom_relat max_hit      n
##    <chr>       <int>  <int>
##  1 A               1  70460
##  2 A               2   3804
##  3 A               3      1
##  4 FC              1 118077
##  5 FC              2   6953
##  6 FC              3      4
##  7 GP              1   3404
##  8 PO              1   8816
##  9 Si              1  31775
## 10 Si              2   1846
```

```
print(paste("Nous avons ", nrow(nb_clusters), " groupes d'individus apparentés, do
nt ", length(which(nb_clusters$n>1)), " comprennent plus d'une paire (triades ou p
lus).", sep=""))
```

```
## [1] "Nous avons 1 groupes d'individus apparentés, dont 1 comprennent plus d'une
paire (triades ou plus)."
```

```
print(paste("Nous avons ", as.numeric(nb_kin_types[nb_kin_types$dom_relat=="PO",3
]), " paires POP, ",
            as.numeric(nb_kin_types[nb_kin_types$dom_relat=="Si" & nb_kin_types$ma
x_hit=="1", 3]), " HSP et ",
            as.numeric(nb_kin_types[nb_kin_types$dom_relat=="Si" & nb_kin_types$ma
x_hit=="2", 3]), " FSP, toutes classes d'âge et populations confondues.", sep=""))
```

```
## [1] "Nous avons 8816 paires POP, 31775 HSP et 1846 FSP, toutes classes d'âge et
populations confondues."
```

Avec notre sous-échantillonnage de 139 individus par sous-population issus du pas de temps 100, on constate une absence de couple POP, 19 couples d'individus HSP partageant 1 parent et 1 couple d'individus FSP partageant 2 parents. Les individus apparentés entre eux forment 45 groupes dont 15 impliquant plus de 2 individus (ce qui devrait être extrêmement rare sous hypothèse de "sparse-sampling" mais n'est pas surprenant ici, au vu de la petite taille de population simulée, et au vu du fait qu'on tient également compte d'autres catégories d'apparentement que POP, HSP ou FSP).

Avec notre échantillonnage complet de 14856 individus, nous trouvons de nombreux couples POP, HSP et FSP. Tous les échantillons appartiennent à un seul et même réseau familial car ils sont tous plus ou moins apparentés.

Nous pouvons à présent reprendre l'ensemble de notre échantillon de référence (14856 individus) et effectuer autant de sous-échantillonnages qu'on le souhaite. Par exemple, nous pouvons nous inspirer de l'Exemple 2 fourni dans le rapport final du projet POPSIZE, section I.2.2.2.

Imaginons que l'on recherche des couples d'individus apparentés de type POP dans un échantillonnage de nos populations telles que simulées dans SLiM. Nous allons supposer que cet échantillonnage provient du pas de temps 95, et que nous avons accès à des individus âgés de 1 an, et des individus âgés de 7 ans (ceci est bien entendu une simplification des conditions réelles d'échantillonnage sur le terrain.)

D'après les équations théoriques présentées en sections I.2.2.2 et OP.I.5, pour espérer observer 55 couples POP, il nous faudrait échantillonner par exemple 409 individus juvéniles d'âge 1 et 409 adultes d'âge 7. Dans notre cas cela s'avèrera impossible (cf., lignes de commandes ci-dessous) et au vu du nombre d'individus disponibles, notre espérance du nombre de couple POP sera de 9,35 (~9).

```
print("Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnage et par classe d'âge:")
```

```
## [1] "Nombre d'individus collectés dans SLiM par pas de temps d'échantillonnage et par classe d'âge:"
```

```
table(samples$age_at_sampling, samples$samp_years_list_post)
```

```
##
##       90  91  92  93  94  95  96  97  98  99 100
##   1  548 546 560 509 473 580 547 525 530 521 579
##   2  333 320 322 324 340 328 326 312 340 330 333
##   3  218 230 224 220 211 224 236 211 220 227 235
##   4  160 171 147 166 159 156 155 136 165 162 161
##   5   83 118 112  97 122 106 116 120  88  98 103
##   6   81  87  74  88  72  73  64  84  74  55  56
##   7   48  53  45  44  48  54  55  54  58  51  51
##   8   28  42  40  21  29  31  29  32  33  36  34
##   9   22  16  17  19  12  11  17  17  17  24  21
##   10   9   7   3   7   9  10  11   8   9   7   9
##   11   3   1   1   2   2   4   3   6   3   3   6
##   12   1   0   1   2   2   2   1   0   1   3   3
##   13   0   0   0   1   1   0   0   1   2   3   0
##   14   0   0   0   0   0   1   0   0   0   0   0
```

```
# On constate que nous avons 580 individus d'âge 1 et 54 individus d'âge 7 à disposition. Nous pourrions donc échantillonner 430 individus pour l'âge 1, mais pas pour l'âge 7. Par défaut, nous allons échantillonner l'ensemble des individus disponibles dans les deux classes d'âge.

subsamp_1 <- samples %>% filter(samp_years_list_post %in% c(95), age_at_sampling %in% c(1,7)) %>%
  group_by(samp_years_list_post, age_at_sampling) %>% sample_n(n())
print("Nombre d'individus sous-échantillonnés pour les classes d'âge 1 et 7 au pas de temps 95:")
```

```
## [1] "Nombre d'individus sous-échantillonnés pour les classes d'âge 1 et 7 au pas de temps 95:"
```

```
table(subsamp_1$age_at_sampling, subsamp_1$samp_years_list_post)
```

```
##
##      95
##   1 580
##   7  54
```

```
# Puis, on applique la même démarche que précédemment pour regrouper les individus apparentés:
subsamp_1_kin <- subsamp_1 %>%
    left_join(anc_rel, by = c("ID" = "ID"))
crel_from_pedIDs <- compile_related_pairs(subsamp_1_kin)
print("Liste des couples d'individus apparentés pour l'échantillonnage de 580 individus d'âge 1 et 54 individus d'âge 7 au pas de temps 95:")
```

```
## [1] "Liste des couples d'individus apparentés pour l'échantillonnage de 580 individus d'âge 1 et 54 individus d'âge 7 au pas de temps 95:"
```

```
crel_from_pedIDs
```

```
## # A tibble: 674 x 25
##    id_1  id_2  conn_comp dom_relat max_hit dr_hits upper_member times_encounter~
##    <chr> <chr>     <dbl> <chr>       <int> <list>         <int>            <int>
##  1 F88_~ F94_~         1 FC              1 <int>             NA                5
##  2 F88_~ F94_~         1 A               1 <int>              1                4
##  3 F88_~ M94_~         1 FC              1 <int>             NA                5
##  4 F88_~ M94_~         1 FC              1 <int>             NA                5
##  5 F88_~ M94_~         1 A               1 <int>              1                4
##  6 F88_~ M94_~         1 FC              1 <int>             NA                5
##  7 F88_~ M94_~         1 A               1 <int>              1                4
##  8 F88_~ F94_~         2 A               1 <int>              1                2
##  9 F88_~ F88_~         3 FC              1 <int>             NA                2
## 10 F88_~ F94_~         3 FC              1 <int>             NA                2
## # ... with 664 more rows, and 17 more variables:
## #   primary_shared_ancestors <list>, psa_tibs <list>, sex_1 <chr>, sex_2 <chr>,
## #   born_year_1 <int>, born_year_2 <int>, samp_years_list_post_1 <int>,
## #   samp_years_list_post_2 <int>, born_pop_1 <int>, sampling_pop_1 <int>,
## #   age_at_sampling_1 <int>, ancestors_1 <list>, born_pop_2 <int>,
## #   sampling_pop_2 <int>, age_at_sampling_2 <int>, ancestors_2 <list>,
## #   anc_match_matrix <list>
```

```
nb_kin_types <- crel_from_pedIDs  %>% distinct(id_1, id_2, .keep_all = TRUE) %>% c
ount(dom_relat, max_hit)
nb_clusters <- crel_from_pedIDs  %>% distinct(id_1, id_2, .keep_all = TRUE) %>% co
unt(conn_comp) %>% arrange(desc(n))
print(paste("Nous avons ", nrow(nb_clusters), " groupes d'individus apparentés, do
nt ", length(which(nb_clusters$n>1)), " comprennent plus d'une paire (triades ou p
lus).", sep=""))
```

```
## [1] "Nous avons 86 groupes d'individus apparentés, dont 44 comprennent plus d'u
ne paire (triades ou plus)."
```

```
print(paste("Nous avons ", as.numeric(nb_kin_types[nb_kin_types$dom_relat=="PO",3
]), " apparentements de type POP dans notre sous-échantillon, toutes sous-populati
ons confondues.", sep=""))
```

```
## [1] "Nous avons 12 apparentements de type POP dans notre sous-échantillon, tout
es sous-populations confondues."
```

Les équations théoriques prévoyaient un nombre de paires apparentées POP de 9,35. Nous en observons 12. Si l'on réapplique cette opération sur chacun des pas de temps séparément (90 à 100), en intégrant à chaque fois l'ensemble des individus disponibles dans les classes d'âge 1 et 7, nous obtenons un nombre de couples POP de 9, 9, 5, 8, 10, 12, 9, 6, 13, 13 et 12, respectivement. Ceci correspond à une moyenne de 9,64 et une variance de 7,25 (attention cependant, puisque ces valeurs ne se basent que sur 11 observations, à ce stade.)

Ce petit exemple simple vise à fournir une illustration de la manière dont la librairie CKMRpop peut nous permettre d'analyser des données issues de notre procédure de simulation. A partir d'un jeu de données simulés, on peut ainsi faire varier les modalités de sous-échantillonnage (en les faisant varier par sous-population, par classes d'âges, etc.) afin d'observer leur influence sur le nombre de couples apparentés que l'on y retrouve.

A l'avenir, il sera utile d'appliquer ces démarches sur des jeux de données simulées de plus grande envergure.

Fin de document R Markdown.

Annexe: Configuration R

```
options(width = 100)
devtools::session_info()
```

```
## - Session info ---------------------------------------------------------------
------------------
##  setting  value
##  version  R version 4.0.4 (2021-02-15)
##  os       Windows 10 x64 (build 19045)
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  French_France.1252
##  ctype    French_France.1252
##  tz       Europe/Paris
```

```
##  date     2023-07-27
##  pandoc   2.7.2 @ C:/Program Files/RStudio/bin/pandoc/ (via rmarkdown)
##
## - Packages ------------------------------------------------------------------
------------------
##  package      * version   date (UTC) lib source
##  ade4         * 1.7-19    2022-04-19 [1] CRAN (R 4.0.5)
##  adegenet     * 2.1.3     2020-05-10 [1] CRAN (R 4.0.4)
##  ape            5.6-2     2022-03-02 [1] CRAN (R 4.0.5)
##  assertthat     0.2.1     2019-03-21 [1] CRAN (R 4.0.4)
##  backports      1.4.1     2021-12-13 [1] CRAN (R 4.0.5)
##  bit            4.0.4     2020-08-04 [1] CRAN (R 4.0.4)
##  bit64          4.0.5     2020-08-30 [1] CRAN (R 4.0.4)
##  boot           1.3-28    2021-05-03 [1] CRAN (R 4.0.5)
##  broom          0.8.0     2022-04-13 [1] CRAN (R 4.0.5)
##  bslib          0.3.1     2021-10-06 [1] CRAN (R 4.0.5)
##  cachem         1.0.6     2021-08-19 [1] CRAN (R 4.0.5)
##  calibrate      1.7.7     2020-06-19 [1] CRAN (R 4.0.4)
##  callr          3.7.0     2021-04-20 [1] CRAN (R 4.0.5)
##  cellranger     1.1.0     2016-07-27 [1] CRAN (R 4.0.4)
##  CKMRpop      * 0.1.3     2021-07-17 [1] CRAN (R 4.0.5)
##  class          7.3-18    2021-01-24 [1] CRAN (R 4.0.4)
##  classInt       0.4-3     2020-04-07 [1] CRAN (R 4.0.4)
##  cli            3.3.0     2022-04-25 [1] CRAN (R 4.0.4)
##  cluster        2.1.1     2021-02-14 [1] CRAN (R 4.0.4)
##  coda           0.19-4    2020-09-30 [1] CRAN (R 4.0.4)
##  codetools      0.2-18    2020-11-04 [2] CRAN (R 4.0.4)
##  colorspace     2.0-3     2022-02-21 [1] CRAN (R 4.0.5)
##  combinat       0.0-8     2012-10-29 [1] CRAN (R 4.0.3)
##  crayon         1.5.1     2022-03-26 [1] CRAN (R 4.0.5)
##  dartR        * 2.0.4     2022-06-05 [1] CRAN (R 4.0.4)
##  data.table     1.14.2    2021-09-27 [1] CRAN (R 4.0.5)
##  DBI            1.1.2     2021-12-20 [1] CRAN (R 4.0.5)
##  dbplyr         2.1.1     2021-04-06 [1] CRAN (R 4.0.5)
##  deldir         0.2-10    2021-02-16 [1] CRAN (R 4.0.4)
##  desc           1.4.1     2022-03-06 [1] CRAN (R 4.0.5)
##  devtools       2.3.2     2020-09-18 [1] CRAN (R 4.0.4)
##  digest         0.6.29    2021-12-01 [1] CRAN (R 4.0.5)
##  dismo          1.3-5     2021-10-11 [1] CRAN (R 4.0.5)
##  doParallel     1.0.17    2022-02-07 [1] CRAN (R 4.0.5)
##  dotCall64      1.0-1     2021-02-11 [1] CRAN (R 4.0.5)
##  dplyr        * 1.0.9     2022-04-28 [1] CRAN (R 4.0.4)
##  e1071          1.7-4     2020-10-14 [1] CRAN (R 4.0.4)
##  ellipsis       0.3.2     2021-04-29 [1] CRAN (R 4.0.5)
##  evaluate       0.15      2022-02-18 [1] CRAN (R 4.0.5)
##  expm           0.999-6   2021-01-13 [1] CRAN (R 4.0.4)
##  fansi          1.0.3     2022-03-24 [1] CRAN (R 4.0.5)
##  fastmap        1.1.0     2021-01-25 [1] CRAN (R 4.0.4)
##  fields         13.3      2021-10-30 [1] CRAN (R 4.0.5)
##  forcats      * 0.5.1     2021-01-27 [1] CRAN (R 4.0.4)
##  foreach        1.5.2     2022-02-02 [1] CRAN (R 4.0.5)
##  fs             1.5.2     2021-12-08 [1] CRAN (R 4.0.5)
##  gap            1.2.3-6   2022-05-13 [1] CRAN (R 4.0.4)
##  gap.datasets   0.0.5     2022-05-09 [1] CRAN (R 4.0.4)
##  gdata          2.18.0.1  2022-05-10 [1] CRAN (R 4.0.4)
##  gdistance      1.3-6     2020-06-29 [1] CRAN (R 4.0.4)
##  gdsfmt         1.26.1    2020-12-22 [1] Bioconductor
```

```
##  generics       0.1.2     2022-01-31 [1] CRAN (R 4.0.5)
##  genetics       1.3.8.1.3 2021-03-01 [1] CRAN (R 4.0.4)
##  GGally         2.1.2     2021-06-21 [1] CRAN (R 4.0.5)
##  ggplot2      * 3.3.6     2022-05-03 [1] CRAN (R 4.0.4)
##  glue           1.6.2     2022-02-24 [1] CRAN (R 4.0.5)
##  gmodels        2.18.1.1  2022-05-17 [1] CRAN (R 4.0.4)
##  gridExtra      2.3       2017-09-09 [1] CRAN (R 4.0.4)
##  gtable         0.3.0     2019-03-25 [1] CRAN (R 4.0.4)
##  gtools         3.8.2     2020-03-31 [1] CRAN (R 4.0.3)
##  haven          2.5.0     2022-04-15 [1] CRAN (R 4.0.5)
##  hms            1.1.1     2021-09-26 [1] CRAN (R 4.0.5)
##  htmltools      0.5.2     2021-08-25 [1] CRAN (R 4.0.5)
##  httpuv         1.6.5     2022-01-05 [1] CRAN (R 4.0.5)
##  httr           1.4.3     2022-05-04 [1] CRAN (R 4.0.4)
##  igraph         1.2.11    2022-01-04 [1] CRAN (R 4.0.5)
##  iterators      1.0.14    2022-02-05 [1] CRAN (R 4.0.5)
##  jquerylib      0.1.4     2021-04-26 [1] CRAN (R 4.0.5)
##  jsonlite       1.8.0     2022-02-22 [1] CRAN (R 4.0.5)
##  KernSmooth     2.23-18   2020-10-29 [1] CRAN (R 4.0.4)
##  knitr          1.39      2022-04-26 [1] CRAN (R 4.0.4)
##  later          1.3.0     2021-08-18 [1] CRAN (R 4.0.5)
##  lattice        0.20-41   2020-04-02 [1] CRAN (R 4.0.4)
##  LearnBayes     2.15.1    2018-03-18 [1] CRAN (R 4.0.3)
##  lifecycle      1.0.1     2021-09-24 [1] CRAN (R 4.0.5)
##  lubridate      1.8.0     2021-10-07 [1] CRAN (R 4.0.5)
##  magrittr       2.0.3     2022-03-30 [1] CRAN (R 4.0.5)
##  maps           3.4.0     2021-09-25 [1] CRAN (R 4.0.5)
##  MASS           7.3-53.1  2021-02-12 [1] CRAN (R 4.0.4)
##  Matrix         1.2-18    2019-11-27 [1] CRAN (R 4.0.3)
##  memoise        2.0.1     2021-11-26 [1] CRAN (R 4.0.5)
##  mgcv           1.8-34    2021-02-16 [1] CRAN (R 4.0.4)
##  mime           0.12      2021-09-28 [1] CRAN (R 4.0.5)
##  mmod           1.3.3     2017-04-06 [1] CRAN (R 4.0.4)
##  mnormt         2.0.2     2020-09-01 [1] CRAN (R 4.0.3)
##  modelr         0.1.8     2020-05-19 [1] CRAN (R 4.0.4)
##  munsell        0.5.0     2018-06-12 [1] CRAN (R 4.0.4)
##  mvtnorm        1.1-3     2021-10-08 [1] CRAN (R 4.0.5)
##  nlme           3.1-152   2021-02-04 [1] CRAN (R 4.0.4)
##  patchwork      1.1.1     2020-12-17 [1] CRAN (R 4.0.5)
##  pegas          1.1       2021-12-16 [1] CRAN (R 4.0.5)
##  permute        0.9-7     2022-01-27 [1] CRAN (R 4.0.5)
##  pillar         1.7.0     2022-02-01 [1] CRAN (R 4.0.5)
##  pkgbuild       1.3.1     2021-12-20 [1] CRAN (R 4.0.5)
##  pkgconfig      2.0.3     2019-09-22 [1] CRAN (R 4.0.4)
##  pkgload        1.2.4     2021-11-30 [1] CRAN (R 4.0.5)
##  plyr           1.8.7     2022-03-24 [1] CRAN (R 4.0.5)
##  png            0.1-7     2013-12-03 [1] CRAN (R 4.0.3)
##  PopGenReport   3.0.7     2022-05-27 [1] CRAN (R 4.0.4)
##  prettyunits    1.1.1     2020-01-24 [1] CRAN (R 4.0.4)
##  processx       3.5.0     2021-03-23 [1] CRAN (R 4.0.4)
##  promises       1.2.0.1   2021-02-11 [1] CRAN (R 4.0.4)
##  ps             1.6.0     2021-02-28 [1] CRAN (R 4.0.4)
##  psych        * 2.2.5     2022-05-10 [1] CRAN (R 4.0.4)
##  purrr        * 0.3.4     2020-04-17 [1] CRAN (R 4.0.4)
##  R.methodsS3    1.8.1     2020-08-26 [1] CRAN (R 4.0.3)
##  R.oo           1.24.0    2020-08-26 [1] CRAN (R 4.0.3)
##  R.utils        2.11.0    2021-09-26 [1] CRAN (R 4.0.5)
```

```
##  R6              2.5.1     2021-08-19 [1] CRAN (R 4.0.5)
##  raster          3.5-15    2022-01-22 [1] CRAN (R 4.0.5)
##  RColorBrewer    1.1-3     2022-04-03 [1] CRAN (R 4.0.5)
##  Rcpp            1.0.8.3   2022-03-17 [1] CRAN (R 4.0.5)
##  readr         * 2.1.2     2022-01-30 [1] CRAN (R 4.0.5)
##  readxl          1.4.0     2022-03-28 [1] CRAN (R 4.0.5)
##  remotes         2.4.2     2021-11-30 [1] CRAN (R 4.0.5)
##  reprex          2.0.1     2021-08-05 [1] CRAN (R 4.0.5)
##  reshape         0.8.9     2022-04-12 [1] CRAN (R 4.0.5)
##  reshape2        1.4.4     2020-04-09 [1] CRAN (R 4.0.4)
##  rgdal           1.5-32    2022-05-09 [1] CRAN (R 4.0.4)
##  RgoogleMaps     1.4.5.3   2020-02-12 [1] CRAN (R 4.0.4)
##  rlang           1.0.2     2022-03-04 [1] CRAN (R 4.0.5)
##  rmarkdown       2.14      2022-04-25 [1] CRAN (R 4.0.4)
##  rprojroot       2.0.3     2022-04-02 [1] CRAN (R 4.0.5)
##  rstudioapi      0.13      2020-11-12 [1] CRAN (R 4.0.4)
##  rvest           1.0.2     2021-10-16 [1] CRAN (R 4.0.5)
##  sass            0.4.1     2022-03-23 [1] CRAN (R 4.0.5)
##  scales          1.2.0     2022-04-13 [1] CRAN (R 4.0.5)
##  seqinr          4.2-16    2022-05-19 [1] CRAN (R 4.0.4)
##  sessioninfo     1.2.2     2021-12-06 [1] CRAN (R 4.0.5)
##  sf              0.9-7     2021-01-06 [1] CRAN (R 4.0.4)
##  shiny           1.7.1     2021-10-02 [1] CRAN (R 4.0.5)
##  SNPRelate       1.24.0    2020-10-28 [1] Bioconductor
##  sp              1.4-7     2022-04-20 [1] CRAN (R 4.0.5)
##  spam            2.8-0     2022-01-06 [1] CRAN (R 4.0.5)
##  spData          2.0.1     2021-10-14 [1] CRAN (R 4.0.5)
##  spdep           1.1-5     2020-06-29 [1] CRAN (R 4.0.4)
##  StAMPP          1.6.3     2021-08-08 [1] CRAN (R 4.0.5)
##  stringi         1.7.6     2021-11-29 [1] CRAN (R 4.0.5)
##  stringr       * 1.4.0     2019-02-10 [1] CRAN (R 4.0.4)
##  terra           1.5-21    2022-02-17 [1] CRAN (R 4.0.5)
##  testthat        3.0.2     2021-02-14 [1] CRAN (R 4.0.4)
##  tibble        * 3.1.7     2022-05-03 [1] CRAN (R 4.0.4)
##  tidyr         * 1.2.0     2022-02-01 [1] CRAN (R 4.0.5)
##  tidyselect      1.1.2     2022-02-21 [1] CRAN (R 4.0.5)
##  tidyverse     * 1.3.1     2021-04-15 [1] CRAN (R 4.0.5)
##  tmvnsim         1.0-2     2016-12-15 [1] CRAN (R 4.0.3)
##  tzdb            0.3.0     2022-03-28 [1] CRAN (R 4.0.5)
##  units           0.7-0     2021-02-25 [1] CRAN (R 4.0.4)
##  usethis         2.0.1     2021-02-10 [1] CRAN (R 4.0.4)
##  utf8            1.2.2     2021-07-24 [1] CRAN (R 4.0.5)
##  vctrs           0.4.1     2022-04-13 [1] CRAN (R 4.0.5)
##  vegan           2.6-2     2022-04-17 [1] CRAN (R 4.0.5)
##  viridis         0.6.2     2021-10-13 [1] CRAN (R 4.0.5)
##  viridisLite     0.4.0     2021-04-13 [1] CRAN (R 4.0.5)
##  vroom           1.5.7     2021-11-30 [1] CRAN (R 4.0.5)
##  withr           2.5.0     2022-03-03 [1] CRAN (R 4.0.5)
##  xfun            0.30      2022-03-02 [1] CRAN (R 4.0.5)
##  xml2            1.3.3     2021-11-30 [1] CRAN (R 4.0.5)
##  xtable          1.8-4     2019-04-21 [1] CRAN (R 4.0.4)
##  yaml            2.3.5     2022-02-21 [1] CRAN (R 4.0.5)
##
##  [1] C:/Users/Sidon/OneDrive/Documents/R/win-library/4.0
##  [2] C:/Program Files/R/R-4.0.4/library
##
```

```
## ------------------------------------------------------------------------
--------------------
```