Intronic primers reveal unexpectedly high major histocompatibility complex diversity in Antarctic fur seals - Code

J. Tebbe, M. Ottensmann & J.I. Hoffman

24th March 2022

A script providing all code used in the analyses for the paper by Hoffman *et al.* (2022).

Packages used for analysis

Necessary packages to run this script. Missing packages can be installed with `install.packages()`

```
## analyses

library(ade4) # data analysis function
library(adegenet) # handling genetic data / genind objects
library(ape) # handling phylogenetic tree data
library(Biostrings) # easily work with genetic string sets
library(EnvStats) # environmental statistics
library(genepop) # population genetic analyses
library(hierfstat) # hierarchical F-statistics
library(inbreedR) # population genetic analyses
library(phyloseq) # phyloseq objects
library(poppr) # population genetic analyses
library(vegan) # statistical tools

## data/object handling

library(tidyverse) # package collection for easy and pretty data science with R
library(patchwork) #
library(reshape2)
library(RColorBrewer)
library(magrittr) # pipe operators
library(gridExtra) # ggplot grid manipulations
library(ggpubr) # ggplot grid and plot alignment functions
library(egg) # ggplot grid and plot alignment functions
```

Generate data sets for analyses

Main data set is a .fas-File containing multiple clone sequences for several individuals of the Antaric fur seal (*Arctozephalus gazella*). Load in as a DNAStringSet.

```
genotype_info <- readDNAStringSet("data/Clones_MHC_ArGa_exon_20210319.fas")
head(genotype_info)
```

```
## DNAStringSet object of length 6:
##     width seq                                               names
## [1]   270 AGGATTTCGTGTTCCAGTTTAAG...GAGAACGACCTTGCAGCGGCGAG AGF11008-C1-I_Klo...
## [2]   270 AGGATTTCGTGTTCCAGTTTAAG...GAGAACGACCTTGCAGCGGCGAG AGF11008-C1-I_Klo...
## [3]   270 AGGATTTCGTGTTCCAGTTTAAG...GAGAACGACCTTGCAGCGGCGAG AGF11008-C1-I_Klo...
## [4]   270 AGGATTTCGTGTTCCAGTTTAAG...GAGAACGACCTTGCAGCGGCGAG AGF11008-C1-I_Klo...
## [5]   270 AGGATTTCGTGTTCCAGTTTAAG...GAGAACGACCTTGCAGCGGCGAG AGF11008-C1-I_Klo...
## [6]   270 AGGATTTCGTGTTCCAGTTTAAG...GAGAACGACCTTGCAGCGGCGAG AGF11008-C1-I_Klo...
```

Read in the respective metadata for cloned individuals.

```
metadata_df <- read.table(file   = "data/sample_list.txt",
                          header = T) %>%
  mutate(
    real_id  = factor(real_id,
                     levels = str_sort(real_id,
                                       numeric = T)),
    colony   = as.factor(colony),
    maturity = as.factor(maturity),
    family   = as.factor(family)
    ) %>%
  arrange(real_id) %>%
  arrange(colony)

head(metadata_df)
```

```
##        clone_id       real_id colony maturity family
## 1 W8552W8258mum W8552W8258mum    FWB        M      1
## 2      W8913mum      W8913mum    FWB        M      2
## 3      W8913pup      W8913pup    FWB        P      2
## 4      W8914mum      W8914mum    FWB        M      3
## 5      W8914pup      W8914pup    FWB        P      3
## 6      W8915mum      W8915mum    FWB        M      4
```

```
str(metadata_df)
```

```
## 'data.frame':    56 obs. of  5 variables:
##  $ clone_id: chr  "W8552W8258mum" "W8913mum" "W8913pup" "W8914mum" ...
##  $ real_id : Factor w/ 56 levels "W8552W8258mum",..: 1 25 26 27 28 29 30 31 32 33 ...
##  $ colony  : Factor w/ 2 levels "FWB","SSB": 1 1 1 1 1 1 1 1 1 1 ...
##  $ maturity: Factor w/ 2 levels "M","P": 1 1 2 1 2 1 2 1 1 1 ...
##  $ family  : Factor w/ 36 levels "1","2","3","4",..: 1 2 2 3 3 4 4 5 6 7 ...
```

Subset metadata to handle mother-pup pair analysis more easily

```
metadata_df_pairs <- metadata_df[
  which(
    duplicated(metadata_df$family) | duplicated(metadata_df$family, fromLast = T)
    == T)
  ,]
```

Identify the names and index positions of unique sequences (only occuring once throughout the whole sequences data set) in the .fas-file

```
# Extracts names and index positions of a StringSet of aligned DNA sequences

# raw_clones StringSet of aligned same length sequences
# allowed.unique.seq Number of filtered sequences handled as unique
# max.mismatch Allowed mismatches to identify unique sequences
```

```r
get_unique_seqs <- function(raw_clones, allowed.unique.seq = 1, max.mismatch = 0){

  if (is.vector(raw_clones) != T){
    raw_clones <- as.vector(raw_clones)
  }
  # sort all instances that only occur once
  unique_indicator <- vector(length = length(raw_clones))
  for (i in seq_along(raw_clones)) {
    # unique_indicator[i] <- sum(match(raw_clones, raw_clones[i]), na.rm = T)
    unique_indicator[i] <- sum(vcountPattern(as.character(raw_clones[i]), raw_clones,
                                             max.mismatch = max.mismatch,
                                             min.mismatch = 0)
                                 , na.rm = T)
  } #end for i

  # index of unique seqs
  index_unique_seq <- which(
    unique_indicator == allowed.unique.seq |
      unique_indicator <= allowed.unique.seq
  )

  # corresponding unique names
  names_unique_seq <- names(raw_clones[index_unique_seq])

  return(list(index_unique_seq = index_unique_seq,
              names_unique_seq = names_unique_seq))
} #end get_unique_seqs
```

Within a DNAStringSet of multiple DNA alignments of same length sequences, find unique sequences in correspondence to `get_unique_seqs` function filters and remove them from the input data. Use the updated input to create a data frame of identified alleles based on unique sequences found in multiple individuals. Alleles are named most to least abundant.

```r
# data DNAStringSet of same length sequence alignment
# allele_name Naming of alleles in data output
# rm.unique Bool whether single sequence occurences are removed

get_allele_info <- function(data, allele_name = "ArGa-DQB*", rm.unique = T){
  if (rm.unique == T) {
    unique_identifier <- get_unique_seqs(data,
                                         allowed.unique.seq = 1,
                                         max.mismatch = 0)

    # delete unique sequences from the data
    if (!purrr::is_empty(unique_identifier[[1]])) {
      genotype_fas <- data[-unique_identifier$index_unique_seq]
    }
  } else {
    genotype_fas <- data
  }


  # create data set with allele sequences
  allele_seq <- unique(genotype_fas)
  # count occurences of identified alleles in the data
  allele_count <- as.vector(
    sapply(seq_along(allele_seq), function(i)
      sum(match(genotype_fas, allele_seq[[i]]), na.rm = T))
  )

  # create data.frame where alleles will be named after
  # its decreasing frequency in the data
  # sorted by the prior allele count
  alleles <- data.frame(seq = as.vector(allele_seq),
                        counts = allele_count,
                        row.names = NULL) %>%
    arrange(., desc(counts)) %>%
    mutate(frequency = (counts/sum(counts))*100) %>%
    `rownames<-`(., sapply(seq_along(allele_count), function(i)
      paste0(allele_name,i))) %>%
    rownames_to_column("name")

  out = list(alleles = alleles,
             genotype_fas = genotype_fas)

  return(out)
} # end get_allele_info
```

Function output is a list of two data frames. Split both to use further on

```
# returning the output to a list decreases code run time
out <- get_allele_info(genotype_info)

# assignment is shorter than function calculations
alleles <- out[[1]]
genotype_fas <- out[[2]]
```

Create data frame for each clone with an allele found more than once in the whole clone data .fas-file.

```
clone_allele_df <- as.data.frame(genotype_fas) %>%
  transmute(sequence = x) %>%
  rownames_to_column(var = "clone_var")

allele_index_in_df <- as.vector(
  sapply(
    clone_allele_df$sequence,
    function(x) match(x, alleles$seq)))

clone_allele_df  %<>%
  transmute(.,
            id         = sapply(
              clone_allele_df$clone_var,
              function(x) {
                stringr::str_split(x, "-")[[1]][1] %>%
                  paste0(., collapse = "-") %>%
                  as.factor()
              }),
            clone_var  = clone_var,
            allele     = alleles$name[allele_index_in_df],
            variant_no = allele_index_in_df,
            variant_count  = alleles$counts[allele_index_in_df],
            sequence   = sequence
  ) %>%
  mutate(., allele = factor(
    allele,
    levels = str_sort(
      unique(allele),
      numeric = T))
  ) %>%
  arrange(
    ., allele
  ) %>%
  mutate(
    variant_counter = as.vector(
      unlist(
        sapply(alleles$counts,
               function(x) seq(1:x))
      ))
  ) %>%
  relocate(., sequence, .after = last_col())



# update clone_allele_df with metadata information
# create index vector where sample ids correspond to the correct
# names in the metadata data.frame
index <- match(as.character(clone_allele_df$id), metadata_df$clone_id)

# rearrange columns in clone_allele_df based on 'index'
clone_allele_df <- clone_allele_df %>%
  mutate(
    id       = metadata_df$real_id[index],
    colony   = metadata_df$colony[index],
    maturity = metadata_df$maturity[index],
    family   = metadata_df$family[index]
  ) %>%
  relocate(., sequence, .after = last_col())

clone_allele_df  %<>% arrange(., variant_no)

# clone_allele_df
# $ id              : sample individual
# $ clone_var       : clone sample
# $ allele          : allele as name
# $ variant_no      : allele as number
# $ variant_count   : allele total count
# $ variant_counter : allele counter; 1:last number of occurence per allele
# $ colony          : colony tag
# $ maturity        : age by maturity, either mother (M) or pup (P)
# $ sequence        : MHC DQB class II exon sequence of clone_var
```

`clone_allele_df` has info for clone variants for 56 sample individuals. Data frame structure includes original DNA sample, allele name, stratum data for allele names and individual meta data corresponding to `metadata_df`.

```r
str(clone_allele_df, vec.len = 0)
```

```
## 'data.frame':    793 obs. of  10 variables:
##  $ id             : Factor w/ 56 levels "W8552W8258mum",..: NULL ...
##  $ clone_var      : chr   ...
##  $ allele         : Factor w/ 30 levels "ArGa-DQB*1","ArGa-DQB*2",..: NULL ...
##  $ variant_no     : int  NULL ...
##  $ variant_count  : int  NULL ...
##  $ variant_counter: int  NULL ...
##  $ colony         : Factor w/ 2 levels "FWB","SSB": NULL ...
##  $ maturity       : Factor w/ 2 levels "M","P": NULL ...
##  $ family         : Factor w/ 36 levels "1","2","3","4",..: NULL ...
##  $ sequence       : chr   ...
```

Create a suitable data frame for a heatmap that contains allele names, sample ids and the respective number an allele occurs in sample id

```r
allele_summary <- matrix(nrow = length(unique(clone_allele_df$id)),
                         ncol = length(unique(clone_allele_df$allele))) %>%
  `rownames<-`(., as.character(unique(clone_allele_df$id))) %>%
  `colnames<-`(., as.character(
    str_sort(
      levels(
        clone_allele_df$allele),
      numeric = T)))

# fill matrix with info on which and how many alleles are found in the
# clones for each individual fur seal
for (i in seq_along(unique(clone_allele_df$id))) {
  alleles_in_id <- summary(
    clone_allele_df$allele[clone_allele_df$id == unique(clone_allele_df$id)[i]]
    )
  allele_summary[i, ] <- alleles_in_id[str_sort(names(alleles_in_id),
                                               numeric = T)]
}

# convert to data.frame and create a "tidy" version,
# ggplot and tidyverse can handle easily
allele_summary %<>%
  t() %>%
  as.data.frame() %>%
  rownames_to_column("alleles") %>%
  pivot_longer(-c(alleles),
               names_to = "sample_id",
               values_to = "counts") %>%
  mutate(., alleles = factor(
    alleles,
    levels = str_sort(
      unique(alleles),
      numeric = T)),
    sample_id = as.factor(sample_id)) %>%
  arrange(., sample_id) %>%
  arrange(., alleles)
```

```
index2 <- match(as.character(allele_summary$sample_id), metadata_df$real_id)
allele_summary <- allele_summary %>%
  mutate(colony = metadata_df$colony[index2],
         maturity = metadata_df$maturity[index2],
         family = metadata_df$family[index2],
         sample_id = factor(sample_id,
                            levels = rev(
                              levels(sample_id)))) %>%
  arrange(desc(sample_id)) %>%
  arrange(alleles)

pair_match_index <- match(allele_summary$sample_id, metadata_df_pairs$real_id)
pair_match_index <- which(is.na(pair_match_index) == T)
allele_summaryX <- allele_summary[-pair_match_index,]

allele_summaryX <- allele_summaryX[allele_summaryX$alleles %in%
                                     levels(allele_summaryX$alleles)[1:19],]
```

Create data.frame with genotype information

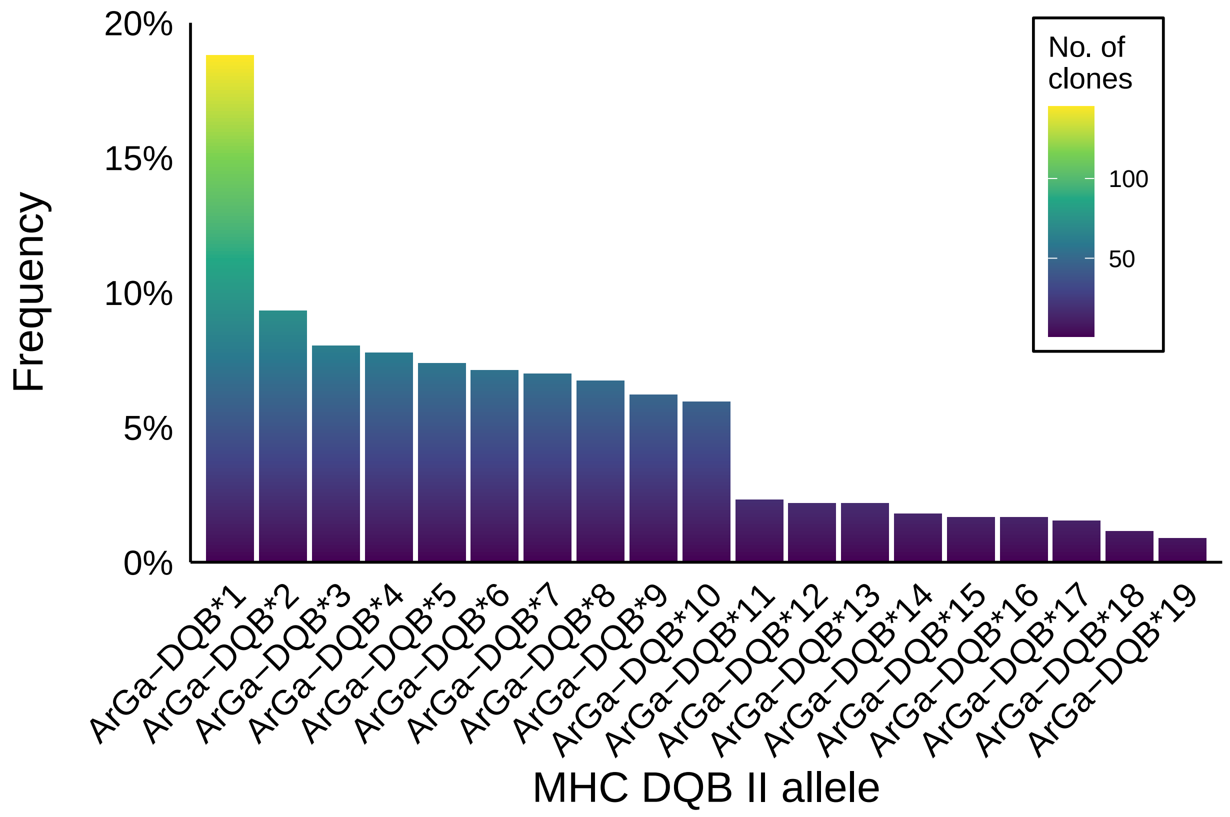
```
clone_genotype_df <- allele_summary[
  allele_summary$alleles %in% levels(allele_summary$alleles)[1:19],]

clone_genotype_df <- clone_genotype_df[which(clone_genotype_df$counts != 0),]

f1 <- function(x){
  length(
    na.omit(
      match(clone_genotype_df$alleles, x)
    ))
}

clone_genotype_df  %<>%
  mutate(.,
         variant_no = clone_allele_df$variant_no[
           match(clone_genotype_df$alleles,
                 clone_allele_df$allele)],
         freq = clone_allele_df$allele_frequency[
           match(clone_genotype_df$alleles,
                 clone_allele_df$allele)],
  ) %>%
  arrange(., alleles) %>%
  mutate(.,
         variant_counts = unlist(
           sapply(clone_genotype_df$alleles, f1))) %>%
  mutate(.,
         variant_counter = unlist(
           sapply(
             sapply(
               unique(
                 clone_genotype_df$alleles), f1),
             function(x) seq(1:x)))) %>%
  arrange(., desc(variant_counter))
```

List for figure storing

```
figures <- vector(mode = "list")
```

Plot clone sequence frequencies

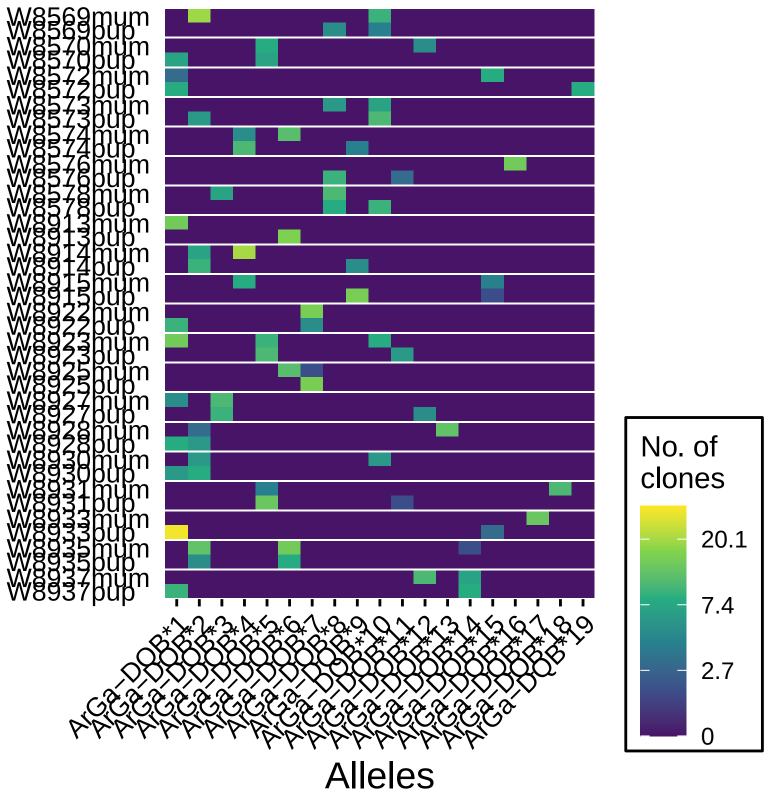
```
# index to remove putative artefacts
figures[[1]] <- ggplot(clone_allele_df[1:771,],
                       aes(x = variant_no,
                           group = dplyr::desc(variant_counter),
                           fill = variant_counter)) +
  geom_bar(aes(y = stat(count) / sum(count))) +
  scale_y_continuous(labels = scales::label_percent(accuracy = 1),
                     limits = c(0,.2),
                     expand = c(0,0)) +
  scale_fill_viridis_c(option = "viridis",
                       begin = 0,
                       end = 1) +
  ylab("Frequency\n") +
  labs(fill = "No. of\nclones") +
  scale_x_continuous(name = "MHC DQB II allele",
                     breaks = seq_along(unique(clone_allele_df$allele)),
                     labels = str_sort(unique(clone_allele_df$allele), numeric = T),
                     expand = c(0, 0.3)) +
  theme_minimal() +
  theme(panel.grid = element_line(color = "white"),
        panel.grid.minor = element_blank(),
        panel.grid.major.x = element_blank(),
        axis.line = element_line(color = "black"),
        axis.text = element_text(color = "black"),
        axis.title = element_text(color = "black",
                                  margin = margin(10,10,20,10)),
        axis.ticks = element_line(color = "black",
                                  size = 0.2),
        axis.line.x = element_line(color = "black"),
        axis.ticks.x = element_blank(),
        axis.title.x = element_text(color = "black", size = 15.5),
        axis.text.x.bottom = element_text(angle = 45,
                                          vjust = 1,
                                          hjust = 1,
                                          size = 13),
        axis.line.y = element_line(color = "black"),
        axis.title.y = element_text(size = 15.5),
        axis.ticks.y = element_blank(),
        axis.text.y.left = element_text(size = 13),
        axis.ticks.length = unit(.15,"cm"),
        plot.background = element_rect(color = "white",
                                       fill = "white"),
        legend.position = c(0.88,.7),
        legend.background = element_rect(fill = "white",
                                         color = "black"),
        plot.margin = unit(c(0.5,0.5,0.5,0,5), "cm")
  )
names(figures)[1] <- "allele_freq_hist"
```



Plot genotype heatmap of mother-pup pairs

```
# plot multiple small heatmaps for single pairs
sep_heat <- function(x, color = "blue"){
  df <- allele_summaryX[which(allele_summaryX$family == x),]
  gg <- ggplot( data = df,
                aes(x = alleles,
                    y = sample_id,
                    fill = log(counts+1))) +
    geom_tile() +
    coord_fixed(ratio = 0.6) +
    scale_fill_viridis_c(limits = c(0, 3.5),
                         begin = 0.05,
                         breaks = 0:3,
                         labels = c(0, 2.7, 7.4, 20.1)) +
    # xlab("Alleles") +
    # ylab("SSB\n") +
    labs(fill = "Log \nclone \nnumber") +
    # theme_minimal() +
    theme(
      panel.grid = element_blank(),
      panel.background = element_blank(),
      axis.text = element_text(color = "black"),
      axis.text.x.bottom = element_blank(),
      axis.text.y = element_text(size = 10,
```

```r
                                       hjust = 0),
      axis.title.x = element_blank(),
      axis.ticks.x = element_blank(),
      axis.line.y.left = element_line(color = color,
                                 size = 1),
      axis.title.y = element_blank(),
      axis.ticks.y = element_blank(),
      # axis.text.y  = element_text(color = color_ssb),
      plot.margin  = unit(c(-.1, 0, 0, 0), "cm"),
      legend.position = "none"
    )

  return(gg)

}

vec <- unique(allele_summaryX$family)
vec_ssb <- unique(allele_summaryX$family[allele_summaryX$colony == "SSB"])
vec_fwb <- unique(allele_summaryX$family[allele_summaryX$colony == "FWB"])

# create two plot lists for small heatmaps to separate the two colonies
plots_ssb <- lapply(vec_ssb, sep_heat, color = "white")
plots_fwb <- lapply(vec_fwb, sep_heat, color = "white")

# merge in order
plot_list <- c(plots_ssb, plots_fwb)

# plot heatmap for last individual with x axis
l <- length(vec)
plot_list[[l]] <- ggplot(
  data = allele_summaryX[which(allele_summaryX$family == vec[l]),],
  aes(x = alleles,
      y = sample_id,
      fill = log(counts+1))) +
  geom_tile() +
  coord_fixed(ratio = 0.6) +
  scale_fill_viridis_c(name = "No. of\nclones",
                       limits = c(0,3.5),
                       begin = 0.05,
                       breaks = 0:3,
                       labels = c(0, 2.7, 7.4, 20.1)) + #log scale
  xlab("Alleles") +
  ylab("FWB\n") +
  # labs(fill = "Log \nclone \nnumber") +
  # theme_minimal() +
  theme(
    panel.grid = element_blank(),
    panel.background = element_blank(),
    axis.ticks = element_line(color = "#000000"),
    axis.text = element_text(color = "black"),
    axis.text.y = element_text(size = 10,
                               hjust = 0),
    axis.title.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y.left = element_line(color = "white",
                                    size = 1),
    axis.text.x.bottom = element_text(angle = 45,
                                      vjust = 1,
                                      hjust = 1,
                                      size = 10),
    axis.title.x = element_text(size = 14),
    plot.margin = unit(c(-1, 0, 0, 0), "mm"),
    legend.position = "right",
    legend.background = element_rect(fill = "white",
                                     color = "black")
  )

# increase headspace for first list element
plot_list[[1]] <- plot_list[[1]] + theme(plot.margin = unit(c(10,0,0,0), "mm"))

figures[[2]] <- egg::ggarrange(plots = plot_list, ncol = 1)
```



```
names(figures)[2] <- "clone_heatmap"
```

Colony comparisons

Create `genind` object for easy data handling

Load in genetic data frames and convert to `adegenet`'s Formal `genind` class to easily handle genetic analyses in `R`.
Start with converting genotypes of Antarctic fur seal based on 41 microsatellite loci.

```r
msats_gp <- read.table(file = "data/msats_genind.txt", sep = "\t")

# `df2genind` needs input for pop argument
sites_msats <- msats_gp[,1]

# only data with genotype info can be converted to genind. Keep original data frame
msats_gen <- msats_gp[,-1]
msats_gen <- df2genind(msats_gen,
                       ploidy = 2,
                       sep = "/", # alleles are separated with "/"
                       NA.char = NA, # missing loci are NA
                       pop = sites_msats)
```

Convert MHC DQB class 2 genotypes by creating the `genind` object first.

```r
# exclude putative artefacts from main data.frame
clone_allele_df <- clone_allele_df[1:771,] %>%
  mutate(., variant_no = str_pad(variant_no, 2, pad = "0"))

# create a list of genotypes
called_clones <- vector(mode = "list", length = 1)
called_clones[[1]] <- seq_along(unlist(attributes(clone_allele_df$id)[1]))
names(called_clones[[1]]) <- unlist(attributes(clone_allele_df$id)[1])


id <- as.character(unique(clone_allele_df$id))
called_clones <- lapply(id, function(x)
  as.character(
    unique(
      clone_allele_df$variant_no[which(!is.na(match(clone_allele_df$id,x)))]
      )
    )
  ) %>%
  `names<-`(., id)

# filter out individuals that do not fit the presumed ploidy of the genotyped locus
# by deleting the least likely allele as we assume diploidy
ploidy_mismatches <- which(lapply(called_clones, length) > 2)
called_clones[ploidy_mismatches] <- lapply(called_clones[ploidy_mismatches],
                                           function(x) x[1:2])


called_clones <- lapply(called_clones, function(x){
  c(x[1], tail(x,1))
})

# build a data frame like
#                 locusA locusB locusC
#        genotype1      11    <NA>      22
#        genotype2      11      34      22
#        genotype3      12      55      21
#        genotype4      32      15      22
# that can be coerced into a "genind"
clone_df <- lapply(called_clones, function(x)
  paste0(x, collapse = "/")) %>%
  as.data.frame(.) %>%
  t(.)


# build data frame with additional info for strata in genind class object
n <- rownames(clone_df)
ind_n <- match(n,clone_genotype_df$sample_id)
strata_df <- data.frame(
  id   = n,
  pops = clone_genotype_df$colony[ind_n],
  mtry = clone_genotype_df$maturity[ind_n],
  fmly = clone_genotype_df$family[ind_n])


# coerce to genind
clone_gen <- df2genind(clone_df,
                          ploidy = 2,
                          sep = "/",
                          pop = strata_df$pops,
                          strata = strata_df)

# Already saved.
# save.df <- genind2df(clone_gen, sep = "/")
# write.table(save.df, file = "data/clone_gen.txt", sep = "\t")
```

We can use the resulting object as is and save it as a data frame that is easily convertible to a `genind` for future use.
Load in MHC DQB class 2 genotype by data frame

```
clone_gp <- read.table(file = "data/clone_genind.txt", sep = "\t")

# `df2genind` needs input for pop argument
sites_clones <- clone_gp[,1]

# 1 column is forced to a vector but must not be a vector to be coerced to
# genind object
clone_gen <- clone_gp[,-1] %>% as.data.frame() %>%
  # keep row and column names
  `rownames<-`(rownames(clone_gp)) %>%
  `colnames<-`("dqbII")

clone_gen <- df2genind(clone_gen,
                       ploidy = 2,
                       sep = "/", # separate alleles with "/"
                       pop = sites_clones)
```

Calculate Fst

Calculate fixation index after Weir & Cockerham (1984)

```r
(fst_msats_gen <- genet.dist(msats_gen, method = "WC"))
```

```
##             FWB
## SSB 0.002807775
```

```r
(fst_clone_gen <- genet.dist(clone_gen, method = "WC"))
```

```
##             FWB
## SSB 0.005438056
```

Permute to get *p*-values

Permutations are inefficient, thus lowered for demonstration. To get similar results as depicted in the publication, run

```
set.seed(111)
perm.fst(msats_gen, nperm = 9999)
perm.fst(clone_gen, nperm = 9999)
```

```r
# `data` Formal class genind to be permuted for Fst
# `nperm` number of Fst permutations
# `resamples` number of data permutations

perm.fst <- function(data, nperm = 100, resamples = 50){
  x <- data
  df <- genind2df(x, sep = "/")
  resamples <- resamples

  # calculate pairwise fst
  mat.obs <- genet.dist(x, method = "WC") %>% as.matrix()
  mat.obs <- mat.obs[1,2]
  cat("Fst: ", mat.obs, "\n")

  # calculate permute fst
  nperm <- nperm
  mat.perm <- sapply(1:nperm,
                     function(i) {
    # permute
    x <- df[sample(1:nrow(df), resamples), ]
    x <- df2genind(as.matrix(x[,-1]), sep = "/", pop = x[,1])

    # calculate pairwise fst
    mat.fst <- genet.dist(x, method = "WC") %>% as.matrix()
    mat.fst <- mat.fst[1,2]
  })

  # handle as randtest for monte-carlo like simulation of p-values
  test.rand <- as.randtest(
    sim = na.omit(sapply(1:nperm,
                         function(i) mat.perm[i])),
    obs = mat.obs,
    alter="greater" # ((# of permutations >= mat.obs) + 1) / (# of nperm +1)
    )
  return(test.rand)

} # end perm.fst

perm.fst(msats_gen)
```

```
## Fst:  0.002807775
```

```
## Monte-Carlo test
## Call: as.randtest(sim = na.omit(sapply(1:nperm, function(i) mat.perm[i])),
##     obs = mat.obs, alter = "greater")
##
## Observation: 0.002807775
##
## Based on 100 replicates
## Simulated p-value: 0.4356436
## Alternative hypothesis: greater
##
##      Std.Obs  Expectation     Variance
## 3.897951e-02 2.755216e-03 1.818093e-06
```

```
perm.fst(clone_gen)
```

```
## Fst:  0.005438056
```

```
## Monte-Carlo test
## Call: as.randtest(sim = na.omit(sapply(1:nperm, function(i) mat.perm[i])),
##     obs = mat.obs, alter = "greater")
##
## Observation: 0.005438056
##
## Based on 100 replicates
## Simulated p-value: 0.5049505
## Alternative hypothesis: greater
##
##       Std.Obs   Expectation      Variance
## -9.436796e-02  5.843592e-03  1.846755e-05
```

Private alleles per site

```
# msats
private_alleles(msats_gen) %>% apply(MARGIN = 1, FUN = sum)
```

```
## FWB SSB
## 108  41
```

```
# mhc clones
private_alleles(clone_gen) %>% apply(MARGIN = 1, FUN = sum)
```

```
## FWB SSB
##   5   3
```

Allelic richness per site

```
# msats
allelic.richness(genind2hierfstat(msats_gen))$Ar %>%
  apply(MARGIN = 2, FUN = mean) %>%
  round(digits = 3)
```

```
##   FWB   SSB
## 7.959 7.819
```

```
# mhc clones
allelic.richness(genind2hierfstat(clone_gen))$Ar %>%
  apply(MARGIN = 2, FUN = mean) %>%
  round(digits = 3)
```

```
##    FWB    SSB
## 15.436 16.000
```

Summary of basic statistics per site per locus

```
# msats
msats_gen_stats <- basic.stats(msats_gen, diploid = TRUE)

# mhc clones
clone_gen_stats <- basic.stats(clone_gen, diploid = TRUE)
```

Mean observed heterozygosity per site

```
# msats
apply(msats_gen_stats$Ho, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##       FWB       SSB
## 0.7921805 0.7819756
```

```
# mhc clone
apply(clone_gen_stats$Ho, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.7576 0.8261
```

Mean expected heterozygosity per site

```
# msats
apply(msats_gen_stats$Hs, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##       FWB       SSB
## 0.7860390 0.7795317
```

```
# mhc clones
apply(clone_gen_stats$Hs, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.9233 0.9298
```

Inbreeding coefficient FIS

```
# msats
apply(msats_gen_stats$Fis, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##          FWB          SSB
## -0.012095122 -0.006690244
```

```
# mhc clones
apply(clone_gen_stats$Fis, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.1795 0.1116
```

Repeat analysis. Run mothers and pups separate

```

df.sats <- genind2df(msats_gen, sep = "/")
df.sats <- df.sats[match(metadata_df$real_id, rownames(df.sats)),]

df.mom <- df.sats[which(metadata_df$maturity =="M"),]
df.mom <- df2genind(df.mom[,-1], sep = "/", pop = df.mom[,1])

df.pup <- df.sats[which(metadata_df$maturity =="P"),]
df.pup <- df2genind(df.pup[,-1], sep = "/", pop = df.pup[,1])
```

Microsatellite data: Separate mother and pup data

```

private_alleles(df.mom) %>% apply(MARGIN = 1, FUN = sum)
```

Microsatellite data: Mothers

```
## FWB SSB
## 149  34
```

```
# allelic richness per site (per locus)
allelic.richness(genind2hierfstat(df.mom))$Ar %>%
  apply(MARGIN = 2, FUN = mean) %>%
  round(digits = 3)
```

```
##   FWB   SSB
## 6.737 6.474
```

```
# all kinds of basic stats
df.mom_stats <- basic.stats(df.mom, diploid = TRUE)

# mean observed heterozygosity per site
apply(df.mom_stats$Ho, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##       FWB       SSB
## 0.7708488 0.7887829
```

```
# mean expected heterozygosity per site
apply(df.mom_stats$Hs, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##       FWB       SSB
## 0.7904512 0.7895707
```

```
# inbreeding coefficient F_IS
apply(df.mom_stats$Fis, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##          FWB          SSB
##  0.016960976 -0.009217073
```

```
# pairwise F_st
genet.dist(df.mom, method = "WC")
```

```
##              FWB
## SSB -0.008140639
```

```

# private alleles per site (per locus)
private_alleles(df.pup) %>% apply(MARGIN = 1, FUN = sum)
```

Microsatellite data: Pups

```
## FWB SSB
## 102  77
```

```
# allelic richness per site (per locus)
allelic.richness(genind2hierfstat(df.pup))$Ar %>%
  apply(MARGIN = 2, FUN = mean) %>%
  round(digits = 3)
```

```
##   FWB   SSB
## 6.397 6.414
```

```
# all kinds of basic stats
df.pup_stats <- basic.stats(df.pup, diploid = TRUE)

# mean observed heterozygosity per site
apply(df.pup_stats$Ho, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##       FWB       SSB
## 0.8251146 0.7746878
```

```
# mean expected heterozygosity per site
apply(df.pup_stats$Hs, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##       FWB       SSB
## 0.7915415 0.7805878
```

```
# inbreeding coefficient F_IS
apply(df.pup_stats$Fis, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##         FWB         SSB
## -0.04271220  0.01067805
```

```
# pairwise F_st
genet.dist(df.pup, method = "WC")
```

```
##             FWB
## SSB 0.004752703
```

```

df.clone <- genind2df(clone_gen, sep = "/")
df.clone <- df.clone[match(metadata_df$real_id, rownames(df.clone)),]

df.mom <- df.clone[which(metadata_df$maturity =="M"),]
df.mom <- df2genind(as.matrix(df.mom[,-1]), sep = "/", pop = df.mom[,1])

df.pup <- df.clone[which(metadata_df$maturity =="P"),]
df.pup <- df2genind(as.matrix(df.pup[,-1]), sep = "/", pop = df.pup[,1])
```

MHC DQB class II data: Separate mother and pup data

```
# private alleles per site (per locus)
private_alleles(df.mom) %>% apply(MARGIN = 1, FUN = sum)
```

MHC DQB class II data: Mothers

```
## FWB SSB
##   6   2
```

```
# allelic richness per site (per locus)
allelic.richness(genind2hierfstat(df.mom))$Ar %>%
  apply(MARGIN = 2, FUN = mean) %>%
  round(digits = 3)
```

```
##    FWB    SSB
## 13.155 14.000
```

```
# all kinds of basic stats
df.mom_stats <- basic.stats(df.mom, diploid = TRUE)

# mean observed heterozygosity per site
apply(df.mom_stats$Ho, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.7000 0.8333
```

```
# mean expected heterozygosity per site
apply(df.mom_stats$Hs, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.9382 0.9470
```

```
# inbreeding coefficient F_IS
apply(df.mom_stats$Fis, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.2539 0.1200
```

```
# pairwise F_st
genet.dist(df.mom, method = "WC")
```

```
##             FWB
## SSB -0.02125225
```

```
# private alleles per site (per locus)
private_alleles(df.pup) %>% apply(MARGIN = 1, FUN = sum)
```

MHC DQB class II data: Pups

```
## FWB SSB
##   7  12
```

```r
# allelic richness per site (per locus)
allelic.richness(genind2hierfstat(df.pup))$Ar %>%
  apply(MARGIN = 2, FUN = mean) %>%
  round(digits = 3)
```

```
##    FWB    SSB
## 10.461 11.000
```

```r
# all kinds of basic stats
df.pup_stats <- basic.stats(df.pup, diploid = TRUE)

# mean observed heterozygosity per site
apply(df.pup_stats$Ho, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.8462 0.8182
```

```r
# mean expected heterozygosity per site
apply(df.pup_stats$Hs, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.9231 0.9182
```

```r
# inbreeding coefficient F_IS
apply(df.pup_stats$Fis, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
##    FWB    SSB
## 0.0833 0.1089
```

```r
# pairwise F_st
genet.dist(df.pup, method = "WC")
```

```
##            FWB
## SSB 0.02847633
```

Correlate genetic diversity

Correlate measurements for genetic diversity of 41 microsatellite loci and one mhc dqb II locus

Generate genetic diversity estimates

For this study, we base genetic diversity on pairwise genetic distance measurements

```r

msats_gen.abs <- poppr::diss.dist(msats_gen) %>%
  # create distance matrix
  as.matrix()
```

Absolute genetic distance: Number of allelic differences in Microsatellite data

```r

# remove putative artefacts from clone info table
clone_allele_df <- clone_allele_df[1:771,]

# create empty matrix
# row x col = samples x alleles
allele_mat <- matrix(nrow = length(unique(clone_allele_df$id)),
                     ncol = length(unique(clone_allele_df$allele))) %>%
  `rownames<-`(., as.character(unique(clone_allele_df$id))) %>%
  `colnames<-`(., as.character(
    str_sort(
      levels(
        clone_allele_df$allele)[1:19],
      numeric = T)))

# fill matrix with info on which and how many alleles are found
# in the clones for each individual fur seal
for (i in seq_along(unique(clone_allele_df$id))) {
  alleles_in_id <- summary(
    clone_allele_df$allele[clone_allele_df$id == unique(
      clone_allele_df$id)[i]])[1:19]

  allele_mat[i, ] <- alleles_in_id[str_sort(
    names(alleles_in_id),
    numeric = T)]
}

# otu_table handles taxa as rows. This case alleles are "taxa" in phylo tree
allele_mat <- ifelse(allele_mat != 0, 1, 0) %>%
  t()

# with artefacts, there are 30 allele levels.
# Make sure, to only keep putative alleles for analysis
allele_mat <- allele_mat[1:19,]


phyloseq_tree <- ape::read.tree("data/unifrac_tree_p.nwk")
# plot tree is interested
# plot(phyloseq_tree)

# create otu table for phyloseq object
arga_phylseq <- otu_table(allele_mat, taxa_are_rows = T)

# merge otu table and phyloseq tree
arga_phylseq <- merge_phyloseq(arga_phylseq, phyloseq_tree)

# create unifrac based on phyloseq
clone_gen.ufrac <- UniFrac(arga_phylseq, weighted = F) %>%
  # convert to distance matrix
  as.matrix()
```

UniFrac distance for MHC DQB II data

Correlate distance matrices Correlate absolute allelic distances with UniFrac distances by performing a Mantel test.

```r
# keep results consistent
set.seed(111)

(mantel.corr <- vegan::mantel(clone_gen.ufrac,
                              msats_gen.abs,
                              permutations = 9999)) # 4 digit CI
```

```
##
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## vegan::mantel(xdis = clone_gen.ufrac, ydis = msats_gen.abs, permutations = 9999)
##
## Mantel statistic r: 0.1016
##       Significance: 0.0015
##
## Upper quantiles of permutations (null model):
##    90%    95%  97.5%    99%
## 0.0464 0.0579 0.0683 0.0816
## Permutation: free
## Number of permutations: 9999
```

Plot genetic distance correlation Vectorize distance matrices

```
msats_gen.abs[upper.tri(msats_gen.abs, diag = T)] <- NA
a <- msats_gen.abs %>% as.vector() %>% na.omit()

clone_gen.ufrac[upper.tri(clone_gen.ufrac)] <- NA
diag(clone_gen.ufrac) <- NA
b <- clone_gen.ufrac %>% as.vector() %>% na.omit()

df <- cbind(a,b) %>% as.data.frame()
```

Generate linear model

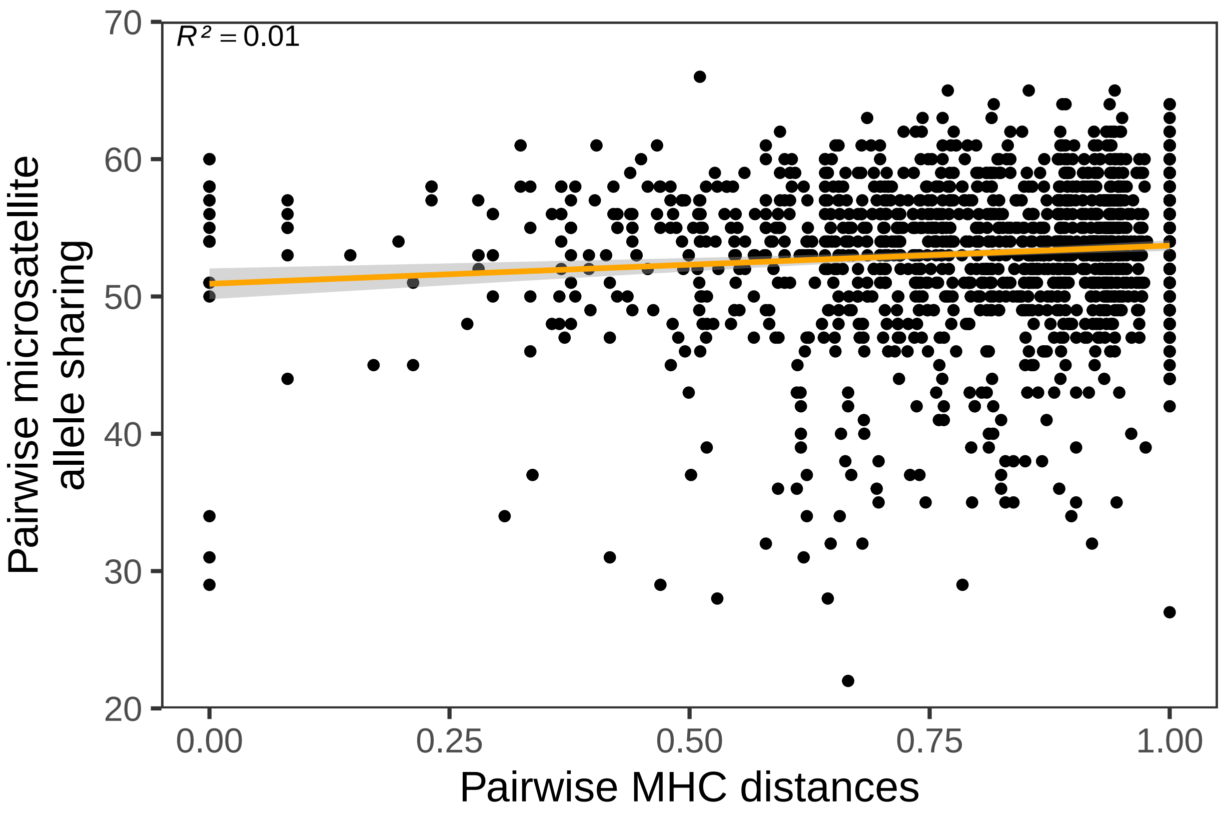
```
model <- lm(b ~ a)
m<- summary(model)
m
```

```
##
## Call:
## lm(formula = b ~ a)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.82289 -0.09476  0.06122  0.14216  0.29955
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.6002743  0.0494968  12.128  < 2e-16 ***
## a           0.0037103  0.0009265   4.005 6.51e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1988 on 1538 degrees of freedom
## Multiple R-squared:  0.01032,    Adjusted R-squared:  0.009676
## F-statistic: 16.04 on 1 and 1538 DF,  p-value: 6.51e-05
```

Plot correlation of distance matrices

```
(
  figures[[3]] <- ggplot(df, aes(b,a)) +
  geom_point() +
    # geom_jitter(height = 0.002, width = 0.07) +
  geom_smooth(method = "lm",
              se = T,
              color = "orange") +
    scale_y_continuous(name   = "Pairwise microsatellite\nallele sharing",
                       limits = c(20, 70),
                       expand = c(0,0)) +
    scale_x_continuous(name   = "Pairwise MHC distances") +
    annotate(geom = "text", x = 0.03, y = 69,
             label ="italic(R²)==0.01", parse = T) +
    theme_bw(base_size = 16) +
    theme(
      panel.grid.minor = element_blank(),
      panel.grid.major = element_blank(),
      plot.margin = unit(c(0.5,0.5,0.5,0,5), "cm"),
      axis.title = element_text(color = "black",
                                margin = margin(10,10,20,10))
    )
)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
names(figures)[3] <- "mantel_plot"
```

correlate Microsatellite and MHC DQB II heterozygosity

Calculate standardized multilocus heterozygosity in `inbreedR`

```
df <- read.table("data/msats/msats_genotypes_inbreedR.txt", sep = "\t") %>%
  # convert to inbreedR format
  convert_raw()

# check table
check_data(df)
```

```
## [1] TRUE
```

```
# standardized multi loc het
sMLH_res <- sMLH(df)

# histogram for interested
# hist(sMLH_res)
```

Convert mhc data into categorical hom/het values

```
clones_het <- read.table(file = "data/clone_mhc_het.txt", sep = "\t")

# make rownames consistent
n <- names(sMLH_res)
n_c <- rownames(clones_het)
n_in <- match(n, n_c)
clones_het <- clones_het[n_in,]
```

Create data frame to ease modelling and plotting

```
corr_het <- cbind(sMLH_res, clones_het$het) %>%
  `colnames<-`(c("smlh","mhc_het")) %>%
  as.data.frame()
```

Create glm

```
# glm with binomially distributed data
het_glm <- glm(cbind(corr_het$mhc_het, 1-corr_het$mhc_het) ~ corr_het$smlh,
               family = "binomial")

summary(het_glm)
```

```
##
## Call:
## glm(formula = cbind(corr_het$mhc_het, 1 - corr_het$mhc_het) ~
##     corr_het$smlh, family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0457   0.4649   0.6023   0.7604   1.0006
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)       6.626      3.940   1.682   0.0927 .
## corr_het$smlh   -5.264      3.836  -1.372   0.1700
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 58.193  on 55  degrees of freedom
## Residual deviance: 56.172  on 54  degrees of freedom
## AIC: 60.172
##
## Number of Fisher Scoring iterations: 4
```

```
anova(het_glm, test = "Chisq") # test glm; chi square due to binomial data
```

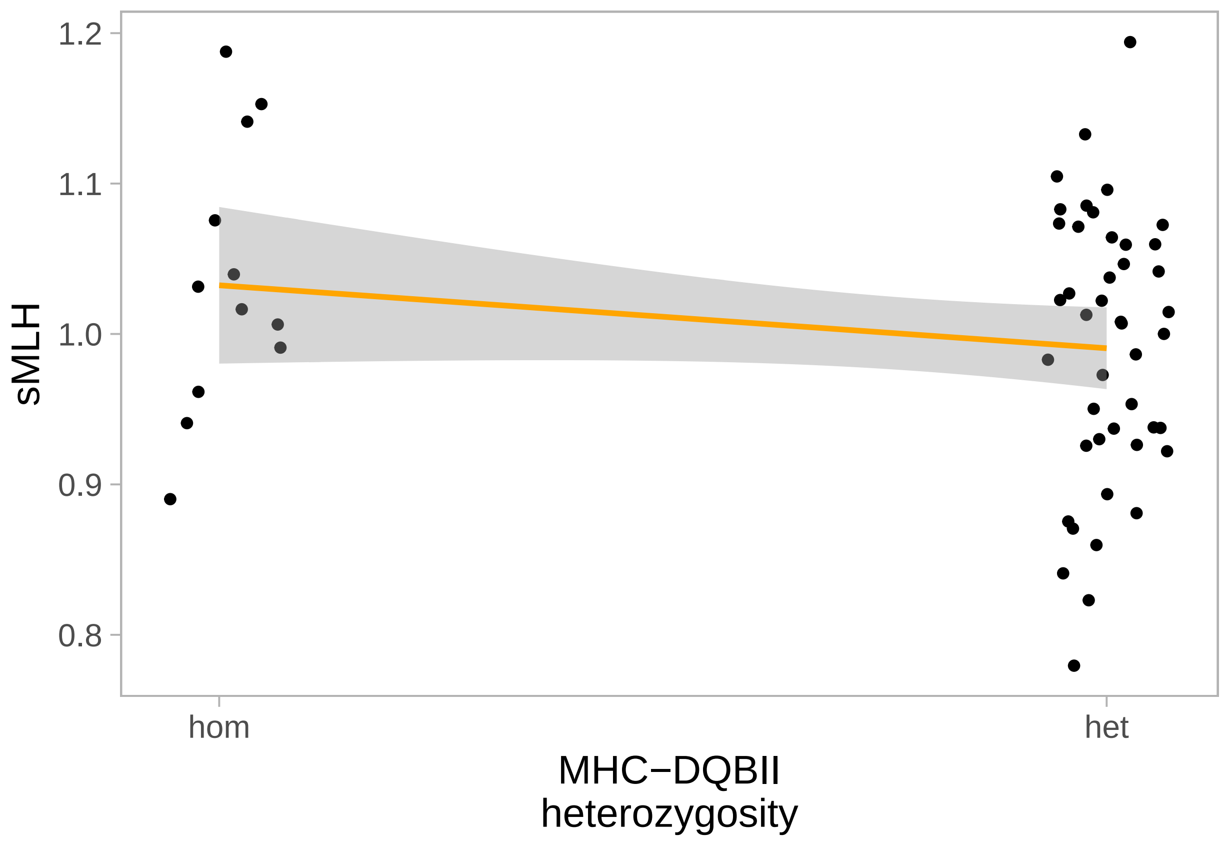
```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: cbind(corr_het$mhc_het, 1 - corr_het$mhc_het)
##
## Terms added sequentially (first to last)
##
##
##                Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## NULL                              55     58.193
## corr_het$smlh  1   2.0212        54     56.172   0.1551
```

```
chi_glm <- qchisq(1-0.1551, df = 54, lower.tail = T)
```

Plot heterozygosity correlation

```
figures[[4]] <- ggplot(data = corr_het,
                      aes(y = smlh,
                          x = mhc_het)) +
  geom_jitter(height = 0.02,
              width = 0.07) +
  geom_smooth(method = "glm",
              color = "orange",
              alpha = 0.4) +
  ylab("sMLH") +
  scale_x_continuous(name = "MHC-DQBII\nheterozygosity",
                     breaks = c(0,1),
                     labels = c("hom", "het")) +
  theme_light(base_size = 15) +
  theme(
    panel.grid = element_blank()
  )

names(figures)[4] <- "het_corr_plot"

figures[4]
```

```
## $het_corr_plot

## `geom_smooth()` using formula 'y ~ x'
```



Allele detection curves and Hamming mismatches

Define functions

Calculate Hamming distances, simulate allele detection

```r
## calculate pairwise difference to primer sequences.
## Optional, account for variable alignment length
##
Hamming.dist <- function(seq, ref, method = c("rel", "abs")) {
  method <- match.arg(method)
  # discard gaps and binding N
  gaps_seq <- which(seq %in% c("-", "N"))
  gaps_ref <- which(ref %in% c("-", "N"))
  gaps <- unique(c(gaps_seq, gaps_ref))

  seqx <- seq[-gaps]
  refx <- ref[-gaps]

  # estimate diff
  diff <- 0
  for (i in 1:length(seqx))  diff <- diff + ifelse(seqx[i] == refx[i], 0, 1)
  # correct for sequence length
  if (method == "rel") {
    diff <-
      ifelse(length(diff) > 0,diff/length(seqx), NA)
  }
  return(diff)
}# end Hamming.dist

## Pick alleles based on hamming value threshold
simulate_hoelzel <- function(data, n = 1:length(data), bs = 999,
                             hamming = hamming_values, mismatch = 1) {

  hamming <- subset(hamming, x <= mismatch)
  x <- rep(n, each = bs)
  y <- lapply(x, function(temp) {
    # sample genotypes
    get <- data[sample(x = 1:length(data),
                       size = temp,
                       replace = T)] %>%
      unlist() %>%
      unique()
    # keep alleles with < mismatch differences
    keep <- get[get %in% rownames(hamming)] %>%
      length()
  })

  df <- data.frame(x = x,y = unlist(y))
  df$x <- as.factor(df$x)
  return(df)
}# end simulate_hoelzel

#' @description Summarizes data
#' @param data a data frame
#' @param measurevar character giving column name of data to summarise
#' @param groupvars character giving column names of grouping variables
#' @param na.rm boolean
#' @param conf.interval confidence interval (default 0.95)
#' @param .drop boolean
#'
#' @source
#' Taken from the R cookbook (cookbook-r.com/Manipulating_data/Summarizing_data/)
#'
summary_stats <- function(data = NULL,
                          measurevar = NULL,
                          groupvars = NULL,
                          na.rm = TRUE,
                          conf.interval = 0.95,
                          .drop = TRUE) {

  length2 <- function(x, na.rm = FALSE) {
    if (na.rm) {
      sum(!is.na(x))
    } else {
      length(x)
    }
  }
  # This does the summary. For each group's data frame, return a vector with
  # N, mean, and sd
  datac <- plyr::ddply(data, groupvars, .drop = .drop,
                       .fun = function(xx, col) {
                         c(N = length2(xx[[col]], na.rm = na.rm),
                           mean = mean(xx[[col]], na.rm = na.rm),
                           sd = sd(xx[[col]], na.rm = na.rm)
                         )
                       },
                       measurevar
  )

  # Rename the "mean" column
  datac <- plyr::rename(datac, c("mean" = measurevar))

  datac$se <- datac$sd / sqrt(datac$N)  # Calculate standard error of the mean

  # Confidence interval multiplier for standard error
  # Calculate t-statistic for confidence interval:
  # e.g., if conf.interval is .95, use .975 (above/below), and use df=N-1
  ciMult <- qt(conf.interval/2 + .5, datac$N - 1)
  datac$ci <- datac$se * ciMult

  return(datac)
}
```

Generate data for nucleotide mistmaches at PBR

```r
## Putative alleles Cloning sequences (full exon, 267bp)
Clones <- ape::read.dna("data/ArGa_DQB-Hoelzel-primer-clones_20211027.fas",
                         format = "fasta") %>%
  as.character() %>%
  apply(.,2, toupper) %>% ## append a dummy colum
  cbind(., "-")
# ----------------------------------------------------------------------------

## Extract and remove primer from the alignment
ClonesPrimer <- Clones[1,]

## remove primer from matrix
Clones <- Clones[-1,]

Clones_hd <- data.frame(x = apply(Clones, 1,
                                  Hamming.dist,
                                  ref = ClonesPrimer,
                                  method = "abs") %>%
                          unlist())

# ----------------------------------------------------------------------------
Clones_glm_df <- data.frame(mismatches = Clones_hd$x,
                            binom = 0,
                            a_counts = c(145, 72, 62, 60, 57, 55, 54, 52, 48,
                                         46, 18, 17, 17, 14, 13, 13, 12, 9, 7))
## Set alleles characterised in Hoelzel et al to 1
Clones_glm_df$binom[c(6, 17)] <- 1
```

Plot mismatches

```r
set.seed(98)
figures[[5]] <- ggplot(Clones_glm_df,
                       aes(x = as.factor(binom),
                           y = mismatches,
                           fill = as.factor(binom))) +
    geom_boxplot(alpha = 0.9,
                 fatten = 3,
                 outlier.shape = NA ) +
    geom_jitter(aes(size = a_counts),
                shape = 21,
                alpha = 0.9,
                width = 0.4,
                height = 0.05,
                color = "black",
                fill = "grey") +
    scale_size(range = c(3,7)) +
    theme_classic(base_size = 16,
                  base_line_size = 1,
                  base_rect_size = 1) +
    scale_x_discrete(name = "Allele detected in both studies",
                     labels = c("No", "Yes")) +
    ylab("Mismatches at primer binding site") +
    labs(tag = "A") +
    scale_fill_manual(values = c("#FDE725FF", "#481567FF")) +
    theme(axis.ticks = element_line(color = "black"),
          axis.line = element_line(color = "black"),
          axis.text = element_text(color = "black"),
          legend.position = "none")

names(figures)[5] <- "hamming_boxplot"
```

Allele detection on simulated primer-mismatches

```r
load("data/called_clones-20211027.RData")
clone_genotypes <- called_clones

## Simulate datasaets
clone_simul <- lapply(0:max(Clones_hd), function(x) {
  simulate_hoelzel(data = clone_genotypes[["clone_exon"]],
                   bs = 99,
                   hamming = Clones_hd,
                   mismatch = x)
})

for (i in 1:length(clone_simul)) {
  clone_simul[[i]]$mismatches <- as.character(i - 1)
}

clone_simul <- do.call("rbind", clone_simul)
clone_simul$x <- as.numeric(as.character(clone_simul$x))
clone_summary <- summary_stats(clone_simul,
                                measurevar = "y",
                                groupvars = c("x","mismatches"),
                                conf.interval = 0.99)
## add number of Hoelzel et al., 1999
clone_summary[nrow(clone_summary) + 1, ] <- c(13, 99, 0, 4, 0, 0, 0)
clone_summary$x <- as.numeric(as.character(clone_summary$x))
```

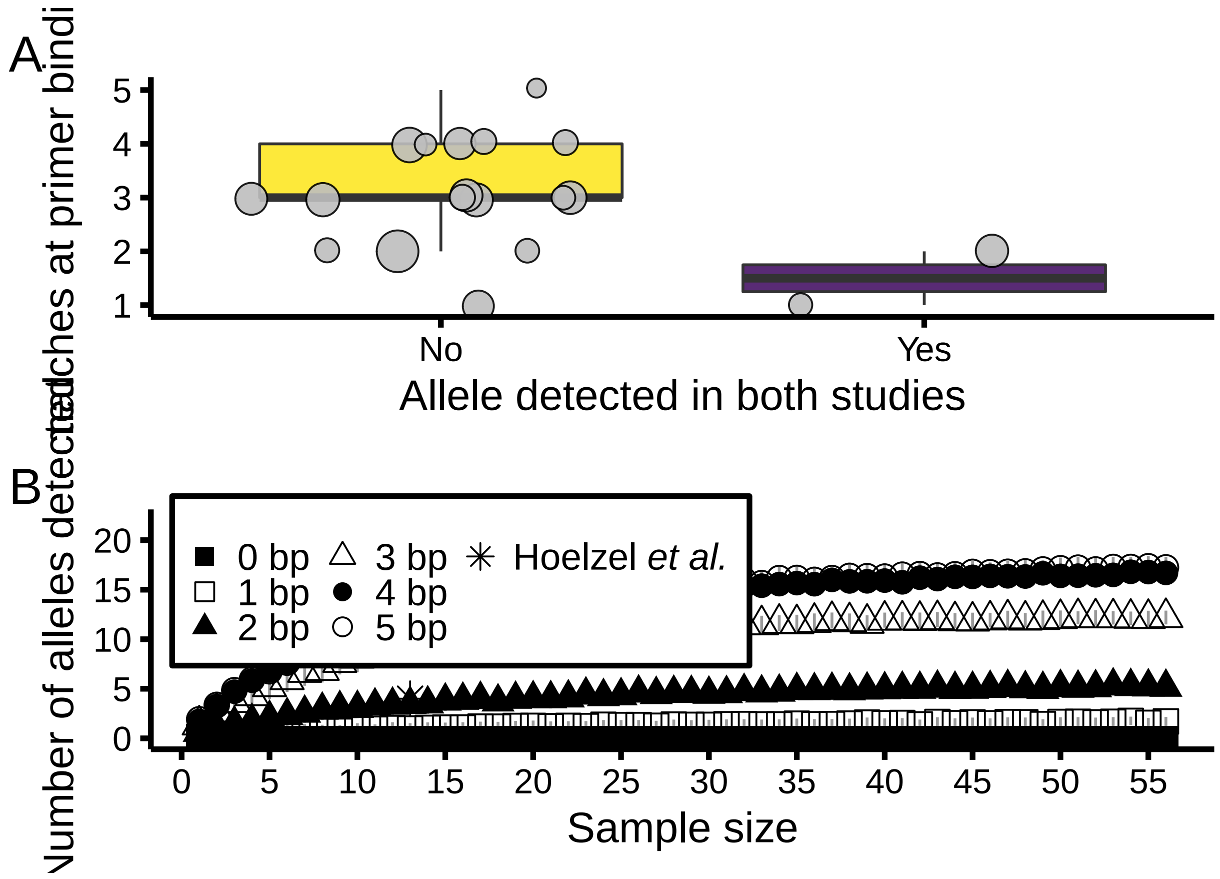
Allele detection curves

Plot size does not match size of publication figure

```
hoelzel.exp <- c(expression("Hoelzel " *italic("et al.")))
figures[[6]] <- ggplot(clone_summary, aes(x,y)) +
  geom_linerange(ymin = clone_summary$y - clone_summary$sd,
                 ymax = clone_summary$y + clone_summary$sd,
                 col = "grey0",
                 alpha = 0.4) +
  geom_point(aes(shape = mismatches),
             size = 4,
             fill = "black") +
  xlab("Sample size") +
  ylab("Number of alleles detected") +
  scale_x_continuous(breaks = seq(0,60,5)) +
  scale_y_continuous(breaks = seq(0,20,5),
                     limits = c(0,22)) +
  labs(tag = "B") +
  scale_shape_manual(labels = c("0 bp", "1 bp", "2 bp",
                                "3 bp", "4 bp", "5 bp",
                                hoelzel.exp),
                     breaks = c(0, 1, 2, 3, 4, 5, 99),
                     values = c(15, 0, 17, 2, 16, 1, 8)) +
  theme_classic(base_size = 16,
                base_line_size = 1,
                base_rect_size = 1) +
  theme(axis.ticks = element_line(color = "black"),
        axis.line = element_line(color = "black"),
        axis.text = element_text(color = "black"),
        legend.title = element_blank(),
        legend.background = element_rect(linetype = 1,
                                         color = "black"),
        legend.position = c(.0,1.0),
        legend.box.margin = margin(-5,0,0,8, "pt"),
        legend.justification = c("left", "top")) +
  guides(shape = guide_legend(ncol = 3,
                              label.hjust = 0))

addSmallLegend <- function(myPlot, pointSize = 0.5, textSize = 3, spaceLegend = 0.1) {
  myPlot +
    guides(shape = guide_legend(override.aes = list(size = pointSize),
                                ncol = 3,
                                label.hjust = 0),
           color = guide_legend(override.aes = list(size = pointSize))) +
    theme(legend.title = element_blank(),
          legend.text  = element_text(size = textSize),
          legend.key.size = unit(spaceLegend, "lines"))
}

figures[[6]] <- addSmallLegend(figures[[6]], pointSize = 3, textSize = 14)

names(figures)[6] <- "allele_detection_curve"
```

Plot as panel

```
ggpubr::ggarrange(figures[[5]], figures[[6]], nrow = 2, ncol = 1, align = "v")
```



Session information

```
sessionInfo()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19043)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=German_Germany.1252  LC_CTYPE=German_Germany.1252
## [3] LC_MONETARY=German_Germany.1252 LC_NUMERIC=C
## [5] LC_TIME=German_Germany.1252
```

```
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] egg_0.4.5           ggpubr_0.4.0        gridExtra_2.3
##  [4] magrittr_2.0.2      RColorBrewer_1.1-2  reshape2_1.4.4
##  [7] patchwork_1.1.1     forcats_0.5.1       stringr_1.4.0
## [10] dplyr_1.0.8         purrr_0.3.4         readr_2.1.2
## [13] tidyr_1.2.0         tibble_3.1.6        ggplot2_3.3.5
## [16] tidyverse_1.3.1     vegan_2.5-7         lattice_0.20-44
## [19] permute_0.9-7       poppr_2.9.3         phyloseq_1.36.0
## [22] inbreedR_0.3.3      hierfstat_0.5-10    genepop_1.1.7
## [25] EnvStats_2.7.0      Biostrings_2.60.2   GenomeInfoDb_1.28.4
## [28] XVector_0.32.0      IRanges_2.26.0      S4Vectors_0.30.2
## [31] BiocGenerics_0.38.0 ape_5.6-2           adegenet_2.1.5
## [34] ade4_1.7-18
##
## loaded via a namespace (and not attached):
##  [1] colorspace_2.0-3       ggsignif_0.6.3         seqinr_4.2-8
##  [4] ellipsis_0.3.2         fs_1.5.2               rstudioapi_0.13
##  [7] farver_2.1.0           fansi_1.0.2            lubridate_1.8.0
## [10] xml2_1.3.3             codetools_0.2-18       splines_4.1.0
## [13] knitr_1.37             pegas_1.1              jsonlite_1.8.0
## [16] broom_0.7.12           cluster_2.1.2          dbplyr_2.1.1
## [19] shiny_1.7.1            compiler_4.1.0         httr_1.4.2
## [22] backports_1.4.1        assertthat_0.2.1       Matrix_1.3-3
## [25] fastmap_1.1.0          cli_3.2.0              later_1.3.0
## [28] htmltools_0.5.2        tools_4.1.0            igraph_1.2.11
## [31] gtable_0.3.0           glue_1.6.2             GenomeInfoDbData_1.2.7
## [34] Rcpp_1.0.8             carData_3.0-5          Biobase_2.52.0
## [37] cellranger_1.1.0       vctrs_0.3.8            rhdf5filters_1.4.0
## [40] multtest_2.48.0        nlme_3.1-152           iterators_1.0.14
## [43] xfun_0.30              rvest_1.0.2            mime_0.12
## [46] lifecycle_1.0.1        rstatix_0.7.0          zlibbioc_1.38.0
## [49] MASS_7.3-54            scales_1.1.1           hms_1.1.1
## [52] promises_1.2.0.1       biomformat_1.20.0      rhdf5_2.36.0
## [55] yaml_2.3.5             stringi_1.7.6          highr_0.9
## [58] foreach_1.5.2          boot_1.3-28            rlang_1.0.2
## [61] pkgconfig_2.0.3        bitops_1.0-7           polysat_1.7-6
## [64] evaluate_0.15          Rhdf5lib_1.14.2        labeling_0.4.2
## [67] cowplot_1.1.1          tidyselect_1.1.2       plyr_1.8.6
## [70] R6_2.5.1               generics_0.1.2         DBI_1.1.2
## [73] withr_2.5.0            pillar_1.7.0           haven_2.4.3
## [76] mgcv_1.8-35            abind_1.4-5            survival_3.2-11
## [79] RCurl_1.98-1.5         car_3.0-12             modelr_0.1.8
## [82] crayon_1.5.0           utf8_1.2.2             tzdb_0.2.0
## [85] rmarkdown_2.13         grid_4.1.0             readxl_1.3.1
## [88] data.table_1.14.2      reprex_2.0.1           digest_0.6.29
## [91] xtable_1.8-4           httpuv_1.6.5           munsell_0.5.0
## [94] viridisLite_0.4.0
```

References

Arora, Sonali, Martin Morgan, Marc Carlson, and H. Pagès. 2021. *GenomeInfoDb: Utilities for Manipulating Chromosome Names, Including Modifying Them to Follow a Particular Naming Style.* https://bioconductor.org/packages/GenomeInfoDb.

Auguie, Baptiste. 2017. *gridExtra: Miscellaneous Functions for "Grid" Graphics.* https://CRAN.R-project.org/package=gridExtra.

———. 2019. *Egg: Extensions for Ggplot2: Custom Geom, Custom Themes, Plot Alignment, Labelled Panels, Symmetric Scales, and Fixed Panel Size.* https://CRAN.R-project.org/package=egg.

Bache, Stefan Milton, and Hadley Wickham. 2022. *Magrittr: A Forward-Pipe Operator for r.* https://CRAN.R-project.org/package=magrittr.

Bougeard, Stéphanie, and Stéphane Dray. 2018. "Supervised Multiblock Analysis in R with the ade4 Package." *Journal of Statistical Software* 86 (1): 1–17. https://doi.org/10.18637/jss.v086.i01.

Chessel, Daniel, Anne-Béatrice Dufour, and Jean Thioulouse. 2004. "The ade4 Package – I: One-Table Methods." *R News* 4 (1): 5–10. https://cran.r-project.org/doc/Rnews/.

Dray, StÃ©phane, Anne-BÃ©atrice Dufour, Jean Thioulouse, Thibaut Jombart, Sandrine Pavoine, Jean R. Lobry, SÃ©bastien Ollier, AurÃ©lie Siberchicot, and Daniel Chessel. 2021. *Ade4: Analysis of Ecological Data: Exploratory and Euclidean Methods in Environmental Sciences.* http://pbil.univ-lyon1.fr/ADE-4/.

Dray, Stéphane, and Anne–Béatrice Dufour. 2007. "The ade4 Package: Implementing the Duality Diagram for Ecologists." *Journal of Statistical Software* 22 (4): 1–20. https://doi.org/10.18637/jss.v022.i04.

Dray, Stéphane, Anne-Béatrice Dufour, and Daniel Chessel. 2007. "The ade4 Package – II: Two-Table and K-Table Methods." *R News* 7 (2): 47–52. https://cran.r-project.org/doc/Rnews/.

Goudet, Jerome, and Thibaut Jombart. 2021. *Hierfstat: Estimation and Tests of Hierarchical f-Statistics.* https://CRAN.R-project.org/package=hierfstat.

Henry, Lionel, and Hadley Wickham. 2020. *Purrr: Functional Programming Tools.* https://CRAN.R-project.org/package=purrr.

Huber, W., Carey, V. J., Gentleman, R., Anders, et al. 2015. "Orchestrating High-Throughput Genomic Analysis with Bioconductor." *Nature Methods* 12 (2): 115–21. http://www.nature.com/nmeth/journal/v12/n2/full/nmeth.3252.html.

Jombart, T. 2008. "Adegenet: A r Package for the Multivariate Analysis of Genetic Markers." *Bioinformatics* 24: 1403–5. https://doi.org/10.1093/bioinformatics/btn129.

Jombart, T., and I. Ahmed. 2011. "Adegenet 1.3-1: New Tools for the Analysis of Genome-Wide SNP Data." *Bioinformatics*. https://doi.org/10.1093/bioinformatics/btr521.

Jombart, Thibaut, and Zhian N. Kamvar. 2021. *Adegenet: Exploratory Analysis of Genetic and Genomic Data.* https://github.com/thibautjombart/adegenet.

Kamvar, Zhian N., Jonah C. Brooks, and Niklaus J. Grünwald. 2015. "Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality." *Front. Genet.* 6 (June): 208. https://doi.org/10.3389/fgene.2015.00208.

Kamvar, Zhian N., Javier F. Tabima, Jonah C. Brooks, and David Folarin. 2021. *Poppr: Genetic Analysis of Populations with Mixed Reproduction.* https://CRAN.R-project.org/package=poppr.

Kamvar, Zhian N., Javier F. Tabima, and Niklaus J. Grünwald. 2014. "*Poppr*: An R Package for Genetic Analysis of Populations with Clonal, Partially Clonal, and/or Sexual Reproduction." *PeerJ* 2 (March): e281. https://doi.org/10.7717/peerj.281.

Kassambara, Alboukadel. 2020. *Ggpubr: Ggplot2 Based Publication Ready Plots.* https://rpkgs.datanovia.com/ggpubr/.

Lawrence, Michael, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin Morgan, and Vincent Carey. 2013. "Software for Computing and Annotating Genomic Ranges." *PLoS Computational Biology* 9. https://doi.org/10.1371/journal.pcbi.1003118.

McMurdie, Paul J., and Susan Holmes. 2013. "Phyloseq: An r Package for Reproducible Interactive Analysis and Graphics of Microbiome Census Data." *PLoS ONE* 8 (4): e61217. http://dx.plos.org/10.1371/journal.pone.0061217.

McMurdie, Paul J., Susan Holmes, with contributions from Gregory Jordan, and Scott Chamberlain. 2021. *Phyloseq: Handling and Analysis of High-Throughput Microbiome Census Data.* http://dx.plos.org/10.1371/journal.pone.0061217.

Millard, Steven P. 2013. *EnvStats: An r Package for Environmental Statistics.* New York: Springer. https://www.springer.com.

———. 2022. *EnvStats: Package for Environmental Statistics, Including US EPA Guidance.* https://github.com/alexkowa/EnvStats.

Müller, Kirill, and Hadley Wickham. 2021. *Tibble: Simple Data Frames.* https://CRAN.R-project.org/package=tibble.

Neuwirth, Erich. 2014. *RColorBrewer: ColorBrewer Palettes.* https://CRAN.R-project.org/package=RColorBrewer.

Oksanen, Jari, F. Guillaume Blanchet, Michael Friendly, Roeland Kindt, Pierre Legendre, Dan McGlinn, Peter R. Minchin, et al. 2020. *Vegan: Community Ecology Package.* https://CRAN.R-project.org/package=vegan.

Pagès, H., P. Aboyoun, R. Gentleman, and S. DebRoy. 2021. *Biostrings: Efficient Manipulation of Biological Strings.* https://bioconductor.org/packages/Biostrings.

Pagès, H., P. Aboyoun, and M. Lawrence. 2021. *IRanges: Foundation of Integer Range Manipulation in Bioconductor.* https://bioconductor.org/packages/IRanges.

Pagès, H., M. Lawrence, and P. Aboyoun. 2021. *S4Vectors: Foundation of Vector-Like and List-Like Containers in Bioconductor.* https://bioconductor.org/packages/S4Vectors.

Pagès, Hervé, and Patrick Aboyoun. 2021. *XVector: Foundation of External Vector Representation and Manipulation in Bioconductor.* https://bioconductor.org/packages/XVector.

Paradis, E., and K. Schliep. 2019. "Ape 5.0: An Environment for Modern Phylogenetics and Evolutionary Analyses in R." *Bioinformatics* 35: 526–28.

Paradis, Emmanuel, Simon Blomberg, Ben Bolker, Joseph Brown, Santiago Claramunt, Julien Claude, Hoa Sien Cuong, et al. 2022. *Ape: Analyses of Phylogenetics and Evolution.* http://ape-package.ird.fr/.

Pedersen, Thomas Lin. 2020. *Patchwork: The Composer of Plots.* https://CRAN.R-project.org/package=patchwork.

R Core Team. 2021. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rousset, François. 2008. "Genepop'007: A Complete Re-Implementation of the Genepop Software for Windows and Linux." *Molecular Ecology Resources* 8: 103–6.

———. 2020. *Genepop: Population Genetic Data Analysis Using Genepop.* https://CRAN.R-project.org/package=genepop.

Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with r.* New York: Springer. http://lmdvr.r-forge.r-project.org.

———. 2021. *Lattice: Trellis Graphics for r.* http://lattice.r-forge.r-project.org/.

Simpson, Gavin L. 2022. *Permute: Functions for Generating Restricted Permutations of Data.* https://github.com/gavinsimpson/permute.

Stoffel, Martin A., Mareike Esser, Joseph Hoffman, and Marty Kardos. 2022. *inbreedR: Analysing Inbreeding Based on Genetic Markers.* https://CRAN.R-project.org/package=inbreedR.

Stoffel, Martin A., Mareike Esser, Marty Kardos, Emily Humble, Hazel Nichols, Patrice David, and Joseph I. Hoffman. 2016. "inbreedR: An r Package for the Analysis of Inbreeding Based on Genetic Markers." *Methods in Ecology and Evolution.* https://doi.org/10.1111/2041-210X.12588.

Team, The Bioconductor Dev. 2021. *BiocGenerics: S4 Generic Functions Used in Bioconductor.* https://bioconductor.org/packages/BiocGenerics.

Thioulouse, Jean, Stéphane Dray, Anne–Béatrice Dufour, Aurélie Siberchicot, Thibaut Jombart, and Sandrine Pavoine. 2018. *Multivariate Analysis of Ecological Data with ade4.* Springer. https://doi.org/10.1007/978-1-4939-8850-1.

Wickham, Hadley. 2007. "Reshaping Data with the reshape Package." *Journal of Statistical Software* 21 (12): 1–20. http://www.jstatsoft.org/v21/i12/.

———. 2016. *Ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York. https://ggplot2.tidyverse.org.

———. 2019. *Stringr: Simple, Consistent Wrappers for Common String Operations.* https://CRAN.R-project.org/package=stringr.

———. 2020. *Reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package.* https://github.com/hadley/reshape.

———. 2021a. *Forcats: Tools for Working with Categorical Variables (Factors).* https://CRAN.R-project.org/package=forcats.

———. 2021b. *Tidyverse: Easily Install and Load the Tidyverse.* https://CRAN.R-project.org/package=tidyverse.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.* https://CRAN.R-project.org/package=ggplot2.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2022. *Dplyr: A Grammar of Data Manipulation.* https://CRAN.R-project.org/package=dplyr.

Wickham, Hadley, and Maximilian Girlich. 2022. *Tidyr: Tidy Messy Data.* https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2022. *Readr: Read Rectangular Text Data.* https://CRAN.R-project.org/package=readr.