Simulation Modeling of Mating Scenarios for the Estimation of False Positive (FPR) and False Negative (FNR) Error Rates

Contents

a) Objectives

The main paper uses Colony 2 software to infer paternity by sampled or unsampled pollen donors and sibship relationships among offspring families from the genotypes of seeds and their known maternal plants. The reliability of such inference likely depends on the genetic structure of the sampled population, including the overall frequency of alleles at different loci, locus-specific rates of genotyping errors incl. null alleles, heterozygosity, and the relatedness among mates (biparental inbreeding). The ability to infer paternal halfsibs among the offspring of near-neighbour maternal plants (i.e., seeds of neighbouring moms sired by the same pollen donor) will also depend on the relatedness of the maternal plants.

The goal of this simulation study is to estimate error rates for different mating system parameters that take into account the observed genetic structure of *Heliconia tortuosa* in the study area. Specifically, we aim to assess false negative rates (FNR) and false positive rates (FPR) to correct estimates for the mating system parameters listed below that are based on the paternity and sibship analysis of genotyped seeds and their known maternal plants with the software Colony 2:

1. Selfing rate.
2. Rate of near-neighbour mating.
3. Rate of full siblings within maternal families.
4. Rate of paternal half-siblings among near-neighbour maternal families.

b) Design of simulation experiment

Colony 2 combines all input data and parameter settings for an analysis in a single input file in text format. To keep everything as comparable as possible to the analysis of the real genotypes, we modified the input file by adding a set of simulated seeds to the real genotypes of seeds and their maternal plants as described below.

The original data contain 814 seeds from 71 maternal plants in 18 forest patches, genotyped at 11 microsatellite loci. Within each patch, all maternal plants sampled in the same year were near-neighbours (the closest five flowering individuals from a randomly selected location ("site") within the patch).

For the simulation study, we formed pairs of maternal plants (moms) within patch. If there were 2 - 3 moms from the same patch, we sampled two of them to form one pair. If there were 4 - 5, we sampled four of them to form two pairs or near-neighbour moms.

We ran the overall simulation a single time due to computational constraints. The simulation contains a series of experiments (mating scenarios) with replicates at the levels of seeds and pairs of maternal plants nested within patches. Specifically, the simulated seeds were generated according to the mating scenarios listed below, with two replicate offspring per scenario and mom:

- M1: Selfing: each mom mates with itself.
- M2: Local pollination: each mom mates with a near-neighbour mom (paired within the same patch and year).
- M3: Immigration: each pair of moms mate with the same dad from patch 205.
- M4: Site fidelity: each pair of moms mate with the near-neighbour of M3.
- M5: Immigration: each pair of moms mates with the same dad from patch 30.
- M6: Immigration: each pair of moms mates with the same dad from patch 60.

For scenarios M1 and M2, within each pair of near-neighbour moms, we mated each genotype with itself (M1) and with its neighbour (M2). Long-distance mates for scenarios M3 - M6 were randomly selected from three small forest patches (patches 205, 30 and 60) where all flowering individuals had been genotyped. Recurring genotypes (i.e., clones) were removed prior to sampling.

For scenarios M3 and M4, we selected for each pair of moms a corresponding pair of nearest-neighbour genotypes from patch 205 based on available GPS locations. Of the 253 genotypes from patch 205, we excluded 187 due to missing GPS coordinates caused by an instrument failure in the field, and 13 due to recurring genotypes (clones), resulting in 53 unique, geo-referenced genotypes. Randomly selecting one genotype for scenario M3, and its nearest neighbour for scenario M4, resulted in pairs of pollen donors separated by 0 - 8 m (except for one pair separated by 15 m). This sampling ensured that M3 and M4 seeds are sired by genotypes that are likely related, as indicated by a kinship coefficient of r = 0.055 estimated for pairs of adult H. tortuosa plants within 10 m distance across the study area.

For scenarios M5 and M6, we selected for each pair of moms one genotype from patch 30 and one from patch 60. These patches had 54 and 61 unique genotypes, respectively. Comparisons between offspring of the same maternal plant resulting from M3, M5 and M6 scenarios thus involve maternal half-siblings sired by independent long-distance mates from three different forest patches.

For any pair of mates according to the above mating scenarios, we simulated offspring by randomly selecting one allele per parent and locus. For each simulated offspring, we recorded two genotypes, one without genotyping error and one with locus-specific error rates (estimated from parent-offspring arrays) between 0.001 - 0.08. Genotyping error was simulated by adding a random number (1:100) to the allele size, with a probability corresponding to the locus-specific error rate.

The replicates indicated above for each mating scenario are replicate offspring from the same mom and dad (i.e., full siblings). Overall, this study design resulted in 1536 simulated seed genotypes: 6 scenarios x 2 replicates x 2 genotyping error settings x 32 pairs x 2 moms per pair. We combined the real (814) and simulated (1536) seed genotypes with the 71 maternal genotypes in the modified input file for Colony 2. The paternal genotypes from patches 205, 30 and 60 used in scenarios M3 - M6 were not included in this file, i.e., they represent unsampled pollen donors.

We analysed this combined dataset in Colony 2 with the same parameter settings as for the analysis of the real genotypes presented in the main paper:

'Heliconia' Sim_1004 2350 ! Number of offspring in the sample 11 ! Number of loci 1234 ! Seed for random number generator 0 ! 0/1=Not updating/updating allele frequency 1 ! 2/1=Dioecious/Monoecious species 1 ! 0/1=Inbreeding absent/present 0 ! 0/1=Diploid species/HaploDiploid species 0 0 ! 0/1=Polygamy/Monogamy for males & females 0 ! 0/1 = Clone inference = No/Yes 1 ! 0/1=Scale full sibship=No/Yes 0 ! 0/1/2/3/4=No/Weak/Medium/Strong sibship prior; 4=Optimal sibship prior for Ne 0 ! 0/1=Unknown/Known population allele frequency 5 ! Number of runs 3 ! 1/2/3/4 = Short/Medium/Long/VeryLong run 1 ! 0/1=Monitor method by Iterate#/Time in second 1 ! Monitor interval in Iterate# / in seconds 0 ! 0/1=DOS/Windows version 2 ! 0/1/2=Pair-Likelihood-Score(PLS)/Full-Likelihood(FL)/FL-PLS-combined(FPLS) method 2 ! 0/1/2/3=Low/Medium/High/VeryHigh precision

c) Estimation of classification error rates

We used a combination of Colony 2 output files two assemble two datasets, one where each row represents a seed and one where each row represents a pair of seeds (dyad) from the same patch. The seed dataset ("Data") was based on the "BestConfig_ordered" output file, from which we retained only the n = 1656 simulated seeds. This dataset contained identifiers for the offspring (i.e., seed), the inferred dad and the known mom. From the offspring ID, we extracted the patch of the mom, the patch and ID of the true dad, the scenario type (M1 - M6), and the replicate ("a" or "b"). For each seed, we added the inferred probability that it is selfed (from output file "Selfer") and the probability of the inferred dad (from output file "ParentPair").

To create the dyad dataset ("Pairs"), we split the seed dataset by site (i.e., combination of patch and year) and extracted all unique pairs of seeds within sites. From the output files "HalfSibDyad" and "FullSibDyad" we extracted all pairs with a probability > 0.5 and combined them to a single list. Due to this cut-off value, each pair was only listed once in the combined list. We then joined the list of unique pairs of seeds (derived from the seeds dataset) with the combined list of inferred full- and halfsib dyads, so that for each simulated pair of seeds within the same patch, we obtained the inferred probability of full siblings. The inferred sibship type was classified as follows:

- Full: inferred probability of full siblings > 0.5.
- Half: same mother and inferred probability of full siblings < 0.5.
- Paternal: different mother and same inferred dad.
- Other: different mother and different inferred dad.

In all analyses below, we used cut-off values of 0.9 for the inferred probability of selfing to classify selfed seeds and for the inferred probability of full siblings to identify full sib pairs within maternal families.

Selfing:

To assess the FNR of selfing, we determined the rate of seeds inferred as outcrossed (i.e., with an inferred dad different from its mom) among all M1 seeds (which are known to be selfed). We evaluated the FPR separately for independent pollen donors and for near-neighbour mates. In each case, we determined the rate of seeds inferred as selfed among the respective set of seeds known to be outcrossed (independent: M3, M5 and M6; near-neighbour: M2).

Near-neighbour mating:

We aimed to assess whether or not a seed is correctly classified as pollinated locally, i.e., by one of the sampled near-neighbour moms. M2 seeds are sired by the paired near-neighbour maternal plant. In addition to the true dad, we considered cases where another sampled, near-neighbour maternal plant was inferred as the dad as true positives, as the inference of local pollination is correct in this case, even if an incorrect dad was inferred. (Note: this case did not actually occur in the simulations, hence this decision did not affect the results).

To assess the FNR of near-neighbour mating, we determined the rate of seeds with an inferred dad that was not a sampled near-neighbour of their mom among all M2 seeds (see above). To assess the FPR, we determined the rate of seeds with an inferred father among the sampled near-neighbour maternal plants (excluding the mother plant) among the M3, M5 and M6 offspring.

Full siblings:

To assess the FNR of full sibship, we determined the rate of pairs of seeds from the same mom that were classified as half siblings but in truth were full siblings. This analysis included M3, M5 and M6 seeds.

We evaluated the FPR separately for independent dads and for pairs of nearest-neighbour dads that are likely to be related. In each case, we determined the rate of pairs of seeds from the same mom that were classified as full siblings but in truth had different dads. The analysis with independent dads included all pairs of seeds with true dads from different patches (combinations of M3, M5 and M6 seeds), whereas the analysis with related dads included all pairs of maternal siblings where one seed was M3 and the other M4.

Paternal half siblings:

To assess the FNR of paternal half siblings, we assessed the rate of inferred half siblings among pairs of M5, or M6, seeds from paired moms. We evaluated the FPR separately for independent dads and for pairs of nearest-neighbour dads that are likely to be related. In each case, we determined the rate of pairs of seeds from the paired moms with different inferred dads but which in truth had the same dad. The analysis with independent dads included M5 and M6 seeds, whereas the analysis with related dads included M3 and M4 seeds.

1. Create Colony input file

```r
#library(here)
library(dplyr)
#library(readxl)
```

a) Import existing input file

Import first part of file that is the same for all runs.

```r
ColonyInputFile <- paste0(here::here(), "/Data/colony2.mac_.20180730/Colony2_Heliconia.dat")
Header <- readLines(ColonyInputFile)[1:26]
```

Import seed genotypes.

```r
nSeeds <- 1584

Genotypes_Seeds <- readLines(ColonyInputFile)[27 + c(1:nSeeds)]
```

Drop 2013 seeds.

```r
S_ID <- sapply(as.list(Genotypes_Seeds), function(x) strsplit(x, split=" ")[[1]][1])
S_Year <- sapply(S_ID, function(x) strsplit(x, split="_")[[1]][3])

Genotypes_Seeds <- Genotypes_Seeds[S_Year == "2016"]
length(Genotypes_Seeds)
```

Population ID for seeds.

```r
Pop <- factor(substr(Genotypes_Seeds, 1, 2))
```

Import mom genotypes.

```r
nMoms <- 158

Genotypes_Mom <- readLines(ColonyInputFile)[31 + nSeeds + c(1:nMoms)]
M_Year <- sapply(as.list(Genotypes_Mom), function(x) substr(x,7,10))
#Genotypes_Mom <- gsub("_0_20", "_", Genotypes_Mom)
```

Drop 2013 moms.

```
Genotypes_Mom <- Genotypes_Mom[M_Year == "2016"]
length(Genotypes_Mom)
```

Mom and Population IDs.

```
Genotypes_Mom <- gsub("_0_2016", "", Genotypes_Mom)
#M_ID <- as.vector(sapply(as.list(Genotypes_Mom), function(x) substr(x,1,6)))
M_ID <- as.vector(sapply(as.list(Genotypes_Mom), function(x) substr(x,1,3)))

PopMom <- factor(substr(M_ID, 1, 2))
length(table(PopMom))
```

b) Import additional dads

Import potential dads (from 3 experimental patches: 205, 03, 60) from external file.

```
Dads <- read.csv("Data_Genotypes_Experiment_Potential_Fathers.csv")
Dads <- data.frame(sapply(Dads, as.character), stringsAsFactors = FALSE)
Dads$ID <- gsub("-", "_", Dads$ID )  # Fix inconsistency in names
```

Simplify some names.

```
#table(nchar(Dads$ID))
m <- which(nchar(Dads$ID) > 7)
Dads$ID[m] <- sapply(strsplit(Dads$ID[m], split="_"), function(x) paste(x[1], x[2], sep="_"))
Dads$ID[m] <- paste0(Dads$ID[m], c("a", "b"))
```

For patch 205, exclude potential dads with missing coordinates, so that we can identify nearest neighbours.

```
Dads <- Dads %>% filter(Patch!=205 | !is.na(Dads$X))
```

Drop clones:

- From each set of clones, retain those with coordinates and select the first one.
- Assemble a lookup table that says which clones will be removed and which clone they will replace them.
- Remove the extra clones.
- Count the number of genotypes retained per experimental patch.

```
Clones_Experiment <- list("30"=readxl::read_excel("Clones_Experiment.xlsx", sheet = "Patch30"),
                          "60"=readxl::read_excel("Clones_Experiment.xlsx", sheet = "Patch60"),
                          "205"=readxl::read_excel("Clones_Experiment.xlsx", sheet = "Patch205"))

# Drop clones that were already dropped from Dads:
Clones_Experiment[[3]][,-1] <- data.frame(sapply(Clones_Experiment[[3]][,-1],
                                    function(x) ifelse(is.element(x, Dads$ID), x, NA)))

clones.rm <- clones.replace <-c()
for(k in 1:length(Clones_Experiment))
{
  for(i in 1:nrow(Clones_Experiment[[k]]))
  {
    clones <- unlist(Clones_Experiment[[k]][i,-1])
    clones <- which(is.element(Dads$ID, clones[!is.na(clones)]))
    if(length(clones) > 1)
    {
      clones.rm <- c(clones.rm, clones[-1])
      clones.replace <- c(clones.replace, rep(clones[1], length(clones) - 1))
    }
  }
}

Clone_lookup <- data.frame(Remove=Dads$ID[clones.rm], Replace=Dads$ID[clones.replace],
                           stringsAsFactors = FALSE)

m <- which(is.element(Dads$ID, Clone_lookup$Remove))
Dads <- Dads[-m,]
table(Dads$Patch)
```

c) Prepare genotypes for sampling

This creates an [11 x 2] matrix of alleles for each genotype. Optional code to verify that there are no identical genotypes anymore.

```
Alleles_P <- Reduce(cbind,lapply(Dads[,-c(1:4)],
                    function(x) matrix(as.numeric(unlist(strsplit(as.character(x),
                               split="[:]"))), ncol=2, byrow=TRUE)))
# Check that there are no clones:
#tmp <- matrix(NA, nrow(Alleles_P), nrow(Alleles_P))
#for(i in 1:(nrow(Alleles_P) - 1))
#{
#  for(j in i:nrow(Alleles_P))
#  {
#    tmp[i,j] <- sum(Alleles_P[i,] != Alleles_P[j,])
#  }
#}
#table(tmp[upper.tri(tmp)])  # Distribution of number of different alleles: no clones

Alleles_P <- lapply(data.frame(t(Alleles_P)), function(x) matrix(as.numeric(x), ncol=2, byrow=TRUE))
names(Alleles_P) <- Dads$ID
Alleles_P <- split(Alleles_P, Dads$Patch)
```

Prepare maternal genotypes for sampling.

```
Alleles_M <- sapply(Genotypes_Mom, function(ls) strsplit(ls, " "))
names(Alleles_M) <- M_ID
Alleles_M <- lapply(Alleles_M, function(x) matrix(as.numeric(x[-1]), ncol=2, byrow=TRUE))
```

d) Assign mates

Study design (replicates are replicate offspring from the same mom and dad):

- M1: Selfing: each mom mates with itself (1 replicate).
- M2: Local pollination: each mom mates with a neighboring mom (same patch and year) (1 replicate).
- M3: Immigration: each pair of moms mate with the same dad from patch 205 (2 replicates).
- M4: Site fidelity: each pair of moms mate with the nearest neighbor of M3 (2 replicates).
- M5: Immigration: each pair of moms mates with the same dad from patch 30 (3 replicates).
- M6: Immigration: each pair of moms mates with the same dad from patch 60 (3 replicates).

Select pairs of moms from the same patch and year:

- If there are 2 - 3 moms from the same patch and year, sample 2 (1 pair of moms).
- If there are 4 - 5 moms from the same patch and year, sample 4 (2 pairs of moms).

```
#Sample_M <- split(Alleles_M, list(PopMom, M_Year))
Sample_M <- split(Alleles_M, PopMom)
#table(sapply(Sample_M, length))

a <- which(sapply(Sample_M, length) == 3)
# Sample two moms per patch/year if 3 moms
Sample_M[a] <- lapply(Sample_M[a], function(ls) ls[sample(length(ls), 2)])
a <- which(sapply(Sample_M, length) > 4)
Sample_M[a] <- lapply(Sample_M[a], function(ls) ls[sample(length(ls), 4)])
a <- which(sapply(Sample_M, length) == 0)
if(length(a) > 0) {Sample_M <- Sample_M[-a]}
#table(sapply(Sample_M, length))
Sample_M <- unlist(Sample_M, recursive=FALSE)
#length(Sample_M)

names(Sample_M) <- sapply(names(Sample_M), function(x) substr(x, 4, 6))
```

Create mating table to define mates:

- M1 and M2: Mate pairs of moms from same patch and year.

```
# Create mating table

nMates = 6
Mates <- data.frame(matrix(NA, length(Sample_M), nMates))
names(Mates) <- c("Selfed", "SamePatchM", "P205_1", "P205_2","P30","P60")
row.names(Mates) <- names(Sample_M)

# Selfing
Mates[,1] <- as.vector(matrix(1:length(Sample_M), 2))

# Assign moms from same patch as mates for each other

Mates[,2] <- as.vector(matrix(1:length(Sample_M), 2)[2:1,])
```

- M3 and M4: Assign each pair of moms the same two nearest-neighbour mates from patch 205.

```
Dads_205 <- Dads %>% filter(Patch == "205")
# Distance matrix
```

```
Dmat <- as.matrix(dist(Dads_205[,c("X", "Y")]))
diag(Dmat) = 100

# Form pairs of neighbouring dads
Neighbours <- sapply(data.frame(Dmat), function(x) which(rank(x, ties.method="random")==1))
Neighbours <- cbind(First=1:length(Neighbours), Nearest=Neighbours)

# Check distances:
#sort(Dmat[sapply(data.frame(Dmat), function(x) rank(x, ties.method="random")==1)])

# Sample first data
Mates[,3] <- rep(sample(nrow(Neighbours), length(Sample_M)/2, replace=TRUE), each=2)
Mates[,4] <- Neighbours[Mates[,3], 2]
#table(Mates[,3]==Mates[,4])

head(Mates)
```

- M5 and M6: Assign each pair of moms the same mates, one from patch 30 and one from patch 60.

```
Mates[,5] <- rep(sample(length(Alleles_P[["30"]]), length(Sample_M)/2, replace=TRUE), each=2)
Mates[,6] <- rep(sample(length(Alleles_P[["60"]]), length(Sample_M)/2, replace=TRUE), each=2)

head(Mates)
```

e) Generate offspring genotypes

Mating function.

```
getOffspring <- function(Mom=Alleles_M[[1]], Dad=Alleles_P[[1]][[1]])
{
  Offspring <- cbind(sapply(data.frame(t(Mom)), function(x) sample(x,size=1)),
        sapply(data.frame(t(Dad)), function(x) sample(x,size=1)))
}
```

Apply mating function to generate set of offspring.

```
Alleles_Offspring <- list()

# M1 and M2: Selfing or mating with mom from same patch (two seeds each per mom)
for(k in 1:2)
{
    Alleles_Offspring[[k]] <- lapply(rep(1:length(Sample_M), each=2), function(i)
    getOffspring(Sample_M[[i]], Sample_M[[Mates[i,k]]]))
  names(Alleles_Offspring[[k]]) <- paste0(rep(names(Sample_M), each=2), "-",
                                          rep(names(Sample_M)[Mates[,k]], each=2),
                                          "-", "M", k, rep(c("a", "b"), nrow(Mates)))
}

# M3 and M4: Mating with neighboring dads from patch 205 (two seeds each per mom)
for(k in 3:4)
{
```

```r
  Alleles_Offspring[[k]] <- lapply(rep(1:length(Sample_M), each=2), function(i)
    getOffspring(Sample_M[[i]], Alleles_P[["205"]][[Mates[i,k]]]))
  names(Alleles_Offspring[[k]]) <- paste0(rep(names(Sample_M), each=2), "-",
                                          rep(names(Alleles_P[["205"]])[Mates[,k]], each=2),
                                          "-", "M", k, rep(c("a", "b"), nrow(Mates)))
}

# M5: Mating with dad from patch 30 (two seeds per mom)
for(k in 5)
{
  Alleles_Offspring[[k]] <- lapply(rep(1:length(Sample_M), each=2), function(i)
    getOffspring(Sample_M[[i]], Alleles_P[["30"]][[Mates[i,k]]]))
  names(Alleles_Offspring[[k]]) <- paste0(rep(names(Sample_M), each=2), "-",
                                          rep(names(Alleles_P[["30"]])[Mates[,k]], each=2),
                                          "-", "M", k, rep(c("a", "b"), nrow(Mates)))
}

# M6 Mating with dad from patch 60 (two seeds per mom)
for(k in 6)
{
  Alleles_Offspring[[k]] <- lapply(rep(1:length(Sample_M), each=2), function(i)
    getOffspring(Sample_M[[i]], Alleles_P[["60"]][[Mates[i,k]]]))
  names(Alleles_Offspring[[k]]) <- paste0(rep(names(Sample_M), each=2), "-",
                                          rep(names(Alleles_P[["60"]])[Mates[,k]], each=2),
                                          "-", "M", k, rep(c("a", "b"), nrow(Mates)))
}

# Combine simulated offspring into a single list
Alleles_Offspring <- unlist(Alleles_Offspring[1:length(Alleles_Offspring)], recursive=FALSE)
length(Alleles_Offspring)
```

Extract error rates.

```r
Error.rates.loci <- readLines(ColonyInputFile)[23:26]
Error.rates.loci  <- data.frame(Reduce(cbind,sapply(Error.rates.loci , function(x) strsplit(x, split=",
Error.rates.loci [,-1] <- sapply(Error.rates.loci [,-1], function(x) as.numeric(as.character(x)))
names(Error.rates.loci ) <- c("Locus", "Type", "Allelic_dropout", "Other_error")
Error.rates.loci
```

NEW: Add second set of simulated seeds, identical to the first set but with genotyping error.

Simulate genotyping error by adding a random number to the size of alleles with a rate according to "Other.error" rates.

```r
addError <- function(x)
{
  Error <- t(sapply(1:nrow(Error.rates.loci ), function(i) sample(c(0,1), size=2, replace=TRUE,
            p=c(1 - Error.rates.loci $Other_error[i], Error.rates.loci $Other_error[i]))))
  x + Error * sample(1:100, length(Error), replace=TRUE)
}

tmp <- Alleles_Offspring
names(tmp) <- paste(names(Alleles_Offspring), "e", sep=".")
```

```r
tmp <- lapply(tmp, function(ls) addError(ls))
table(sapply(1:length(tmp), function(i) sum(addError(tmp[[i]]) != Alleles_Offspring[[i]])))

Alleles_Offspring <- c(Alleles_Offspring, tmp)
length(Alleles_Offspring)
```

Prepare offspring data for Colony.

```r
Genotypes_NewSeeds  <- sapply(Alleles_Offspring, function(x) paste(as.vector(t(x)), collapse=" "))
Genotypes_NewSeeds <- paste(names(Genotypes_NewSeeds), Genotypes_NewSeeds, sep=" ")
```

Combine with existing seed genotypes (will be included to make things as realistic as possible).

```r
#Genotypes_Seeds <- readLines(ColonyInputFile)[27 + c(1:nSeeds)]
Genotypes_Seeds <- c(Genotypes_Seeds, Genotypes_NewSeeds)
Pop <- factor(substr(Genotypes_Seeds, 1, 2))
length(Genotypes_Seeds)
```

f) Generate Colony input files

Create colony input file for full dataset.

```r
  # File destination
  # ----------------

  r = 4       # Indicate Simulation run number here!

  Path <- "~/Desktop/colonytoo/"               # Folder where Colony will be run (in parallel)
  runName <- paste0("Sim_", r+1000)

  #outFile <- paste0(here::here(), "/ColonyInputFiles/", runName, ".dat")
  outFile <- paste0(Path, runName, ".dat")    # Write file into Colony folder

  # Add header information
  # ----------------------

  a <- 1:length(Genotypes_Seeds)

  tmp <- Header
  tmp[2] <- runName
  tmp[3] <- paste0(length(Genotypes_Seeds), "      ! Number of offspring in the sample")
  tmp[19] <- "0         ! 0/1=DOS/Windows version"
  tmp[21]  <- "2         ! 0/1/2/3=Low/Medium/High/VeryHigh precision"

  write(tmp,file=outFile,append=FALSE)
  write("",file=outFile,append=TRUE)

  # Add seed genotypes
  # ------------------
  write(Genotypes_Seeds[a],file=outFile,append=TRUE)
  write("",file=outFile,append=TRUE)
```

```r
# Add mom genotypes
# ------------------

a2 <- 1:length(Genotypes_Mom)

write(readLines(ColonyInputFile)[29 + nSeeds],file=outFile,append=TRUE)
write(paste0(length(a2), "  ", length(a2),
             "      !numbers of candidate males & females"),
      file=outFile,append=TRUE)
write("",file=outFile,append=TRUE)
write(Genotypes_Mom[a2],file=outFile, append=TRUE)
write("",file=outFile,append=TRUE)
write(Genotypes_Mom[a2],file=outFile, append=TRUE)
write("",file=outFile, append=TRUE)

# Add further information
# ------------------------

write("0  0      !#known father-offspring dyads, paternity exclusion threshold",
    file=outFile,append=TRUE)
write("",file=outFile,append=TRUE)

write(paste0(length(a),
             "  11    !#known mother-offspring dyads, maternity exclusion threshold"),
      file=outFile, append=TRUE)

# Add known dyads
# ---------------

#Dyads <- readLines(ColonyInputFile)[36 + nSeeds + 2 * nMoms + c(1:nSeeds)]
#Dyads <- gsub("_0_20", "_", Dyads) # New: shorten mom names to match offspring names

# Drop 2013 dyads
#Dyad_Year <- sapply(strsplit(Dyads, split="_"), function(x) x[4])
#Dyads <- Dyads[Dyad_Year == "16"]

Seed_ID <- sapply(strsplit(Genotypes_Seeds, split=" "), function(x) x[1])
Dyads <- paste(Seed_ID, substr(Seed_ID, 1, 3), sep=" ")
#NewDyads <- paste(names(Alleles_Offspring),
#      sapply(strsplit(names(Alleles_Offspring), split="-"), function(x) x[1]),
#      sep=" ")
#Dyads <- c(Dyads, NewDyads)
#PopDyad <- factor(substr(Dyads, 1, 2))

write(Dyads[a],file=outFile, append=TRUE)

# Add final information
# ----------------------

write(readLines(ColonyInputFile)[-c(1: (36 + 2 * nSeeds + 2 * nMoms))],
    file=outFile, append=TRUE)
write("",file=outFile,append=TRUE)
```

2. Compile Colony results

Warning: Check that Colony input file is in the same folder as Colony. Should not have any blanks in the path (e.g, run on Desktop, not OneDrive).

a) Run Colony

Run a single file (commented out for knitting to html). If using parallel version of Colony, change accordingly. Note: I could not get the parallel version to work.

```
#Path <- "~/Desktop/colony2.mac/"              # Folder with Colony output
#setwd(Path)
##inFile = "Patch_10e"    # Select file to run
#inFile = runName         # Select file to run

#Call <- paste0("./colony2s.out IFN:", paste0(inFile, ".dat"))  # Serial version
##Call <- paste0("./colony2p.out IFN:", paste0(inFile, ".dat")) # Parallel version

#termId <- rstudioapi::terminalExecute(Call)
#while (is.null(rstudioapi::terminalExitCode(termId))) { Sys.sleep(0.1) }
#rstudioapi::terminalKill(termId)
```

Run in parallel.

```
Path <- "~/Desktop/colonytoo/"              # Folder with Colony output
setwd(Path)
inFile = runName         # Select file to run

#Call <- paste0("./colony2s.out IFN:", paste0(inFile, ".dat"))  # Serial version
Call <- paste0("mpiexec -n 6 ./colony2p.out IFN:", paste0(inFile, ".dat")) # Parallel version


termId <- rstudioapi::terminalExecute(Call)
while (is.null(rstudioapi::terminalExitCode(termId))) { Sys.sleep(0.1) }
rstudioapi::terminalKill(termId)
```

b) Import results

Compile Colony results for a single simulation run (inFile):

```
#inFile = "Sim_1004"                          # Select results for analysis
inFile = runName
Path <- "~/Desktop/colonytoo/"                # Folder with Colony output

Result <- list(
  FullSibDyad = read.csv(paste0(Path, inFile, ".FullSibDyad")),
  HalfSibDyad = read.csv(paste0(Path, inFile, ".HalfSibDyad")),
  Selfer = read.csv(paste0(Path, inFile, ".Selfer")),
  BestConfig_ordered = read.csv(paste0(Path, inFile, ".BestConfig_ordered"), sep=""),
  Inbreeding = read.csv(paste0(Path, inFile, ".Inbreeding"), sep=""),
  Ne = read.csv(paste0(Path, inFile, ".Ne")),
  Paternity = read.csv(paste0(Path, inFile, ".Paternity")),
  DadGenotype = read.csv(paste0(Path, inFile, ".DadGenotype")),
  ParentPair = read.csv(paste0(Path, inFile, ".ParentPair")),
  BestFSFamily = read.csv(paste0(Path, inFile, ".BestFSFamily")),
  BestCluster = read.csv(paste0(Path, inFile, ".BestCluster"))
)
```

c) Annotate best configuration

Extract simulated seeds from Colony file "BestConfiguration".

```
m <- which(sapply(strsplit(as.character(Result$BestConfig_ordered$OffspringID),
                           split="-"), length) == 3)
Data <- Result$BestConfig_ordered[m,]
Data <- data.frame(sapply(Data, as.character), stringsAsFactors = FALSE)
```

Get patch, site, dad and mating type (M) information from offspring name.

```
Data$Patch <- substr(Data$MotherID, start=1, stop=2)

tmp <- as.character(Data$OffspringID)
tmp <- strsplit(tmp, split="-")
Data$M <- sapply(tmp, function(x) substr(x[3], 1, 2))
Data$Replicate <- sapply(tmp, function(x) substr(x[3], 3, 3))  # Will work with revised simulations
Data$Error <- sapply(tmp, function(x) substr(x[3], 5, 5))  # CHECK THIS!

tmp <- sapply(tmp, function(x) x[2])
tmp <- strsplit(tmp, split="_")
Data$Patch.Dad <- sapply(tmp, function(x) x[1])
Data$Patch.Dad <- gsub("[^0-9]", "", Data$Patch.Dad)
tmp <- sapply(tmp, function(x) paste(x[1], x[2], sep="_"))
Data$Dad <- tmp
```

Add probability of selfing from Colony file "Selfer".

```
tmp <- data.frame(sapply(Result$Selfer, as.character), stringsAsFactors = FALSE)
tmp$OffspringID <- gsub("205-", "205_", tmp$OffspringID) # Not needed for revised simulations
Data <- dplyr::left_join(Data, tmp)
names(Data)[names(Data)== "Probability"] <- "P.Selfer"
```

Add probability of first inferred dad from Colony file "ParentPair".

```
tmp <- data.frame(sapply(Result$ParentPair, as.character), stringsAsFactors = FALSE)
tmp <- split(tmp, tmp$OffspringID)
m2 <- which(sapply(tmp, nrow) > 1)
#tmp[m2]

# Drop second assigned fathers
tmp[m2] <- lapply(tmp[m2], function(x) x[1,])
tmp <- Reduce(rbind, tmp)

Data <- left_join(Data, tmp[,c(1,4)])
names(Data)[names(Data)== "Probability"] <- "P.ParentPair"
table(Data$P.ParentPair)
```

Check seeds that have P.ParentPair < cut-off: REMOVE THESE UNASSIGNED SEEDS? None were unassigned.

```r
D.cutoff <- 0.9
Data %>% filter(P.ParentPair < D.cutoff)

Data <- Data %>% filter(P.ParentPair > D.cutoff) # Remove unassigned seeds
```

Split into seeds without / with error: TO DO!

```r
Data.noError <- Data %>% filter(Error != "e")
Data.Error <- Data %>% filter(Error == "e")
```

d) List within-site pairs

Create list of pairs (same site only).

```r
Data2 <- split(Data, Data$Patch) # Only consider pairs within the same patch

tmp <- lapply(Data2, function(ls) Reduce(rbind,lapply(1:(nrow(ls)-1), function(i)
  data.frame(cbind(ls[rep(i, nrow(ls) - i),], ls[(i+1):nrow(ls),])))))
Pairs <- Reduce(rbind, tmp)
rm(tmp, Data2)

names(Pairs) <- paste(rep(names(Data),2), rep(1:2, each=ncol(Data)), sep="_")
Pairs <- data.frame(sapply(Pairs, as.character), stringsAsFactors = FALSE)
```

Add type (“Full”, “Half”, “Other”, “Paternal”) and full-sib probability.

NEW: consider probability for paternal halfsibs.

```r
Pairs$Pair <- paste(Pairs$OffspringID_1, Pairs$OffspringID_2, sep="-")

Data.FullSib <- Result$FullSibDyad %>%
  mutate(P.Full = Probability) %>% select(-Probability)  %>%
  mutate(Pair = paste(OffspringID1, OffspringID2, sep="-"))
Data.HalfSib <- Result$HalfSibDyad %>%
  mutate(P.Half = Probability)  %>% select(-Probability) %>%
  mutate(Pair = paste(OffspringID1, OffspringID2, sep="-"))

Data.Sibs <- full_join(Data.FullSib, Data.HalfSib) %>% select(Pair, P.Full, P.Half)

Pairs <- left_join(Pairs, Data.Sibs)
```

Fix: pairs without P.Full.

```r
Pairs$P.Full[is.na(Pairs$P.Full)] <- 0
```

```r
Pairs <- Pairs %>% mutate(Type.true = "Other") %>%
  mutate(Type.true =replace(Type.true, MotherID_1 != MotherID_2, "Other")) %>%
  mutate(Type.true =replace(Type.true, (MotherID_1 != MotherID_2) &
                                          Dad_1 == Dad_2, "Paternal")) %>%
  mutate(Type.true =replace(Type.true, MotherID_1 == MotherID_2, "Half")) %>%
  mutate(Type.true =replace(Type.true, (MotherID_1 == MotherID_2) &
                            Dad_1 == Dad_2, "Full"))

table(Pairs$Type.true, useNA="ifany")
```

Check probabilities for full or half sibs among pairs, grouped by true sibship type.

```
table((Pairs %>% filter(Type.true == "Full"))$P.Full, useNA="ifany")

table((Pairs %>% filter(Type.true == "Half"))$P.Half, useNA="ifany")

table((Pairs %>% filter(Type.true == "Paternal"))$P.Half, useNA="ifany")

table((Pairs %>% filter(Type.true == "Other"))$P.Half, useNA="ifany")
```

Check (true) paternal halfsibs: these include selfed seeds! Analysis will be based only on M3 - M6 seeds.

```
Pairs %>% filter(Type.true == "Paternal") %>% group_by(M_1, M_2) %>% summarise(n=n())
```

3. Determine error rates

Error rates will be collected in a list.

```
Error.rates <- list()
```

a) Selfing

Question 1: selfing rate among seeds?

- **independent**: seeds that were identified as selfed but in truth were sired by M3, M5 or M6 dads.
- **inbred**: seeds that were identified as selfed but in truth were sired by M2 dads (i.e., moms sampled in any year from the same patch).
- **S.cutoff**: define selfed seeds by cutoff based on probability of selfed.

```
S.cutoff <- 0.9

True <-   Data %>% filter(M=="M1")  # Selfed seeds
False.independent <-  Data %>% filter(is.element(M, c("M3", "M5", "M6")))
False.inbred <-  Data %>% filter(M=="M2")  # Local mates

False.negative.rate <- True %>% summarize(error.rate = mean(P.Selfer <= S.cutoff))
False.positive.rate.independent <- False.independent %>%
  summarize(error.rate = mean(P.Selfer > S.cutoff))
False.positive.rate.inbred <- False.inbred %>%
  summarize(error.rate = mean(P.Selfer > S.cutoff))

cat(paste0("Selfing: False negative rate: ", round(False.negative.rate,3),
            ", false positive rate (independent): ", round(False.positive.rate.independent,3),
           ", false positive rate (inbred): ", round(False.positive.rate.inbred,3),
            ", n1 = ", nrow(True), ", n2 = ", nrow(False.independent),
           ", n3 = ", nrow(False.inbred)," seeds."))

Error.rates$Selfing$False.negative.rate <- as.numeric(False.negative.rate)
Error.rates$Selfing$False.positive.rate.independent <-
  as.numeric(False.positive.rate.independent)
Error.rates$Selfing$False.positive.rate.inbred <-
  as.numeric(False.positive.rate.inbred)
```

b) Near-neighbour mating

- **Local.mate**: Given that the mate was local, what is the probability that the dad was correctly identified, or that the seed was inferred to be selfed, or assigned to another locally sampled dad, or to a sampled dad from another patch, or to an unsampled dad? The FNR was defined as the proportion of seeds that was not assigned to the correct dad.

NOTE: one could include those assigned to another locally sampled dad, as this would still contribute to the estimate of the overall rate of near-neighbour mating. This would not change the numbers much though.

- **Nonlocal.mate**: Given that the mate was not local, what is the probability that the seed was inferred to be selfed, or assigned to any locally sampled dad, or to a sampled dad from another patch, or to an unsampled dad? The FPR was defined as the proportion of seeds that was incorrectly assigned to a locally sampled dad, excluding the maternal plant.

```
True <- Data %>% filter(M=="M2")
False.negative.rate <- True %>% summarize(error.rate = mean(Dad != FatherID))

M1.true <- data.frame(N = nrow(True), True %>%
    summarize(Correct.dad = mean(Dad == FatherID),
              Selfed = mean(MotherID == FatherID),
              Local = mean(Dad != FatherID & MotherID != FatherID &
                             substr(FatherID, 1,2) == Patch),
              Other.site = mean(Dad != FatherID & substr(FatherID,1,1) != "#" &
                                 (substr(FatherID, 1,2) != Patch)),
              Unsampled = mean(substr(FatherID,1,1) == "#")))

# Take 1 seed per family with independent external dad
False <- Data %>% filter(is.element(M, c("M3", "M5", "M6"))) %>%
                  filter(Replicate == "a") %>%
                  filter(Patch != "30" | Patch.Dad != "30") %>%
                  filter(Patch != "60" | Patch.Dad != "60")

M1.false <- data.frame(N = nrow(False), Correct.dad = NA, False %>%
    summarize(Selfed = mean(MotherID == FatherID),
              Local = mean(MotherID != FatherID &
                             substr(FatherID, 1,2) == Patch),
              Other.site = mean(substr(FatherID,1,1) != "#" &
                                 (substr(FatherID, 1,2) != Patch)),
              Unsampled = mean(substr(FatherID,1,1) == "#")))

rbind(True.local.mate = M1.true, True.nonlocal.mate = M1.false)
```

```
cat(paste0("Local pollination: False negative rate: ", round(1 - M1.true$Correct.dad,3),
           ", false positive rate: ", round(M1.false$Local,3),
           ", n1 = ", nrow(True), ", n2 = ", nrow(False), " seeds."))

#sum(M1.true[-1])
#sum(M1.false[-c(1:2)])

# FNR calculated with correct and other local inferred dads:
Error.rates$LocalPollination$False.negative.rate <- 1 - M1.true$Correct.dad - M1.true$Local
Error.rates$LocalPollination$False.positive.rate.independent <- M1.false$Local
Error.rates$LocalPollination$False.positive.rate.inbred <- NA
```

c) Correlated paternity within mom

Question 3: rate of full sibs.

- **independent**: pairs that were identified as fullsibs but in truth were halfsibs sired by independent dads (from different patches: M3, M5 or M6).
- **inbred**: pairs that were identified as fullsibs but in truth were halfsibs sired by nearest-neighbour dads (M3 and M4).
- **S.cutoff**: exclude pairs with at least one selfed seed.

```
F.cutoff = 0.9   # Cutoff for probability of fullsibs

True <- Pairs %>% filter(is.element(M_1, c("M3", "M5","M6"))) %>%
                 filter(is.element(M_2, c("M3", "M5","M6"))) %>%
                 filter(MotherID_1 == MotherID_2) %>%
                 filter(P.Selfer_1 <= S.cutoff & P.Selfer_2 <= S.cutoff) %>%
                 filter(Dad_1 == Dad_2)

False.independent <- Pairs %>% filter(is.element(M_1, c("M3", "M5","M6"))) %>%
                 filter(is.element(M_2, c("M3", "M5","M6"))) %>%
                 filter(MotherID_1 == MotherID_2) %>%
                 filter(P.Selfer_1 <= S.cutoff & P.Selfer_2 <= S.cutoff) %>%
                 filter(Dad_1 != Dad_2)

False.inbred <- Pairs %>% filter(is.element(M_1, c("M3", "M4"))) %>%
                 filter(is.element(M_2, c("M3", "M4"))) %>%
                 filter(MotherID_1 == MotherID_2) %>%
                 filter(P.Selfer_1 <= S.cutoff & P.Selfer_2 <= S.cutoff) %>%
                 filter(Dad_1 != Dad_2)

False.negative.rate <- True %>% summarize(error.rate = mean(P.Full <= F.cutoff))
False.positive.rate.independent <- False.independent %>%
  summarize(error.rate = mean(P.Full > F.cutoff))
False.positive.rate.inbred <- False.inbred %>%
  summarize(error.rate = mean(P.Full > F.cutoff))

cat(paste0("Correlated paternity: False negative rate: ", round(False.negative.rate,3),
            ", false positive rate (independent): ", round(False.positive.rate.independent,3),
           ", false positive rate (inbred): ", round(False.positive.rate.inbred,3),
            ", n1 = ", nrow(True), ", n2 = ", nrow(False.independent),
           ", n3 = ", nrow(False.inbred)," pairs."))

Error.rates$CorrelatedPaternity$False.negative.rate <-
  as.numeric(False.negative.rate)
Error.rates$CorrelatedPaternity$False.positive.rate.independent <-
  as.numeric(False.positive.rate.independent)
Error.rates$CorrelatedPaternity$False.positive.rate.inbred <-
  as.numeric(False.positive.rate.inbred)
```

d) Paternal half-siblings

Question 4: rate of shared dads.

- **independent**: pairs that were identified as parental halfsibs but in truth were sired by dads from different patches (M3, M5 or M6 dads).
- **inbred**: pairs that were identified as parental halfsibs but in truth were sired by two nearest neighbour dads (M3 and M4).
- **S.cutoff**: define selfed seeds by cutoff based on probability of selfed.
- Note that "Pairs" only includes pairs from within the same patch.

```
# True Independent pairs: select paternal halfsib pairs among seeds from M3, M5, M6
True <- Pairs %>% filter(is.element(M_1, c("M3", "M5","M6"))) %>%
                 filter(is.element(M_2, c("M3", "M5","M6"))) %>%
                 filter(MotherID_1 != MotherID_2) %>%
                 filter(Dad_1 == Dad_2)

False.independent <- Pairs %>% filter(is.element(M_1, c("M3", "M5","M6"))) %>%
                 filter(is.element(M_2, c("M3", "M5","M6"))) %>%
                 filter(MotherID_1 != MotherID_2) %>%
                 filter(Dad_1 != Dad_2)

False.inbred <- Pairs %>% filter(is.element(M_1, c("M3", "M4"))) %>%
                 filter(is.element(M_2, c("M3", "M4"))) %>%
                 filter(MotherID_1 != MotherID_2) %>%
                 filter(Dad_1 != Dad_2)

False.negative.rate <- True %>% summarize(error.rate = mean(FatherID_1 != FatherID_2))
False.positive.rate.independent <- False.independent %>%
  summarize(error.rate = mean(FatherID_1 == FatherID_2))
False.positive.rate.inbred <- False.inbred %>%
  summarize(error.rate = mean(FatherID_1 == FatherID_2))

cat(paste0("Paternal Halfsibs: False negative rate: ", round(False.negative.rate,3),
            ", false positive rate (independent): ", round(False.positive.rate.independent,3),
           ", false positive rate (inbred): ", round(False.positive.rate.inbred,3),
            ", n1 = ", nrow(True), ", n2 = ", nrow(False.independent),
           ", n3 = ", nrow(False.inbred)," pairs."))

Error.rates$PaternalHalfsibs$False.negative.rate <-
  as.numeric(False.negative.rate)
Error.rates$PaternalHalfsibs$False.positive.rate.independent <-
  as.numeric(False.positive.rate.independent)
```

```
Error.rates$PaternalHalfsibs$False.positive.rate.inbred <-
  as.numeric(False.positive.rate.inbred)
```

Export list of error rates.

```
#saveRDS(Error.rates, paste0(here::here(), "/Error.rates.noError.", inFile,".rds"))
#saveRDS(Error.rates, paste0(here::here(), "/Error.rates.withError.", inFile,".rds"))
#Error.rates <- readRDS(paste0(here::here(), "/Error.rates.noError.", inFile,".rds"))
#Error.rates <- readRDS(paste0(here::here(), "/Error.rates.withError.", inFile,".rds"))
```

Without genotyping error:

```
Error.rates.Sim_1003 <- readRDS(paste0(here::here(), "/Error.rates.Sim_1003.rds"))

Table.Sim_1003 <- matrix(unlist(Error.rates.Sim_1003), 4, 3, byrow=TRUE,
                         dimnames=list(names(Error.rates.Sim_1003),
                                       names(Error.rates.Sim_1003[[1]])))
round(Table.Sim_1003, 4)
write.table(data.matrix(Table.Sim_1003), "Table.Sim_1003.txt",row.names=FALSE, quote=FALSE, sep="\t")
```

With genotyping error:

```
Error.rates.Sim_1003e <- readRDS(paste0(here::here(), "/Error.rates.Sim_1003e.rds"))

Table.Sim_1003e <- matrix(unlist(Error.rates.Sim_1003e), 4, 3, byrow=TRUE,
                         dimnames=list(names(Error.rates.Sim_1003e),
                                       names(Error.rates.Sim_1003e[[1]])))
round(Table.Sim_1003e, 4)
write.table(data.matrix(Table.Sim_1003e), "Table.Sim_1003e.txt",row.names=FALSE, quote=FALSE, sep="\t")
```

4. Estimate true number of seeds or pairs

a) Function

```
getRate <- function(T, n, SE=NA, sp, se)
{
  p <- T/n

  Estimate <- max(0, (p - (1 - sp)) / (se + sp - 1))
  SE.corrected <- as.numeric(SE / (se + sp - 1))

  data.frame(Observed.rate=p, SE.initial = SE, Estimate=Estimate, SE.corrected=SE.corrected,
             T.obs = T, T.inferred= Estimate * n)
}
```

Example:

```
dat <- Data %>% filter(is.element(M, c("M1", "M3", "M5", "M6"))) %>%
                           filter(Replicate == "a" | M == "M1")
T <- nrow(dat %>% filter(P.Selfer > S.cutoff))
n <- nrow(dat)
SE <- NA
sp <- 1 - Error.rates$Selfing$False.positive.rate.independent
se <- 1 - Error.rates$Selfing$False.negative.rate

getRate(T=T, n=n, SE=NA, sp=sp, se=se)
```

b) Observed rates

```
Observed.rates <- list()
Observed.rates$CorrelatedPaternity$T = 636
Observed.rates$CorrelatedPaternity$n = 4604
Observed.rates$CorrelatedPaternity$SE.initial = 0.005107005
```

```
tmp1 <- Observed.rates$CorrelatedPaternity
tmp2 <- Error.rates$CorrelatedPaternity
getRate(T=tmp1$T, n=tmp1$n, SE=tmp1$SE.initial,
         sp= 1-tmp2$False.positive.rate.independent, se= 1-tmp2$False.negative.rate)
```