R packages for this computer lab

Module: Plant Genetic Resources (3502-470)

Karl Schmid, University of Hohenheim

23 May 2025

Table of contents

For this computer lab we will need the R package:

- `pegas` to perform the analyses

Install and load the package[1]. It also installs the package `ape`, which is a dependency and provides several functions for the population genetic analysis of the data[2].

[1] Website for this package: Link

[2] Website for this package: Link

```
# install.packages("pegas") # run this only once on your computer
library(pegas)
```

The data set

The data set for this exercise consists of 148 genotyped individuals that were prepared in the previous exercise. As a reminder the data span a region of 3 Megabases on chromosome 1 of the maize genome. The individuals belong to teosinte (the wild ancestor of cultivated maize), maize landraces and modern maize varieites. First, load the data set that you saved as an `Rdata` objects in the first computer lab:

```
load("data/zea_snps.RData")
load("data/teosinte_acc.Rdata")
load("data/landraces_acc.Rdata")
load("data/varieties_acc.Rdata")
```

Genetic diversity

We will look at the genetic diversity within each of the three data sets.

The function `seg.sites` returns the indices (positions) segrating sites in a data frame, which can then be counted using the `length()` function.

First we calculate the number of variants, i.e. the number of segregating sites S. For a nice presentation of the output, a data frame is constructed and it is converted to a nice-looking table using the `knitr::kable()` function

```
# Calculate the number of segregating sites for each pool
teosinte_seg <- length(seg.sites(zea_snps[teosinte_acc,]))
landraces_seg <- length(seg.sites(zea_snps[landraces_acc,]))
varieties_seg <- length(seg.sites(zea_snps[varieties_acc,]))
```

```
# Create a data frame with the results
result_table <- data.frame(
  Pool = c("Teosinte", "Landrace", "Variety"),
  `Segregating sites` = c(teosinte_seg, landraces_seg, varieties_seg)
)

# Display the table nicely using knitr::kable()
knitr::kable(result_table, caption = "Segregating Sites by Pool")
```

Next, we calculate the nucleotide diversity π:

```
n_snps <- ncol(zea_snps) # get the number of SNPs in the 3 MB long region
pi_teosinte <- (nuc.div(zea_snps[teosinte_acc,])*n_snps)/3000000
pi_teosinte
pi_landraces <- (nuc.div(zea_snps[landraces_acc,])*n_snps)/3000000
pi_landraces
pi_varieties <- (nuc.div(zea_snps[varieties_acc,])*n_snps)/3000000
pi_varieties
```

The reason why we first multiply with the reported number of variants in this data set and then divide by the sequence length is that we want to have a per site estimate and the original vcf input data file did only contain the variant positions.

Tip

It is easy to compare the effect of correcting for the length of the sequenced region.
Compare the output of the command `nuc.div(zea_snps[teosinte_acc,])` with the output of 'pi_teosinte <- (nuc.div(zea_snps[teosinte_acc,])*n_snps)/3000000"
Why is the output of the first command higher than of the second command?

Please review the formula to calculate nucleotide diversity that you learned about in the lecture.

Exercise

Modify the above code to include the value of pi as another column in the table.

Question

- Which data set has the highest genetic diversity? Which one the lowest?
- Can you come up with an explanation of the differences in genetic diversity?

Sliding window analysis

The above calculation gives an average nucleotide diversity for the whole 3 megabases long region. There is, however, the expectation that there is some variation in genetic diversity within this region. We therefore use a

sliding window approach to calculate nucleotide diversity in windows of 10000 bp that slide along the chromosomes in steps of 100 bp.

Before we can start with the sliding window analysis, we have to get the position of each polymorphism. Unfortunately, the `DNAbin` object does not store the chromosome and position on the chromosome in the object, but it is encoded in the column name of the object because the `vcfR` package reads the chromosome number and position from the vcf file and then encodes it. Therefore, `S1_265004016` reflects the position 1_265004016 on chromosome 1.

We have extract from the column names of the `DNAbin` object and then extract the position and chromosome.

First, load the `stringr` library for string manipulation.

```
# install.packages("stringr") # run only once
library(stringr)
```

To be able to analyse different populations, we define a large function `analyse_sliding_window`. The function first creates windows and then iterates over them. Instead of using a `for` loop for the iteration, which is slow in R, we use the `sapply` function to serially *apply* a function to the data.

```
window_size <- 10000       # Window size: 10,000 base pairs
step_size <- 10000           # Step size: 100 base pairs
```

```
analyse_sliding_windows <- function(snp_data, window_size, step_size) {
  # Get the polymorphism names from the DNAbin object
  polymorphisms <- colnames(snp_data)

  # Use regex to extract the chromosome and position components from each name.
  # The regex "^S(\\d+)_(\\d+)$" captures two groups: chromosome (group 1) and position (group 2)
  matches <- str_match(polymorphisms, "^S(\\d+)_(\\d+)$")

  # Convert the captured values to numeric
  chromosome <- as.numeric(matches[, 2])
  position <- as.numeric(matches[, 3])

  # Get starting and end positions on the chromosome.
  # Here we assume the positions are sorted in ascending order.
  startpos <- position[1]
  endpos <- tail(position, 1)

  # Generate the starting positions for the sliding windows
  starts <- seq(startpos, endpos - window_size + 1, by = step_size)

  # Calculate the midpoint for each window (used as the position indicator)
  midpoints <- starts + floor(window_size / 2)

  # Calculate nucleotide diversity per nucleotide for each window.
  # Here we divide the nucleotide diversity (calculated by nuc.div on the window) by the window size.
  nuc_diversity <- sapply(starts, function(start) {
    # Define the end position of the window
```

```
    end <- start + window_size - 1

    # Subset polymorphisms based on chromosome 1 and the current window
    selected_polymorphisms <- polymorphisms[chromosome == 1 & position >= start & position <= end]

    # Extract the corresponding columns from the DNAbin object
    selected_snps <- snp_data[, selected_polymorphisms, drop = FALSE]

    # Calculate nucleotide diversity over the window (using nuc.div from the pegas package)
    diversity <- nuc.div(selected_snps)

    # Return the nucleotide diversity per nucleotide
    diversity / window_size
  })

  # Combine midpoints and nucleotide diversity values into a data frame
  results_df <- data.frame(
    Midpoint = midpoints,
    Nucleotide_Diversity = nuc_diversity
  )

  return(results_df)
}
```

Run the function for the complete dataset and then plot the diversity along the chromosome.

```
zea_snps.div <- analyse_sliding_windows(zea_snps, window_size, step_size)
teosinte_snps.div <- analyse_sliding_windows(zea_snps[teosinte_acc,], window_size, step_size)
landraces_snps.div <- analyse_sliding_windows(zea_snps[landraces_acc,], window_size, step_size)
```

Now plot the level of diversity along the chromosome. We use `ggplot2` for the plotting, and for this purpose, we create a dataframe with the diversity values. Since the midpoints are the same for all, we can easily combine them into a new dataframe for the plotting

```
df_plot <- data.frame(
  Midpoint = zea_snps.div$Midpoint,
  Zea_Div = zea_snps.div$Nucleotide_Diversity,
  Teosinte_Div = teosinte_snps.div$Nucleotide_Diversity,
  Landraces_Div = landraces_snps.div$Nucleotide_Diversity
)
```

Then plot the dataframe

```
library(ggplot2)
```

```
ggplot(df_plot, aes(x = Midpoint)) +
  #geom_line(aes(y = Zea_Div, color = "Zea Diversity"), size = 1) +
  geom_line(aes(y = Teosinte_Div, color = "Teosinte Diversity"), size = 1) +
  geom_line(aes(y = Landraces_Div, color = "Landraces Diversity"), size = 1) +
  labs(x = "Midpoint", y = "Nucleotide Diversity", color = "Group") +
  theme_minimal()
```

Exercise

1. Add the code for the `varieties` to plot the variation along the chromosome.
2. Which general patterns do you recognize with respect to the overall levels of diversity and the variation in genetic diversity along the chromosome in the different pools of the sample?
3. Which explanations can you come up with to explain the high variation of diversity levels along the chromosome?
4. Can you think of any biases with respect to the total levels of diversity and the levels of diversity in individual genomic regions? (Hint: Think of the effect of the filtering during data preparation or in later stages of the analysis)

1 Identification of outlier windows

The comparison of diversity levels of individual windows between pools may provide further insights.

We can use a scatterplot and a linear model to identify windows, whose level of diversity differs strongly between pools.

```
# Calculate the maximum value for the limits (ignoring NA values)
max_val <- max(c(df_plot$Teosinte_Div, df_plot$Landraces_Div), na.rm = TRUE)

ggplot(df_plot, aes(x = Teosinte_Div, y = Landraces_Div)) +
  geom_point() +
  # Add the x = y line (dashed line)
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "gray") +
  # Fit and add the linear model line without confidence interval (blue line)
  geom_smooth(method = "lm", se = TRUE, color = "blue") +
  # Set x and y limits the same based on the maximum value
  scale_x_continuous(limits = c(0, max_val)) +
  scale_y_continuous(limits = c(0, max_val)) +
  labs(
    x = "Teosinte Nucleotide Diversity",
    y = "Landraces Nucleotide Diversity",
    title = "Scatterplot of Nucleotide Diversities: Teosinte vs Landraces",
    subtitle = "Dashed line: x = y | Blue line: Linear Model Fit"
  ) +
  theme_minimal()
```

Exercise

1. Make a similar plot for the comparison of Landraces versus Varieties diversity.
2. Interprete the resulting plots. What to the lines indicate?
3. Which approach would you take to (statistically) identify and further characterize outliner windows?
4. Which process could explain the presence of the outlier windows?