Simulation Modeling of Pollen Loads, Inference of Pollinator Foraging Tactics, and Sensitivity Analysis

Contents

1. Preparations

Load packages, import data, and define functions used for simulating pollen loads and calculating expected values of mating system parameters from them.

Load packages

Import observed mating system parameters

Import observed mating system parameters (corrected for false positive/negative rates).

- Corrected_Rate: observed rates, corrected for FPR/FNR determined from simulation without genotyping error. Not used here.
- Corrected_Rate_Mismatches: observed rates, corrected for FPR/FNR determined from simulation with genotyping error. These were used here.
- Uncorrected: uncorrected observed rates. Not used here.

```
Rates.table <- readRDS(paste0(here::here(), "/parameters_new_errors.rds")) #"/FPR_FNR_Error_Rates.rds"
row.names(Rates.table) <- Rates.table$Paremeter
Rates.table <- Rates.table[,-1]

Rates.table$Uncorrected <- c(0.02583026, 0.01262626, 0.2136364, 0.1296044, 0.008493686)

Rates.table
```

Create list of lists to compile results

```
Fitted <- list(p.fit = data.frame(matrix(NA, nrow=ncol(Rates.table),ncol=4,
                        dimnames=list(names(Rates.table), c("fit", "min", "max", "SE")))),
                      p.SST = list(),
                      p = list(),
                      p.RMV = list(),
                      N.NBR = list(),
                      Set = list(),
                      Smoothing = list(),
                      Scenarios = list(),
                      Sensitivity.Table = list(),
                      Set.Table = list(),
                      Bias = list(),
                      p.TF = list(),
                      p2 = list(),
                      nTotal = list())

#Fitted <- readRDS(paste0(here::here(), "/Fitted.rds"))
```

Function `get.p.fit.within`

Function to estimate the parameter p of the geometric distribution assuming complete traplining (`p.SST = 0`).

- Est: estimate of CP.within.fruit.
- n: number of pollen donors for which to calculate probability to include in sum.
- Function calculated the expected CP for a range of parameter values of p and identifies the value that results in CP closest to the observed rate.

```
get.p.fit.within <- function(Est=0.2089927, n = 100)
{
  p <- 1/seq(2,4,0.001)
  CP <- sapply(p, function(x) sum(dgeom(c(1:n)-1, x)^2))

  p.fit.within <- p[which(abs(CP - Est) == min(abs(CP - Est)))]
  p.fit.within
}
```

Function `get.donorlist`

Function to simulate pollen loads:

```
get.donorlist <- function(p.SST=0, N.NBR=5, p.RMV=1, nFruit=135, p=1/3,
                          p.TF=0, p2=1/2, nTotal=120, nSites=100)
{
  # Version date: Feb 18, 2022
  if(p.TF > 0 & p.SST > 0) cat("Please set p.TF or p.SST to 0!")

  Routes <- list()
  for(m in 1:N.NBR) Routes[[m]] =  paste(LETTERS[m], c(1:nSites), sep=".")
  names(Routes) <- LETTERS[1:N.NBR]
```

```
Clusters <- list()
for(m in 1:round(nTotal/N.NBR)) Clusters[[m]] = paste("L",m, c(1:N.NBR), sep=".")
Clusters[[1]] <- paste(LETTERS[1:N.NBR], 0, sep=".")

# Create exhaustive list of pollen donors, same list for each mom
tmp <- unlist(Routes)
tmp.2 <- unlist(Clusters[-1])
DonorList <- data.frame(PollenDonors = c(paste(names(Routes), 0, sep="."), tmp,
                        as.vector(outer(paste0(tmp, "_"), c(2:N.NBR), paste0)),
                        unlist(Clusters[-1]))) # NEW
rownames(DonorList) <- DonorList$PollenDonors
DonorList <- rep(list(DonorList), N.NBR)

# Here, fruit = day: fruits of the same day are correlated if TF
# i.e., the same bird visits the N.NBR plants in random order

for(f in 1:nFruit)
{
  # Is today's bird a territorial forager?
  is.TF = FALSE
  if(p.TF > 0 & runif(1) < p.TF)
  {
    is.TF = TRUE
    a.TF = sample(N.NBR)
    PollenDonors.TF <- lapply(Clusters, sample)
    PollenDonors.TF <- PollenDonors.TF[c(1, sample(2:length(Clusters)))]
  }

  for(m in 1:N.NBR)
  {
    tmp1 <- c()
    j.start = 1

    # Territorial foraging module:
    # ----------------------------
    if(is.TF == TRUE)
    {
      PollenDonors <- PollenDonors.TF
      PollenDonors[[1]] <- Clusters[[1]][a.TF[cumsum(a.TF==m)<1]]
      PollenDonors <- unlist(PollenDonors)
    }

    # Trapling foraging module:
    # -------------------------
    if(is.TF == FALSE)
    {
      # Select which route will be sampled:
      # -----------------------------------
      # With probability (1-p.shift), this will be route m.
      # With probability of p.shift, this will be a randomly chosen route (other than m)
      # All other routes have the same probability
      # It is assumed that 'sampling' before 'shifting' does not result in pollination
      # and that the pollinator only shifts once per day and resource location.
```

```
    # Hence, no correlated pollinations on the same day (different from TF).

    # Random shifting according to p.SST and N.NBR:
    Steps <- sample(1:N.NBR, 1, prob=c(rep(p.SST/(N.NBR-1), N.NBR-1), 1-p.SST))
    a <- c(1:N.NBR, 1:N.NBR)[Steps + c(1:N.NBR)]

    # Complete trapline foraging sub-module:
    # --------------------------------------
    PollenDonors <- Routes[[a[m]]]

    # Sample-and-shift foraging sub-module:
    # -------------------------------------
    # This will modify the list of pollen donors from the
    # complete trapline foraging sub-module

    if(p.SST > 0 & Steps != N.NBR)
    {
      tmp1 = paste(names(Routes)[a[m]], 0,sep=".")
      PollenDonors <- c(tmp1, PollenDonors)
      j.start = 2
    }

    # Simulate sample-and-shift at previously visited sampling locations
    Add <- c()
    for(j in j.start:length(PollenDonors))
    {
      if(runif(1) < p.SST)
      {
        new = paste(PollenDonors[j], sample(c(2:N.NBR),1), sep="_")
        if(N.NBR == 2) new = paste(PollenDonors[j], 2, sep="_")
        Add = c(Add, new)
      }
    }

    if(length(Add) > 0)
    {
      PollenDonors <- c(PollenDonors, Add)
      tmp <- split(PollenDonors, sapply(strsplit(PollenDonors, split="_"),
                                        function(x) x[[1]][1]))
      tmp <- tmp[order(as.numeric(sapply(strsplit(names(tmp), split="[.]"),
                                         function(x) x[[2]])))]
      PollenDonors <- as.character(unlist(sapply(tmp,
                                    function(x) sort(x, decreasing=TRUE))))
      tmp1 <- c(tmp1, sapply(strsplit(Add, split="_"), function(x) x[[1]][1]))
    }

  }

  # For all foraging types:
  # -----------------------
  # Determine probability of each pollen donor to sire seed.
  # For SST, reduce probabilities for sampled plants according to p.RMV:
```

```
      if(is.TF==TRUE)
      {
        Prob <- p2
        for(d in 2:length(PollenDonors)) Prob[d] = p2 * (1-sum(Prob))
      }

      if(is.TF==FALSE)
      {
        b <- !is.element(PollenDonors, tmp1)
        Prob <- p
        if(j.start==2) Prob = p * p.RMV
        for(d in 2:length(PollenDonors)) Prob[d] = p * (1-sum(Prob)) * c(p.RMV, 1)[b[d]+1]
      }

      # Add column of pollen donor probabilities for fruit f of plant m to DonorList:
      tmp <- data.frame(PollenDonors, Prob)
      names(tmp)[2] <- paste0("F",f)
      DonorList[[m]] <- left_join(DonorList[[m]], tmp,
                                  by=c("PollenDonors" ="PollenDonors"))
    }
  }

  # Tidy up DonorList:
  # ------------------
  for(m in 1:N.NBR)
  {
    rownames(DonorList[[m]]) <- DonorList[[m]]$PollenDonors
    DonorList[[m]] <- DonorList[[m]][,-1]
    DonorList[[m]][is.na(DonorList[[m]])] <- 0
    colSums(DonorList[[m]])
  }
  names(DonorList) <- names(Routes)
  DonorList
}
```

Function `get.CP`

Function to extract mating system parameters from simulated pollen distributions.

```
get.CP <- function(DonorList)
{
  # Version date: Feb 18, 2022

  N.NBR=length(DonorList)
  CP.res <- data.frame(matrix(NA, 2, 6, dimnames=list(c("All", "Sample"),
                c("CP.within.fruit", "CP.between.fruit", "CP.within.mom",
                  "CP.between.mom", "Local", "Nep"))))

  # Calculate expected rates based on probabilities

  res <- data.frame(matrix(NA, N.NBR, 3, dimnames=list(names(DonorList),
                    c("CP.within.fruit", "CP.between.fruit", "Local"))))
  for(m in 1:length(DonorList))
```

```
  {
    tmp <- t(data.matrix(DonorList[[m]])) %*% data.matrix(DonorList[[m]])
    CP.within.fruit <- mean(diag(tmp))
    CP.between.fruit <- mean(tmp[lower.tri(tmp)])
    Local <- mean(colSums(DonorList[[m]][1:N.NBR,]))
    res[m,] <- c(CP.within.fruit, CP.between.fruit, Local)
  }

  CP.res["All", "CP.within.fruit"] <- mean(res$CP.within.fruit)
  CP.res["All", "CP.between.fruit"] <- mean(res$CP.between.fruit)
  CP.res["All", "Local"] <- mean(res$Local)

  tmp <-sapply(DonorList, function(ls) apply(ls, 1, mean))

  tmp <- t(tmp) %*% tmp
  CP.res["All", "CP.within.mom"] <- mean(diag(tmp))
  CP.res["All", "CP.between.mom"] <- mean(tmp[lower.tri(tmp)])
  CP.res["All", "Nep"] <- 1/CP.res["All", "CP.within.mom"]

  CP.res
}
```

```
get.CP.site <- function(DonorList)
{
  # Version date: April 12, 2022

  N.NBR=length(DonorList)
  CP.res <- data.frame(matrix(NA, 2, 8, dimnames=list(c("All", "Sample"),
                c("CP.within.fruit", "CP.between.fruit", "CP.within.mom",
                  "CP.between.mom", "Local", "Nep", "CP.within.site", "Nep.site"))))

  # Calculate expected rates based on probabilities

  res <- data.frame(matrix(NA, N.NBR, 3, dimnames=list(names(DonorList),
                    c("CP.within.fruit", "CP.between.fruit", "Local"))))
  for(m in 1:length(DonorList))
  {
    tmp <- t(data.matrix(DonorList[[m]])) %*% data.matrix(DonorList[[m]])
    CP.within.fruit <- mean(diag(tmp))
    CP.between.fruit <- mean(tmp[lower.tri(tmp)])
    Local <- mean(colSums(DonorList[[m]][1:N.NBR,]))
    res[m,] <- c(CP.within.fruit, CP.between.fruit, Local)
  }

  CP.res["All", "CP.within.fruit"] <- mean(res$CP.within.fruit)
  CP.res["All", "CP.between.fruit"] <- mean(res$CP.between.fruit)
  CP.res["All", "Local"] <- mean(res$Local)

  tmp <-sapply(DonorList, function(ls) apply(ls, 1, mean))

  # NEW April 2022: Nep.site
  tmp2 <- apply(tmp, 1, mean)
  #CP.within.site <- t(tmp2) %*% tmp2
```

```
  #Nep.site <- 1/CP.within.site
  #c(CP.within.site, Nep.site)
  CP.res["All", "CP.within.site"] <- t(tmp2) %*% tmp2
  CP.res["All", "Nep.site"] <- 1/CP.res["All", "CP.within.site"]

  tmp <- t(tmp) %*% tmp
  CP.res["All", "CP.within.mom"] <- mean(diag(tmp))
  CP.res["All", "CP.between.mom"] <- mean(tmp[lower.tri(tmp)])
  CP.res["All", "Nep"] <- 1/CP.res["All", "CP.within.mom"]
  #1/mean(diag(tmp))

  CP.res
}
```

Wrapper function `fx`

This function:

- Simulates pollen distributions (pollen loads).
- Calculates CP values.

```
fx <- function(y=0, z=0, w=5, nFruit=135, nSites=100,
               u=0, v=Fitted$p.fit[r,1], t=120, p=Fitted$p.fit[r,1])
{
  # Version date: April 1, 2021

  D <- get.donorlist(p.SST=y, p.RMV=z, N.NBR=w, nFruit=nFruit, p=p,
                     p.TF=u, p2=v, nTotal=t)
  CP <- get.CP(D)
  unlist(CP["All",])
}
```

2. Pattern-oriented simulation modeling for complete vs. sample-and-shift traplining

Step 1: Expressing p.RMV as a function of p and p.SST

Special case: complete traplining Under complete traplining, p.SST = 0, and p.RMV is irrelevant as no plants are "sampled". Thus, p can be estimated directly from the observed correlated paternity within fruits (assuming that each fruit results from pollination by a single pollinator visit).

Here we apply the function `get.p.fit.within` to get estimate of p ± SE for complete traplining.

```
Fitted$p.fit <- data.frame(matrix(NA, ncol(Rates.table), 3,
                           dimnames=list(names(Rates.table), c("fit", "min", "max"))))
for(r in 1:ncol(Rates.table))
{
  # Version date: 9 March 2021

  Estimate = Rates.table["Within Fruit Outcrossed", r]
  SE = SE.table["Within Fruit Outcrossed", r]
```

```
  Fitted$p.fit[r,1:3] <- c(fit = get.p.fit.within(Est=Estimate, n = 100),
                           min = get.p.fit.within(Est=Estimate - SE, n = 100),
                           max = get.p.fit.within(Est=Estimate + SE, n = 100))
  Fitted$p.fit[r,"SE"] <- (Fitted$p.fit[r,"max"]  - Fitted$p.fit[r,"min"] )/2
}

Fitted$p.fit
```

Model the rate of local pollination Relationship between `p`, `p.SST` and `p.RMV` to express `p.RMV` as a function of `p` and `p.SST`: `p.RMV = Estimate / (p * p.SST)`.

```
r = 2  # Select set of observed rates

Estimate <- Rates.table["Local Mating Outcrossed", r]

P.RMV.opt <- expand.grid(p=seq(0.25,0.5,0.05),
                         p.SST=sort(c(seq(0.05,0.3,0.05), c(0.125, 0.175))))
P.RMV.opt <- P.RMV.opt %>%
  mutate(p.RMV=Estimate / ( p * p.SST)) %>%
  mutate(p.RMV = replace(p.RMV, p.RMV > 1, NA)) %>%
  filter(!is.na(p.RMV))

Fitted$p.RMV[[r]] <- P.RMV.opt
```

Step 2: Expressing p as a function of p.SST

Model CP within fruit Model CP within fruit as a function of `p`, `p.SST` and `p.RMV` to express `p` as a function of `p.SST` (and the associated value of `p.RMV`).

```
X <- Fitted$p.RMV[[r]]
R <- 30
X <- X[rep(1:nrow(X),R),]
Res <- data.frame(X, Reduce(rbind, pbmcapply::pbmclapply(1:nrow(X), function(i)
                      fx(y=X$p.SST[i], z=X$p.RMV[i], w=10, p=X$p[i]),
                         mc.cores=parallel::detectCores())))

Res.p <- Res
saveRDS(Res.p, paste0(here::here(), "/Res.p.rds"))
#Res.p <- readRDS(paste0(here::here(), "/Res.p.rds"))

Estimate <- Rates.table["Within Fruit Outcrossed", r]
```

Optimization: `p.opt` as a function of `p.SST`

```
optimize.p <- function(x, E, a, b1, b2)
{
  (E - (a + b1 * x + b2 * I(x^2)))^2
}

get.p <- function(ls)
{
```

```
  # Version date: April 8, 2021

  mod1 <- lm(CP.within.fruit ~ p + p.2,
                       data=ls %>% mutate(p.2 = p^2))
  p.opt <- NA
  if(sum(is.na(mod1$coef)) == 0)
  {
    p.opt <- optimize(optimize.p, interval=c(0.25, 0.5),
                      E=Estimate, a=mod1$coef[1], b1=mod1$coef[2], b2=mod1$coef[3])$minimum
  }

  c(a=mod1$coef[1], b1=mod1$coef[2], b2=mod3$coef[3], p.opt=p.opt)
}

Res.p <- Res.p %>% mutate(p.SST.f = factor(p.SST))

test <- split(Res.p, Res.p$p.SST.f)  # Removed splitting by N.NBR

P.opt <- data.frame(Reduce(rbind,lapply(test, get.p)))
P.opt <- P.opt %>% mutate(p.RMV = as.vector(sapply(test, function(x) x$p.RMV[1])),
         p.SST = as.vector(sapply(test, function(x) x$p.SST[1]))) %>%
         mutate(p.SST.f = factor(p.SST))

names(P.opt) <- c("intercept", "slope.b1", "slope.b2", "p.opt",
                  "p.RMV", "p.SST", "p.SST.f")
P.opt <- P.opt %>% filter(!is.na(p.opt))

Fitted$p[[r]] <- P.opt
```

Plot:

```
p1 <- ggplot(Res.p %>% mutate(p.SST=p.SST.f),
             aes(x=p, y=CP.within.fruit, color=p.SST)) +
  geom_point() + geom_hline(yintercept = Estimate) + theme_bw() +
  geom_smooth(method="lm", se=FALSE, formula=y ~ x + I(x^2))
```

Express p as a function of p.SST:

```
tmp<- rbind(P.opt, c(NA, NA, NA, Fitted$p.fit[r,"fit"], NA, 0, NA))
p2 <- ggplot(tmp , aes(x=p.SST, y=p.opt)) +
  geom_point(color=c(rep("black", nrow(P.opt)), "violetred2")) +
  stat_smooth(data=P.opt %>% filter(p.SST >= 0.1), method="lm", se=FALSE,
              formula=y ~ x + I(x^2))

g1 <- gridExtra::grid.arrange(p1, p2, nrow = 1)
grid::grid.draw(g1)
```

Export figures:

```
pdf(paste0(here::here(), "/Fit_p.opt.pdf"), width=8, height=3)
grid::grid.draw(g1)
dev.off()
```

```
Models=list( mod1=lm(p.opt ~ p.SST ,
                     data=P.opt %>% filter(p.SST >= 0.1)),
             mod3=lm(p.opt ~ p.SST + p.SST.2,
                     data=P.opt %>% filter(p.SST >= 0.1) %>%
                     mutate(p.SST.2 = p.SST^2)))

a <- order(sapply(Models, AIC))[1]
best <- Models[a][[1]]
summary(best)

P.opt$p.smooth[P.opt$p.SST >= 0.1] <- fitted(best)

Fitted$p[[r]] <- P.opt
Fitted$Smoothing[[r]] <- list()
Fitted$Smoothing[[r]]$p <- best
```

Step 3: Expressing N.NBR as a function of p.SST

Model CP between moms Model CP between moms as a function of `p.SST` (and associated values of `p` and `p.RMV`) and `N.NBR` to express `N.NBR` as a function of `p.SST`.

```
X <- expand.grid(p.SST=Fitted$p[[r]] %>% pull(p.SST),
            N.NBR=c(3,5,6,7,8,9,12,18,25))
X <- left_join(X, Fitted$p[[r]] %>% mutate(p=p.smooth) %>% select(p.SST, p.RMV, p))
X <- X %>% filter(!is.na(p))

R <- 10
X <- X[rep(1:nrow(X),R),]

Res <- data.frame(X, Reduce(rbind, pbmcapply::pbmclapply(1:nrow(X), function(i)
                      fx(y=X$p.SST[i], z=X$p.RMV[i], w=X$N.NBR[i], p=X$p[1]),
                         mc.cores=parallel::detectCores())))

Res.N.NBR <- Res %>% mutate(p.SST.f=factor(p.SST))
saveRDS(Res.N.NBR, paste0(here::here(), "/Res.N.NBR.rds"))
#Res.N.NBR <- readRDS(paste0(here::here(), "/Res.N.NBR.rds"))

Estimate <- Rates.table["Between Moms Outcrossed", r]
```

```
p1 <- ggplot(data = Res.N.NBR , aes(x = N.NBR, y = CP.between.mom, col = p.SST.f)) +
  geom_point(size=0.8) +
  scale_x_continuous(trans='log', breaks=c(3,5,6,7,8,9,12,18,25),
                     minor_breaks=NULL) +
  scale_y_continuous(trans='log', breaks=c(0.002, 0.005, 0.01, 0.02),
                      minor_breaks=NULL) +
  geom_smooth(method=lm, se=FALSE, size=0.8) + theme_bw() +
  geom_hline(yintercept=Estimate)

p1
```

Fit separate intercepts and slopes:

```
mod.lm <- lm(log(CP.between.mom) ~ log(N.NBR) * p.SST.f, data=Res.N.NBR)
n=nlevels(Res.N.NBR$p.SST.f)
intercept <- mod.lm$coefficients[c(1:(n+1))[-2]]
intercept[-1] <- intercept[-1] + intercept[1]
slope <- mod.lm$coefficients[c(2, c(2:n) + n)]
slope[-1] <- slope[-1] + slope[1]

df <- data.frame(p.SST.f=levels(Res.N.NBR$p.SST.f), intercept=intercept, slope=slope)
df <- df %>% mutate(N.NBR.opt = exp((log(Estimate) - intercept) / slope),
                    p.SST=as.numeric(p.SST.f)) %>%
             mutate(p.SST.2 = p.SST^2)
```

Express N.NBR as a function of p.SST.

```
mod2 <- lm(N.NBR.opt ~ p.SST + p.SST.2, data=df)
summary(mod2)

df$smooth <- predict(mod2, df)
df$N.NBR.fit <- round(df$smooth)

Fitted$N.NBR[[r]] <- df
#Fitted$Smoothing[[r]] <- list()
Fitted$Smoothing[[r]]$N.NBR <- mod2
```

```
p2 <- ggplot(data = df, aes(x = p.SST, y = N.NBR.opt)) +
  geom_point() +
  geom_smooth(method="lm", se=FALSE, formula='y ~ x + I(x^2)')

g1 <- gridExtra::grid.arrange(p1, p2, nrow = 1, widths=c(0.57, 0.43))
grid::grid.draw(g1)
```

Export figures:

```
pdf(paste0(here::here(), "/Fit_N.NBR.opt.pdf"), width=8, height=3)
grid::grid.draw(g1)
dev.off()
```

Step 4: Optimizing p.SST

Model CP between fruits Model CP between fruits as a function of p.SST (and the associated values of p, p.RMV and N.NBR): what rate of sample-and-shift behaviour would result in the observed CP between fruits?

```
#r = 2  # Select set of rates

X <- Fitted$N.NBR[[r]] %>% mutate(N.NBR=N.NBR.fit) %>%
                           select(p.SST, p.SST.2, N.NBR)
X <- X %>% mutate(p=predict(Fitted$Smoothing[[r]]$p, newdata=X))
X <- X %>% mutate(p.RMV=Rates.table["Local Mating Outcrossed",r]/(p * p.SST))

R <- 30
```

```
X <- X[rep(1:nrow(X),R),]

Res <-data.frame(X, Reduce(rbind, pbmcapply::pbmclapply(1:nrow(X), function(i)
                        fx(y= X$p.SST[i], z=X$p.RMV[i], w=X$N.NBR[i], p=X$p[i]),
                           mc.cores=parallel::detectCores())))

Res.p.SST <- Res
saveRDS(Res.p.SST, paste0(here::here(), "/Res.p.SST.rds"))
#Res.p.SST <- readRDS(paste0(here::here(), "/Res.p.SST.rds"))

Estimate <- Rates.table["Between Fruit Outcrossed", r]
```

```
Dat <- Res.p.SST %>% group_by(p.SST) %>%
  summarize(CP.between.fruit=mean(CP.between.fruit))

ggplot(Dat, aes(x=p.SST, y=CP.between.fruit)) + geom_point() +
    geom_hline(yintercept=Estimate) +
    stat_smooth(method="glm", se=F, formula='y ~ x',
                method.args = list(family="gaussian"(link=log)))

ggsave("Fit_p.SST.pdf", width = 5, height = 3.5, units = "in")
```

```
mod.glm <- glm(CP.between.fruit ~ p.SST, data=Res.p.SST, family=gaussian(link="log"))
summary(mod.glm)

p.SST.opt <- as.vector((log(Estimate) -mod.glm$coef[1]) / mod.glm$coef[2])

P.SST.opt <- data.frame(p.SST=p.SST.opt) %>% mutate(p.SST.2=p.SST^2)
P.SST.opt <- P.SST.opt %>%
  mutate(p=predict(Fitted$Smoothing[[r]]$p, newdata=P.SST.opt)) %>%
  mutate(p.RMV=Rates.table["Local Mating Outcrossed",r]/(p * p.SST)) %>%
  mutate(N.NBR = predict(Fitted$Smoothing[[r]]$N.NBR, newdata=P.SST.opt))

P.SST.opt <- P.SST.opt %>% select(-p.SST.2) %>%
  mutate(N.NBR.smooth=N.NBR) %>%
  mutate(N.NBR=round(N.NBR))
Fitted$p.SST[[r]] <- P.SST.opt
Fitted$Smoothing[[r]]$p.SST <- mod.glm

#saveRDS(Fitted, paste0(here::here(), "/Fitted.rds"))
```

3. Pattern-oriented simulation modeling for territorial foraging

Fit pTF to CP.between.fruit

```
r = 2  # Select set of observed rates
gc() # garbage collection - avoid errors

X <- expand.grid(p=Fitted$p.fit$fit[r],
                     p.TF=sort(unique(c(seq(0.05,0.5,0.05)))),
                     N.NBR=c(4,6,8,10))
```

```
X <- X %>% mutate(p2=p)

R <- 30
X <- X[rep(1:nrow(X),R),]
Res <- data.frame(X, Reduce(rbind, pbmcapply::pbmclapply(1:nrow(X), function(i)
                        fx(u=X$p.TF[i], v=X$p2[i], w=X$N.NBR[i], p=X$p[i]),
                           mc.cores=parallel::detectCores())))

Res.p.TF <- Res
saveRDS(Res.p.TF, paste0(here::here(), "/Res.p.TF.rds"))
#Res.p.TF <- readRDS(paste0(here::here(), "/Res.p.TF.rds"))
```

```
Estimate <- Rates.table["Between Fruit Outcrossed", r]

Dat.p.TF <- Res.p.TF %>% group_by(p.TF, N.NBR) %>%
  summarize(CP.between.fruit=mean(CP.between.fruit),
            CP.between.mom=mean(CP.between.mom),
            Local=mean(Local))

plot1 <- ggplot(Dat.p.TF %>% mutate(N.NBR=factor(N.NBR)),
       aes(p.TF, CP.between.fruit, group=N.NBR, col=N.NBR)) +
  geom_point() + ylim(0, 0.2) +
  stat_smooth(method="lm", se=F, formula='y ~ x + I(x^2)') +
  geom_hline(yintercept=Estimate) +
  geom_segment(x=0.25, y=0, xend=0.25, yend=Estimate,
               linetype = "dashed", col="darkgray", size=0.7)
plot1 + theme_light()
```

Optimize/smooth p.TF as a function of N.NBR (to fit CP.between.fruit).

```
r=2
Estimate <- Rates.table["Between Fruit Outcrossed", r]

optimize.p.TF <- function(x, E, a, b1, b2)
{
  (E - (a + b1 * x + b2 * I(x^2)))^2
}

get.p.TF <- function(ls)
{
  # Version date: March 13, 2022

  #mod1 <- lm(CP.between.fruit ~ p2, data=ls)
  mod1 <- lm(CP.between.fruit ~ p.TF + I(p.TF^2), data=ls)
  p.TF.opt <- NA
  if(sum(is.na(mod1$coef)) == 0)
  {
    p.TF.opt <- optimize(optimize.p.TF, interval=c(0.01, 1),
                         E=Estimate, a=mod1$coef[1], b1=mod1$coef[2],
                         b2=mod1$coef[3])$minimum
  }
```

```
  c(a=mod1$coef[1], b1=mod1$coef[2], p.TF.opt=p.TF.opt)
}

# Limit to range that has sufficient data and results in positive values of p.TF.
tmp <- Res.p.TF %>% mutate(N.NBR = factor(N.NBR))
test <- split(tmp, tmp$N.NBR)
P.TF.opt <- data.frame(Reduce(rbind,lapply(test, get.p.TF)))

P.TF.opt$N.NBR <- as.numeric(names(test))
names(P.TF.opt)[1:2] <- c("Intercept", "slope")
rownames(P.TF.opt) <- NULL
P.TF.opt
```

Plot against CP.between.mom and Local mating

Note: models were only fitted with ggplot, not numerically, as there was no fit consistent with all mating system parameters.

```
r=2
Estimate <- Rates.table["Between Moms Outcrossed", r]

# Linear model
plot2 <- ggplot(Dat.p.TF %>% mutate(N.NBR=as.factor(N.NBR)),
         aes(x=p.TF, y=CP.between.mom, group=N.NBR, color=N.NBR)) +
  geom_smooth(method="lm", se=FALSE, formula='y ~ x + I(x^2)') + geom_point() +
  geom_hline(yintercept = Estimate) +
  geom_vline(xintercept = 0.25, linetype = "dashed",
             col="darkgray", size=0.7)
plot2 + theme_light()
```

```
r=2
Estimate <- Rates.table["Local Mating Outcrossed", r]

# Linear model
plot3 <- ggplot(Dat.p.TF %>% mutate(N.NBR=as.factor(N.NBR)),
         aes(x=p.TF, y=Local, group=N.NBR, color=N.NBR)) +
  geom_smooth(method="lm", se=FALSE, formula='y ~ x') + geom_point() +
  geom_hline(yintercept = Estimate) + ylim(-0.01,0.4) +
  geom_vline(xintercept = 0.25, linetype = "dashed",
             col="darkgray", size=0.7)
plot3 + theme_light()
```

```
gTF <- cowplot::plot_grid(plot1 + theme_light(), plot2 + theme_light(),
                          plot3 + theme_light(), align = "v", ncol = 1)
ggsave(file="Fitting.TF.pdf", plot=gTF, width=6, height = 7)
gTF
```

4. Simulate mating system parameters with the inferred parameter values

Scenarios:

- Complete traplining.
- Inferred tactic of SST.
- Inferred rate of territorial foraging (0.25) (not included in manuscript).
- Territorial foraging only.

```
r=2
X <- Fitted$p.SST[[r]] %>% select(-N.NBR.smooth)
X[2,] <- c(0,get.p.fit.within(Rates.table["Within Fruit Outcrossed",r]), 1, X[1,4])
X$p2 = NA
X$p.TF=0

X[3,] <- c(0, get.p.fit.within(Rates.table["Within Fruit Outcrossed",r]), 1, X[1,4],
           get.p.fit.within(Rates.table["Within Fruit Outcrossed",r]), 0.25)
X[4,] <- c(0, get.p.fit.within(Rates.table["Within Fruit Outcrossed",r]), 1, X[1,4],
           get.p.fit.within(Rates.table["Within Fruit Outcrossed",r]), 1)

R <- 100
X <- X[rep(1:nrow(X),R),]

Res <- data.frame(X, Reduce(rbind, pbmcapply::pbmclapply(1:nrow(X), function(i)
                       fx.site(y=X$p.SST[i], z=X$p.RMV[i], w=X$N.NBR[i], p=X$p[1],
                          u=X$p.TF[i], v=X$p2[i]),
                          mc.cores=parallel::detectCores())))
Res.final <- Res
```

```
Set.Table <- Res.final %>%
  group_by(p.SST, p.RMV, p.TF, p, N.NBR) %>%
  summarize(CP.within.fruit.mean=mean(CP.within.fruit),
            CP.within.fruit.s=sd(CP.within.fruit),
            CP.between.fruit.mean=mean(CP.between.fruit),
            CP.between.fruit.s=sd(CP.between.fruit),
            CP.within.mom.mean=mean(CP.within.mom),
            CP.within.mom.s=sd(CP.within.mom),
            CP.between.mom.mean=mean(CP.between.mom),
            CP.between.mom.s=sd(CP.between.mom),
            Local.mating.mean=mean(Local),
            Local.mating.s=sd(Local),
            Nep.mean=mean(Nep),
            Nep.s=sd(Nep),
            Nep.site.mean=mean(Nep.site),
            Nep.site.s=sd(Nep.site))

#View(Set.Table)

write.table(format(as.data.frame(Set.Table), digits=3),
            paste0(here::here(), "/Set.Table.txt"), sep="\t", row.names=FALSE, quote=FALSE)

#Fitted$Set.Table <- list()
Fitted$Set.Table[[r]] <- Set.Table
saveRDS(Fitted, paste0(here::here(), "/Fitted.rds"))
```

5. Sensitivity analysis for complete vs. sample-and-shift trapling foraging

Simulate pollen loads for a broad range of trapline foraging parameter values to assess their effect on mating system parameters.

```
r = 2  # Select set of rates

X <- expand.grid(p.SST = c(0, 0.131, 0.5, 1),
                 N.NBR = c(6, 20),
                 p.RMV = c(0, 0.056, 0.1, 0.2, 0.5, 1),
                 p=c(0.2, 0.366, 0.5, 1))
R <- 5
X <- X[rep(1:nrow(X),R),]
X.f <- X %>% mutate(p.SST.f = factor(p.SST), p.RMV.f=factor(p.RMV),
                                    N.NBR.f=factor(N.NBR), p.f=factor(p))

Res <-data.frame(X, Reduce(rbind, pbmcapply::pbmclapply(1:nrow(X), function(i)
                    fx(y=X$p.SST[i], z= X$p.RMV[i], w=X$N.NBR[i], p=X$p[i]),
                    mc.cores=parallel::detectCores())))

Sensitivity <- list(X=X, X.f=X.f, Res=Res)

saveRDS(Sensitivity, paste0(here::here(), "/Sensitivity.rds"))
#Sensitivity <- readRDS(paste0(here::here(), "/Sensitivity.rds"))
```

Interaction plots: assemble data.

```
Dat <- Sensitivity$Res %>%
  mutate(N.NBR = factor(N.NBR),
         p.SST = factor(p.SST), p=factor(p, levels=c(1, 0.5, 0.366, 0.2))) %>%
  group_by(p.SST, p.RMV, N.NBR, p) %>%
  summarise(Nep = mean(Nep), Local.mating=mean(Local))
```

Plot Nep:

```
p1 <- ggplot(Dat,  aes(p.RMV, Nep)) +
  geom_line(size = 1, aes(group = p.SST, color = p.SST)) +
  geom_point(size = 2, aes(color = p.SST), shape = 16) +
  geom_point(data=Dat %>% filter(p.SST==0.131, N.NBR==6, p.RMV==0.056, p==0.366),
             size = 2.5, shape = 17, color="violetred2") +
  facet_grid(N.NBR ~ p, labeller = label_both) + theme_bw() +
  scale_y_continuous(trans='log', breaks=c(1, 10, 100, 1000)) +
  scale_x_continuous(breaks=c(0, 0.2,0.5, 1), minor_breaks = c(0.056, 0.1,0.2)) +
  ggthemes::scale_colour_colorblind()

ggsave(file="Nep.pdf", plot=p1, width=8, height = 4)
p1
```

Plot local mating:

```
p2 <- ggplot(Dat, aes(p.RMV, Local.mating)) +
  geom_line(size = 1, aes(group = p.SST, color = p.SST)) +
```

```
  geom_point(size = 2, aes(color = p.SST), shape = 16) +
  geom_point(data=Dat %>% filter(p.SST==0.131, N.NBR==6, p.RMV==0.056, p==0.366),
             size = 2.5, shape = 17, color="violetred2") +
  scale_x_continuous(breaks=c(0, 0.2, 0.5, 1), minor_breaks = c(0.056, 0.1)) +
  scale_y_continuous(breaks=c(0, 0.25, 0.5, 0.75, 1), minor_breaks = NULL) +
  facet_grid(N.NBR ~ p, labeller = label_both) + theme_bw() +
  ggthemes::scale_colour_colorblind()

ggsave(file="Local.mating.pdf", plot=p2, width=8, height = 2.2)
p2
```

6. Sensitivity analysis for territorial foraging

```
r = 2  # Select set of rates

X <- expand.grid(p.TF = c(0, 0.25, 0.5, 1),
                 N.NBR = c(6, 20),
                 p.RMV = c(0),
                 p=c(0.2, 0.366, 0.5, 1))
R <- 30
X <- X[rep(1:nrow(X),R),]
X.f <- X %>% mutate(p.TF.f = factor(p.TF),
                    N.NBR.f=factor(N.NBR), p.f=factor(p))

Res.TF <-data.frame(X, Reduce(rbind, pbmcapply::pbmclapply(1:nrow(X), function(i)
                    fx(y=0, z=1, w=X$N.NBR[i], p=X$p[i], u=X$p.TF[i], v= X$p[i]),
                    mc.cores=parallel::detectCores())))

Sensitivity.TF <- list(X=X, X.f=X.f, Res=Res.TF)

saveRDS(Sensitivity.TF, paste0(here::here(), "/Sensitivity.TF.rds"))
#Sensitivity.TF <- readRDS(paste0(here::here(), "/Sensitivity.TF.rds"))
```

Assemble data for plots:

```
Dat.TF <- Sensitivity.TF$Res %>%
  mutate(N.NBR = factor(N.NBR), p.TF = factor(p.TF)) %>%
  group_by(p.TF, N.NBR, p) %>%
  summarise(Nep = mean(Nep), Local.mating=mean(Local))
```

```
p1.TF <- ggplot(Dat.TF,  aes(p, Nep)) +
  geom_line(size = 1, aes(group = p.TF, color = p.TF)) +
  geom_point(size = 2, aes(group = p.TF, color = p.TF)) +
  facet_grid(rows=vars(N.NBR) , labeller = label_both) + theme_bw() +
  scale_y_continuous(trans='log', breaks=c(1, 10, 100, 1000), limits=c(1,1000)) +
  scale_x_continuous(breaks=c(0.2, 0.366, 0.5, 1), minor_breaks = NULL) +
  ggthemes::scale_colour_colorblind()
p1.TF
```

```
p2.TF <- ggplot(Dat.TF,  aes(p, Local.mating)) +
  geom_line(size = 1, aes(group = p.TF, color = p.TF)) +
  geom_point(size = 2, aes(group = p.TF, color = p.TF)) +
  facet_grid(rows=vars(N.NBR), labeller = label_both) + theme_bw() +
  scale_y_continuous(breaks=c(0, 0.25, 0.5, 0.75, 1), minor_breaks = NULL,
                     limits=c(0,1)) +
  scale_x_continuous(breaks=c(0.2, 0.366, 0.5, 1), minor_breaks = NULL) +
  ggthemes::scale_colour_colorblind()
p2.TF
```

```
g1 <- cowplot::plot_grid(p1.TF, p2.TF,
                         align = "v", ncol = 2)
ggsave(file="Sensitivity.TF.pdf", plot=g1, width=8, height = 2.5)
g1
```