

# Bioinformatic Methods for the Analysis of Immune Repertoires

Enkelejda Miho<sup>1</sup>

<sup>1</sup>Department of Biosystems Science and Engineering, ETH Zürich  
✉ enkelejda.miho@ethz.ch

April 11, 2017

## Contents

<b>I</b>	<b>Computational Tools</b>	<b>3</b>
<b>1</b>	<b>An Introduction to R (with knitr)</b>	<b>3</b>
1.1	Required Software . . . . .	3
1.2	Raw Data . . . . .	5
1.3	Literate Programming: Dynamic Reporting and Reproducibility . . . . .	6
1.4	Data-Structures . . . . .	6
1.5	Accessing Data and Summary Statistics . . . . .	10
1.6	Visualization . . . . .	11
1.7	R Functions . . . . .	15
<b>2</b>	<b>Bioinformatic Tools for Antibody Repertoire Annotation</b>	<b>16</b>
2.1	IMGT/HighV-QUEST . . . . .	16
2.2	MiXCR [1] . . . . .	17
2.3	R packages needed for the analysis . . . . .	18
<b>II</b>	<b>Preprocessing and Analysis of Antibody Repertoire NGS Data</b>	<b>19</b>
<b>3</b>	<b>Databases and data repositories</b>	<b>19</b>
3.1	Guide to data retrieval and download . . . . .	20
<b>4</b>	<b>Assembly of paired-end reads</b>	<b>22</b>
4.1	Software installation . . . . .	22
4.2	Input . . . . .	22
4.3	Output . . . . .	22
<b>5</b>	<b>Read annotation</b>	<b>23</b>
5.1	Input to IMGT . . . . .	24
5.2	Explore output IMGT . . . . .	24
<b>6</b>	<b>Filtering</b>	<b>29</b>
6.1	Filtering CDR3 singletons, short CDR3, stop codons . . . . .	29
6.2	Processing . . . . .	37

<b>7</b>	<b>Immune repertoire analysis</b>	<b>46</b>
7.1	Diversity . . . . .	46
7.2	Somatic hypermutations (SHM) . . . . .	48
7.3	CDR3 length distribution . . . . .	50
7.4	V(D)J gene and allele frequency . . . . .	51
7.5	Mining antibody repertoires: selection techniques . . . . .	52
7.6	Saving output and reproducibility . . . . .	54
<b>III</b>	<b>Advanced Topics</b>	<b>57</b>
<b>8</b>	<b>Align to database and identify sequence of interest</b>	<b>57</b>
<b>9</b>	<b>Visualization of high-dimensional data</b>	<b>57</b>
<b>10</b>	<b>The architecture of natural antibody repertoires</b>	<b>63</b>
<b>11</b>	<b>Acknowledgements</b>	<b>66</b>

## Part I

# Computational Tools

## 1 An Introduction to R (with knitr)

Several programming languages can be used for the analysis of high-throughput sequencing data of immune repertoires. In this course, I present immune repertoire analysis through R, which is an interpreted language widely used for statistical analysis and data mining.

While a comprehensive explanation of R is given on the R web site (<http://www.r-project.org>), this is an introduction to R (download, installation and data structures) for immune repertoire analysis. The extended capabilities of R through user-created packages are applied through the presentation of advanced graphical analysis (such as the `ggplot2` package developed by Hadley Wickham [2, 3]) and reporting tools (`knitr` [4]) that contributes to reproducible research as it automates final analysis reports.

### 1.1 Required Software

#### 1. Download and Install R.

R can be downloaded from the "Comprehensive R Archive Network" (CRAN) <https://cran.r-project.org/mirrors.html> by selecting a mirror. Given the location of the course, it is advised to select the USA mirror.

Select precompiled binary distributions of the base system and contributed packages for your system:

- Linux
- Mac OS X
- Windows

#### 2. Download and Install RStudio Desktop.

The integrated development environment (IDE) R Studio is available for Windows, Mac or Linux OS from <https://www.rstudio.com/products/rstudio/download/>.

#### 3. Set Working Directory, Install and Load R packages, R packages Help

- Set working directory (`setwd`) in the folder of the analysis. You can use the function `setwd()` and give the path to the folder or navigate the directories by using the tab "Files" and when in the chosen directory, clicking "More", and "Set As Working Directory". To send a line of code to your console (execute one command), use either "control + enter" (PC users) or "command + return" (Mac users).
- Install Bioconductor, `ggplot2`, and `knitr`. Start R/RStudio and type the following commands in the console > (commands indicated in grey boxes)

```
setwd("~/Desktop/PEGS_2017_EnkelejdaMiho/code")

source("http://www.bioconductor.org/biocLite.R")
biocLite()

install.packages(c("ggplot2", "knitr"),
                 dependencies = TRUE,
                 repos = "http://cran.us.r-project.org")
```

```
# Other packages used in this course:
# xtable, VennDiagram, gplots, etc
```

- Load R packages (after installing).

```
library("ggplot2")
library("knitr")
```

- Help in R: get more information on any specific named function with the `help`, `apropos`, `?`, and `??` functions.

```
apropos("mean") # Returns the names of all objects in the search

## [1] ".colMeans"      ".rowMeans"      "colMeans"      "kmeans"        "mean"
## [6] "mean_cl_boot"   "mean_cl_normal" "mean_sdl"      "mean_se"       "mean.Date"
## [11] "mean.default"   "mean.difftime"  "mean.POSIXct"  "mean.POSIXlt"  "rowMeans"
## [16] "weighted.mean"

example("mean") # Examples part of R's online help

##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50

help("mean") # Require help regarding the function "mean", equivalent to "?mean".
# Documentation on a topic with name (typically, an R object or a data set)
# can be displayed by either help("name") or ?name.

?mean # Access the documentation on a topic with name (e.g. "mean")
?plot # Access the documentation on a topic (e.g. "plot")

??mean # Search the Help System
```

```
> help("mean")
```

mean base, R Documentation

## Arithmetic Mean

### Description

Generic function for the (trimmed) arithmetic mean.

### Usage

```
mean(x, ...)
```

```
## Default S3 method: mean(x, trim = 0, na.rm = FALSE, ...)
```

### Arguments

**x** An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.

**trim** the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

**na.rm** a logical value indicating whether NA values should be stripped before the computation proceeds.

... further arguments passed to or from other methods.

### Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

### References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language. Wadsworth & Brooks/Cole.

### See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

### Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

#### 4. Download and Install MiXCR (<http://mixcr.readthedocs.io>).

MiXCR is available for Windows, Mac or Linux from <http://mixcr.readthedocs.io/en/latest/install.html>

- Check you have Java 1.8+ installed on your system (open the Terminal and type)  
\$ `java -version`  
and follow the instructions on the MiXCR website. In brief, use the following commands for the installation on Mac/Linux using Homebrew (<https://brew.sh/>)  
\$ `brew tap milaboratory/all`  
\$ `brew install mixcr`
- Download and unzip MiXCR executable in the same folder as the raw data.

## 1.2 Raw Data

The raw data analyzed in this course have been kindly provided by Jiang Zhu [5]. Refer to [5] for the experimental protocol and library preparation (based on 5'-RACE PCR). Total reads from heavy and light chains in fastq files from high-throughput sequencing of antibody repertoire of HIV-1 infected donor IAVI 17 and two healthy donors (uninfected donor 1 and 2) are indicated in the following table.

	Donor	Reads
1	HIV-1	3104454
2	Uninfected 1	3257758
3	Uninfected 2	3350792

Fastq files have been preprocessed with IMGT/MiXCR and R. The intermediary annotated files from IMGT/HighV-QUEST and clonotypes output from MiXCR are supplied with the course materials.

### 1.3 Literate Programming: Dynamic Reporting and Reproducibility

This reports constitutes also a demonstration of using `knitr`, an R package, for dynamic report generation with R, enabling integration of R code with LaTeX and supporting reproducible research. Here, I report code and analysis results of antibody repertoires from the case study. The advantage of literate programming is that when you update underlying data all figures change accordingly.

### 1.4 Data-Structures

1. **Vector.** Using `c`, the "combine" or "concatenate" function, elements are put together in a vector and assign (`<-`) them to an object (`cdr3_lengths`). Let's assume we want to analyze the mean complementarity determining region 3 (CDR3) length of ten antibodies, each of 11, 15, 22, 12, 17, 20, 19, 12, 21, 19 a.a.

The `cdr3_lengths` vector is of class i) numeric and it is unidimensional. But vectors can also be of class ii) character (letters/words), for example a set of amino acids or CDR3 a.a. sequences, iii) logical (TRUE/FALSE values), or iv) mixed.

The advantages and usability of the different classes differ: while vectors are used for unidimensional data (e.g., CDR3 length), matrices and dataframes are used for multidimensional data (dataframes with data of different classes) and lists are a form of dataframes where elements can have different lengths, thus very flexible.

```
### Put elements into a numeric vector with the c function, "combine" or "concatenate"
cdr3_lengths <- c(11, 15, 22, 12, 17, 20, 19, 12, 21, 19)

# Store the mean as its own object
mean_cdr3_lengths <- mean(cdr3_lengths) # try sum(), range(), max()

# Print the result
print(mean_cdr3_lengths)

## [1] 16.8

# set.seed makes the sampling reproducible
set.seed(1)

### Character vector of 10 random sequences
cdr3_sequences <- replicate(10,
  paste0("CAR", paste(sample(unlist(strsplit('ACDEFGHIKLMNPQRSTVWY', "")),
    sample(4:20,1)), collapse = ""), "W"))

# Returns the length of the generated CDR3s
nchar(cdr3_sequences)

## [1] 12  9 16 22 16 23 15 18 16 24

# Over-write an object
# cdr3_lengths <- nchar(cdr3_sequences)

cdr3_lengths

## [1] 11 15 22 12 17 20 19 12 21 19
```

```

nchar(cdr3_sequences)
## [1] 12 9 16 22 16 23 15 18 16 24

### Logical vector
cdr3_lengths %in% nchar(cdr3_sequences)
## [1] FALSE TRUE TRUE TRUE FALSE FALSE FALSE TRUE FALSE FALSE

# NOT: !x
cdr3_lengths[!cdr3_lengths==22] # all values excluding 22
## [1] 11 15 12 17 20 19 12 21 19

# AND: x&y
# OR: x|y
# XOR, indicates elementwise exclusive OR: xor(x,y)

# Mixed vector and missing data
print(cdr3_sequences[2])
## [1] "CARFEPHVW"

set.seed(1)
mixed_vector <- c(c(rep("A",3), "B"),
                  sample(cdr3_sequences, 4),
                  seq(3, 13, 3))

print(as.numeric(mixed_vector))
## [1] NA NA NA NA NA NA NA NA NA 3 6 9 12

mean(as.numeric(mixed_vector)) # Result "NA" because not all elements are numeric
## [1] NA

# Remove NA
mean(as.numeric(mixed_vector), na.rm = T)
## [1] 7.5

# "intersect" performs set intersection
which_lengths_overlap <- intersect(cdr3_lengths, nchar(cdr3_sequences))

print(which_lengths_overlap)
## [1] 15 22 12

### Produce figure of overlapping CDR3 lengths in the two repertoires

# install.packages("VennDiagram") # Uncomment to install package "VennDiagram"

library(VennDiagram) # Load package "VennDiagram"

# Output figure
# pdf("figure/overlap_cdr3_lengths.pdf") # Uncomment this line to produce file
venn_r1_r2 <- venn.diagram(list("Repertoire 1" = cdr3_lengths,
                                "Repertoire 2" = nchar(cdr3_sequences)),
                            filename = NULL, print.mode = "raw",
                            #set print.mode = "perc", for percentage display
                            fill = c("blue", "orange"),

```

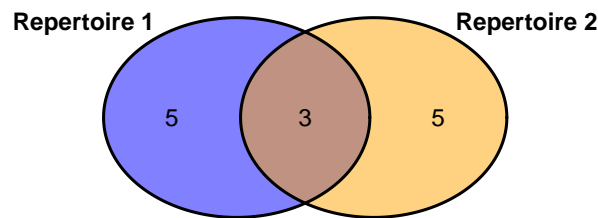
```

cex = 1, fontfamily = "sans",
cat.cex = 1, cat.dist = c(0.1,0.1),
cat.fontface = "bold",
cat.fontfamily = "sans", margin = 0.1)

grid.draw(venn_r1_r2)
# dev.off() # Uncomment this line to produce the pdf file

# Use ?venn.diagram for a detailed explanation of the arguments used

```



2. **Matrix.** Matrices are two dimensional (row x column) objects and can only have one class of data (character, numeric, logical).

```

matrix_example <- matrix(mixed_vector, nrow = 4, ncol = 3)
matrix_example

##      [,1] [,2]      [,3]
## [1,] "A"  "CARRWHQYELCSFANW" "3"
## [2,] "A"  "CARHLMKDPWVCIFRGTESYAW" "6"
## [3,] "A"  "CARRQKYIEACTGHNW" "9"
## [4,] "B"  "CARRIGPEMCSQDAW" "12"

# To order the filling of the matrix by row, set byrow = TRUE
matrix_by_row <- matrix(mixed_vector, nrow = 4, ncol = 3,
                        byrow = T)

matrix_by_row

##      [,1]      [,2]      [,3]
## [1,] "A"      "A"      "A"
## [2,] "B"      "CARRWHQYELCSFANW" "CARHLMKDPWVCIFRGTESYAW"
## [3,] "CARRQKYIEACTGHNW" "CARRIGPEMCSQDAW" "3"
## [4,] "6"      "9"      "12"

```

3. **Dataframe.** A data frame is also a two-dimensional (row x column) object and within each column the data must be of the same class. Statistical functions usually use dataframes.

```

# Transform matrix into dataframe
dataframe_example <- as.data.frame(matrix_example)

# Set row and column names
colnames(dataframe_example) <- c("Sample", "CDR3", "Abundance")

```



```
rownames(dataframe_example) <- c(1:nrow(dataframe_example))

dataframe_example
##      Sample                CDR3 Abundance
## 1      A      CARRWHQYELCSFANW          3
## 2      A CARHLMKDPWVCIFRGTESYAW          6
## 3      A      CARRQKYIEACTGHNW          9
## 4      B      CARRIGPEMCSQDAW         12

# Add new column
dataframe_example$Vgene <- c("V1-2", "V1-69", "V3-46", "V5-1")
```

4. **Arrays** are three dimensional (row x column x height) objects. Within each column the data must be of the same class.

```
array_example <- array(0, dim = c(2, 3, 2))
```

5. **List.** Lists are set of objects, each of which can be a different class, thus very flexible and often allow automation of analysis.

```
list_example <- list(cdr3_lengths, cdr3_sequences, dataframe_example,
                    matrix_example, array_example)

list_example
## [[1]]
## [1] 11 15 22 12 17 20 19 12 21 19
##
## [[2]]
## [1] "CARIMTERSLKW"          "CARFEPHVW"          "CARRWHQYELCSFANW"
## [4] "CARHLMKDPWVCIFRGTESYAW" "CARRQKYIEACTGHNW"    "CARGKYNFIMCQEWTDLVSPRHW"
## [7] "CARRIGPEMCSQDAW"       "CARVRWIHPKQFDLGCAW" "CARWNVPGHDAKCFSW"
## [10] "CARLYEPISDQRHAWGMVTFNCKW"
##
## [[3]]
##      Sample                CDR3 Abundance Vgene
## 1      A      CARRWHQYELCSFANW          3 V1-2
## 2      A CARHLMKDPWVCIFRGTESYAW          6 V1-69
## 3      A      CARRQKYIEACTGHNW          9 V3-46
## 4      B      CARRIGPEMCSQDAW         12 V5-1
##
## [[4]]
##      [,1] [,2]                [,3]
## [1,] "A"  "CARRWHQYELCSFANW"    "3"
## [2,] "A"  "CARHLMKDPWVCIFRGTESYAW" "6"
## [3,] "A"  "CARRQKYIEACTGHNW"    "9"
## [4,] "B"  "CARRIGPEMCSQDAW"     "12"
##
## [[5]]
## , , 1
##
##      [,1] [,2] [,3]
## [1,] 0    0    0
## [2,] 0    0    0
##
```

```
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
```

Google always! Often results will be at stackoverflow (<http://stackoverflow.com/>), a resource for programming.

## 1.5 Accessing Data and Summary Statistics

```
cdr3_lengths
## [1] 11 15 22 12 17 20 19 12 21 19
### Algebraic manipulation
# Algebraic operations are vectorized
cdr3_lengths + 1 # Result is each element of the vector +1:
## [1] 12 16 23 13 18 21 20 13 22 20
# More operations - , /, %*%
### Extract specific values
cdr3_lengths[3]
## [1] 22
cdr3_lengths[2:5] # Sequence of values
## [1] 15 22 12 17
matrix_example[1,] # Returns first row
## [1] "A" "CARRWHQYELCSFANW" "3"
matrix_example[,2] # Returns second column
## [1] "CARRWHQYELCSFANW" "CARHLMKDPWVCIFRGTESYAW" "CARRQKYIEACTGHNW"
## [4] "CARRIGPEMCSQDAW"
dataframe_example$CDR3 # Elements in column "CDR3", equivalent to dataframe_example[, "CDR3"]
## [1] CARRWHQYELCSFANW CARHLMKDPWVCIFRGTESYAW CARRQKYIEACTGHNW
## [4] CARRIGPEMCSQDAW
## Levels: CARHLMKDPWVCIFRGTESYAW CARRIGPEMCSQDAW CARRQKYIEACTGHNW CARRWHQYELCSFANW
colnames(dataframe_example) # Returns column names
## [1] "Sample" "CDR3" "Abundance" "Vgene"
names(dataframe_example) # Returns column names
## [1] "Sample" "CDR3" "Abundance" "Vgene"
list_example[[2]] # Returns 5th element of list
## [1] "CARIMTERSLKW" "CARFEPHVW" "CARRWHQYELCSFANW"
## [4] "CARHLMKDPWVCIFRGTESYAW" "CARRQKYIEACTGHNW" "CARGKYNFIMCQEWTDLVSPRHW"
## [7] "CARRIGPEMCSQDAW" "CARVRWIHPKQFDLGCAW" "CARWNVPGHDAKCFSW"
## [10] "CARLYEPISDQRHAWGMVTFNCKW"
```

```

### Extract unique values
unique(cdr3_lengths)

## [1] 11 15 22 12 17 20 19 21

### Check data: class, length, summary, structure and head
class(cdr3_lengths) # data class (numeric)

## [1] "numeric"

length(cdr3_lengths) # vector length

## [1] 10

summary(cdr3_lengths) # summary statistics

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    11.00  12.75   18.00   16.80   19.75   22.00

str(dataframe_example) # Try ?str # Internal structure of an R object

## 'data.frame': 4 obs. of  4 variables:
##  $ Sample      : Factor w/ 2 levels "A","B": 1 1 1 2
##  $ CDR3         : Factor w/ 4 levels "CARHLMKDPWVCIFRGTESYAW",...: 4 1 3 2
##  $ Abundance    : Factor w/ 4 levels "12","3","6","9": 2 3 4 1
##  $ Vgene       : chr  "V1-2" "V1-69" "V3-46" "V5-1"

### Change data class
dataframe_example$Abundance <- as.numeric(dataframe_example$Abundance)
dataframe_example$CDR3 <- as.character(dataframe_example$CDR3)
dataframe_example$Vgene <- c("V1-2", "V1-69", "V3-2", "V5-1")
head(dataframe_example) # first elements of an object

##   Sample      CDR3 Abundance Vgene
## 1      A  CARRWHQYELCSFANW      2 V1-2
## 2      A CARHLMKDPWVCIFRGTESYAW      3 V1-69
## 3      A  CARRQKYIEACTGHNW      4 V3-2
## 4      B  CARRIGPEMCSQDAW      1 V5-1

```

## 1.6 Visualization

```

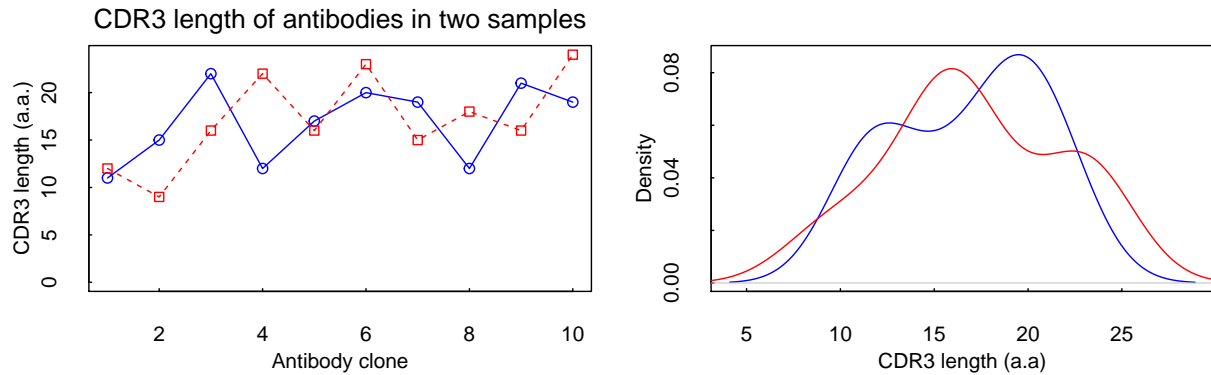
par(mar=c(4,4,2,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,1,0),tcl=.1)
### Line plot of CDR3 lengths
# Graph CDR3 in function of their length in amino acids
plot(cdr3_lengths, type="o", col="blue", xlab="Antibody clone",
     ylab="CDR3 length (a.a)", ylim=c(0,max(nchar(cdr3_sequences))))

# Graph lengths of the CDR3 sequences simulated with red dashed line and square points
lines(nchar(cdr3_sequences), type="o", pch=22, lty=2, col="red")

# Create a title with a red, bold/italic font
title(main="CDR3 length of antibodies in two samples", col.main="black", font.main=1)

### Density plot of CDR3 lengths
plot(density(cdr3_lengths), xlab="CDR3 length (a.a)", main="", col="blue")
lines(density(nchar(cdr3_sequences)), col="red")

```

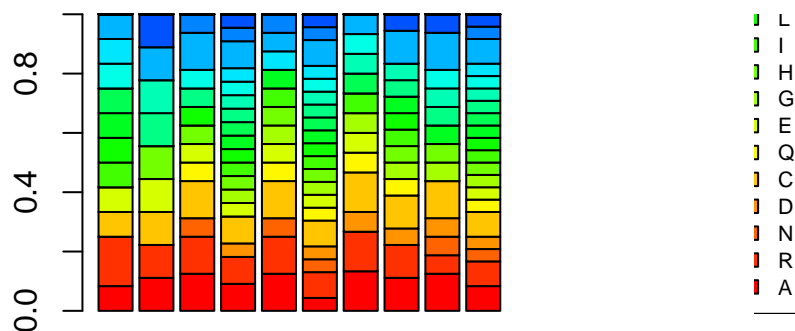


Spearman's  $\rho$  statistic was used to estimate a rank-based measure of association between CDR3 lengths in the 10 antibody clones of the two samples. With knitr, you can also write inline expressions, e.g. we find that  $\rho = 0.2014277$ .

Plotting stacked barchart of the CDR3 a.a. composition.

```
### Stacked barchart of a.a. composition
# pdf("figure/stack_barchart_aafreq.pdf") # uncomment this line to save pdf
seqs <- cdr3_sequences
AAs <- alphabetFrequency(AAStringSet(seqs)) ## alphabetFrequency also works on sets
layout(matrix(c(1,2), 1, 2, byrow = TRUE), width=c(0.7,0.3)) ## allow to place the legend
nAAs = AAs/rowSums(AAs) ## if you want to normalize by total length, use this
barplot(t(nAAs),col=rainbow(ncol(AAs))) ## yields a stacked barplot, one bar per sequence
plot.new()
legend(x="bottom", legend=rev(colnames(AAs)), fill=rev(rainbow(ncol(AAs))),cex=0.7)
# dev.off()

### Piecharts for each of the CDR3 sequences
# for(i in 1:length(cdr3_sequences)){
#   seq = cdr3_sequences[i]
#   AAs = table(strsplit(seq,"", useBytes=TRUE))
#   pie(AAs, col=rainbow(length(AAs)), main="Residue abundance")
# }
```

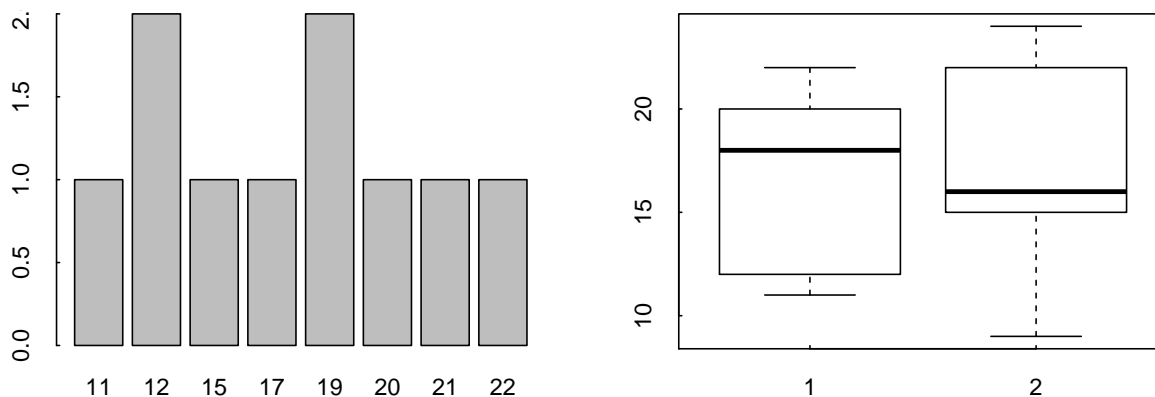


Barplot of CDR3 lengths and boxplot of CDR3 lengths in two samples (1 and 2).

```
par(mar=c(4,4,.1,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,.7,0),tcl=.1)

barplot(table(cdr3_lengths))

boxplot(cdr3_lengths, nchar(cdr3_sequences))
```

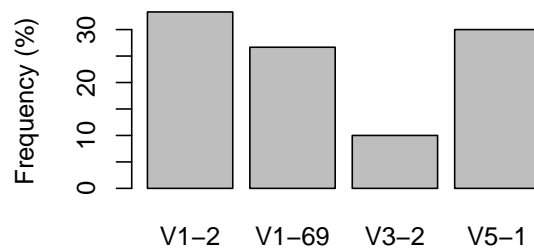


In order to calculate V gene frequencies, use functions `table`, `prop.table` and plot with `barplot`. In case of data from different samples, **ggplot2** [2, 3] is a very useful package for graphics.

```
set.seed(7)

vgene <- sample(dataframe_example$Vgene,30, replace = TRUE)

barplot(prop.table(table(vgene))*100, ylab = "Frequency (%)")
```



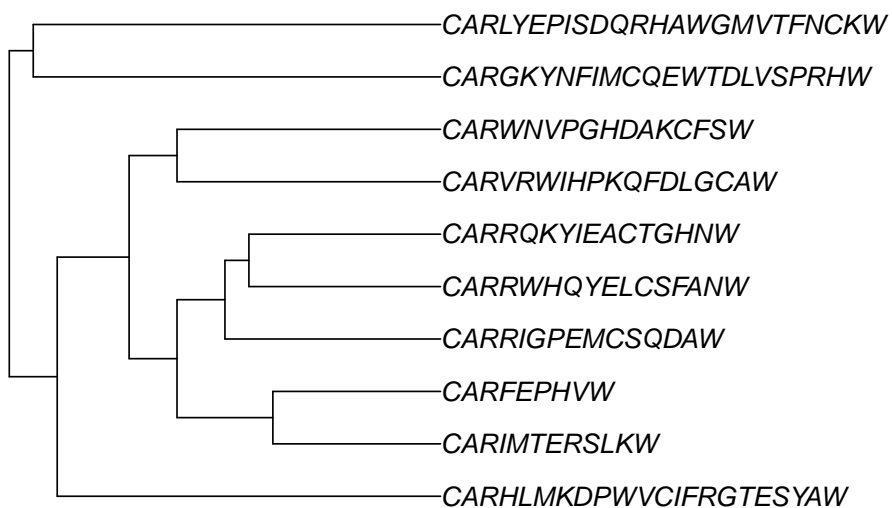
In order to build a phylogenetic tree we need to quantify the sequence-distance between CDR3 clones using the function `stringDist` from `Bistrings` package and `hclust` from the `stats` package that computes hierarchical clustering on a set of dissimilarities. `as.phylo` converts an object into a tree of class "phylo" which can be plotted.

```
library(ape)

cdr3_cluster <- hclust(stringDist(as.character(cdr3_sequences), method = "levenshtein"))

cdr3_cluster$labels = as.character(cdr3_sequences)

plot(as.phylo(cdr3_cluster))
```



## 1.7 R Functions

Standard methods to read (write) data frames from (to) text files use the functions `read.table()`, `read.csv()`, `read.xls()` and `scan()`. To write output in files, use functions similar to `write.table()`.

Read and write data as R object use `save()` and `load()`.

Further R manuals are available at <https://cran.r-project.org/manuals.html>.

Basic functions are described in the following examples:

```
### If statement
# if(logical boolean){function to perform if the boolean is TRUE}
x <- 100000000

if(x > 10000){
  "x is a really big number"
}

## [1] "x is a really big number"

if(x < 10000){
  "x isn't that big"
} ## Nothing happens here because x < 10000 evaluates to false

# Further reading: ifelse statement

### For loop
for(i in 1:10){
  print(i)
} # see while loop

## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10

# Further reading: while loop

### apply family
apply(matrix_example, 1, nchar) # Count the characters by row (1)

##      [,1] [,2] [,3] [,4]
## [1,]   1   1   1   1
## [2,]  16  22  16  15
## [3,]   1   1   1   2

apply(apply(matrix_example, 1, nchar), 2, median) # Median of the previous by column (2)
## [1] 1 1 1 2

sapply(1:10, function(x) x+1) # see apply, lapply, tapply
## [1] 2 3 4 5 6 7 8 9 10 11
```

```

### Writing a function

our_mean_function <- function(x){
  our_sum <- sum(x)
  our_mean <- our_sum / length(x)
  return(our_mean)
}

our_mean_function(1:10)

## [1] 5.5

```

## 2 Bioinformatic Tools for Antibody Repertoire Annotation

### 2.1 IMGT/HighV-QUEST

For large-scale data (hundreds of thousands of sequences) one of the tools to be used for annotation is IMGT/HighV-QUEST [? ]: NGS High-Throughput analysis of IG and TR (<http://www.imgt.org/HighV-QUEST/login.action>).

As input this tool takes paired fasta files and outputs a series of txt files. It is necessary to read-in with R in the IMGT output files in order to extract information in order to further analyze the high-dimensional data.

```

### Read-in IMGT output

# Combine the output files 1_Summary.txt from IMGT
files1 <- c(
  "IMGT/dataset2_pandaseq_aa/1_Summary.txt",
  "IMGT/dataset2_pandaseq_ab/1_Summary.txt",
  "IMGT/dataset2_pandaseq_ac/1_Summary.txt")

# Read-in the specific columns needed for further analysis
ds_files1 <- lapply(1:length(files1), function(x) read.table(files1[x],
  colClasses = c("NULL", "NULL", rep("character", 2), "NULL", "character",
    "NULL", "NULL", "NULL", "character", "NULL", "NULL", "NULL",
    "character", rep("NULL", 4), rep("NULL", 2), rep("NULL", 9)),
  sep="\t", header=TRUE, strip.white=TRUE, na.strings=TRUE, fill=TRUE, nrow=500001))

# In particular, in the first IMGT file read-in the following data
colnames(ds_files1[[1]])
#[1] "Functionality", "V.GENE.and.allele", "V.REGION.identity..",
# "J.GENE.and.allele", "D.GENE.and.allele"

# Label the sample (in this case one sample)
Donor <- rep(c("HIV-1 IAVI 17"), times = c(nrow(ds_files1[[1]]) +
  nrow(ds_files1[[2]]) +
  nrow(ds_files1[[3]])))

# Combine all the rows from the different files
summary_file <- do.call("rbind", ds_files1)

# Combine with the donor label

```



```
summary_files <- cbind(summary_file, Donor)

head(summary_files)
```

Repeat the read-in procedure and row binding for the following IMGT files:

- 1\_Summary.txt, extract functionality (productive/non), V gene and allele, V region identity, J gene and allele, and D gene and allele.
- 3\_Nt-sequences.txt, in order to extract the nucleotides of V.D.J.REGION, V.REGION, FR1.IMGT, CDR1.IMGT, FR2.IMGT, CDR2.IMGT, FR3.IMGT, CDR3.IMGT, JUNCTION, D.REGION, J.REGION and FR4.IMGT
- 5\_AA-sequences.txt, extract the a.a. sequence of the previous regions
- 8\_V-REGION-nt-mutation-statistics.txt for the V region and the other regions (i) V.REGION.Nb.of.mutations, (ii) Nb.of.silent.mutations, and (iii) Nb.of.nonsilent.mutations.
- 9\_V-REGION-AA-change-statistics.txt, extract the number of a.a. changes in the corresponding regions

After saving the analysis with save(), load the data in R, as follows:

```
save(imgt_file, "imgt_file.RData")

load("imgt_file.RData")

# Start processing and analysis
annotated <- nrow(imgt_file) #[1] 3350792
```

## 2.2 MiXCR [1]

Run MiXCR:

1. Align reads generating a report and an alignment file  
\$ java -jar mixcr.jar align -loci IGH -species hsa -report alignmentReport\_dataset2.log -save-description R\_2013\_11\_20\_17\_05\_16\_user\_SN2-10-Donate\_517\_5\_RACE\_11813.fastq alignment\_dataset2.vdjca
2. Assemble into clones  
\$ java -jar mixcr.jar assemble -OassemblingFeatures="[CDR3]" -report assembleReport\_dataset2.log alignment\_dataset2.vdjca clones\_dataset2.clns
3. Transform into text  
\$ java -jar mixcr.jar exportClones clones\_dataset2.clns clones\_dataset2.txt

Read-in the clones.txt output file from MiXCR.

```
### Read-in MiXCR output

colclasses <- rep("NULL", 33)

colclasses[c(1,5,6,7,8,9,10,11,32)] <- rep("character", length(c(1,5,6,7,8,9,10,11,32)))

cdr3_clones <- read.csv2("clones.txt", row.names=NULL, sep = "\t", colClasses = colclasses)
```

## 2.3 R packages needed for the analysis

For preliminary processing and data analysis, the following R libraries have proven useful in the past:

```
library(pcaMethods)
library(seqinr)
library(RColorBrewer)
library(xtable)
library(plyr)
library(ggplot2)
library(ShortRead)
library(grid)
library(reshape)
library(ape)
library(phylotools)
library(stringr)
library(gridExtra)
library(hexbin)
library(data.table)
library(VennDiagram)
library(scales)
library(fastmatch)
library(HDMD)
library(Biobase)
library(pcaMethods)
library(Biostrings)
library(stringdist)
library(ConsensusClusterPlus)
library(Hmisc)
library(gplots)
library(corrgram)
library(igraph)
library(NMF)
```

## Part II

# Preprocessing and Analysis of Antibody Repertoire NGS Data

## 3 Databases and data repositories

NGS data are usually deposited with the acceptance for publication of an original research article. Public databases contain the deposition of single sequences of from B cell receptors (immunoglobulins), T cell receptors, multiple sequences of immune repertoires samples and germlines.



Figure 1: Databases of immune sequences.

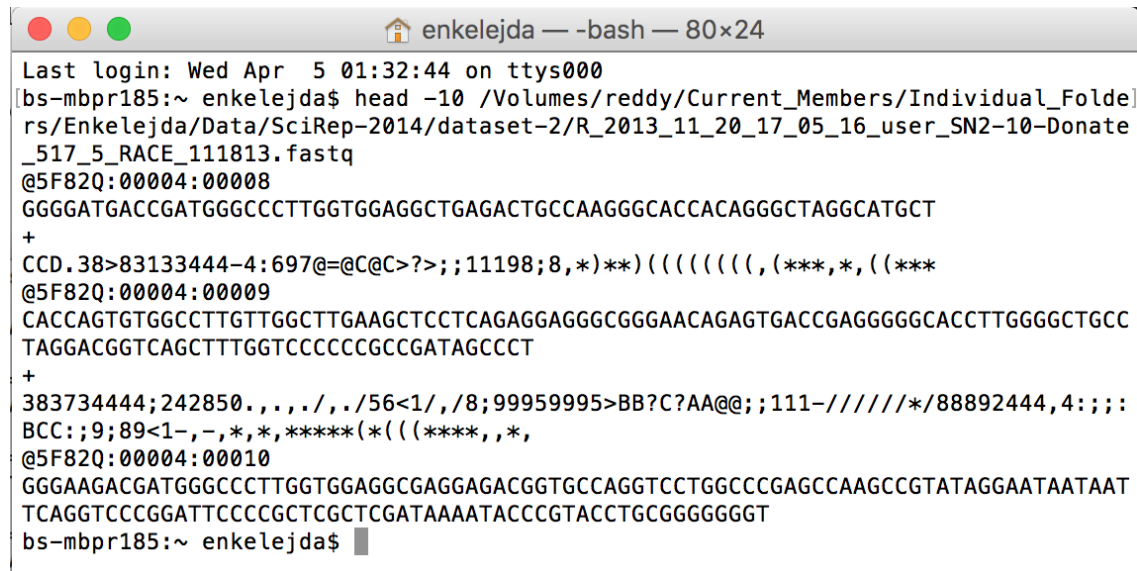
Main databases:

- Sequence Read Archive (SRA, <https://www.ncbi.nlm.nih.gov/sra>)
- European Nucleotide Archive (ENA, <http://www.ebi.ac.uk/ena>)
- IMGT databases (7 databases, <http://www.imgt.org/> [6])
- RCSB Protein Data Bank (<http://www.rcsb.org/pdb/home/home.do> [7])
- GenBank, the NIH genetic sequence database (<http://www.rcsb.org/pdb/home/home.do> [8])

Databases of antigen-specific sequences include:

- bNAber database of broadly neutralizing antibodies against HIV-1 (<http://bnaber.org/> [9])
- Los Alamos National Laboratory HIV Molecular Immunology Database (<https://www.hiv.lanl.gov/content/immunology/index.html>)
- AbYsis (<http://www.bioinf.org.uk/abysis2.7/> [10])

Sequences are downloadable as fastq or fasta files. The head (first 10 lines) of a fastq file is shown in Figure 2, as obtained from the terminal.



```

Last login: Wed Apr  5 01:32:44 on ttys000
bs-mbpr185:~ enkelejda$ head -10 /Volumes/reddy/Current_Members/Individual_Folde
rs/Enkelejda/Data/SciRep-2014/dataset-2/R_2013_11_20_17_05_16_user_SN2-10-Donate
_517_5_RACE_111813.fastq
@5F82Q:00004:00008
GGGGATGACCGATGGGCCCTTGGTGGAGGCTGAGACTGCCAAGGGCACCACAGGGCTAGGCATGCT
+
CCD.38>83133444-4:697@=CC@C>?>;11198;8,*)**)(((((((, (**,*, ((**
@5F82Q:00004:00009
CACCAGTGTGGCCTTGTGGCTTGAAGCTCCTCAGAGGAGGGCGGGAACAGAGTGACCGAGGGGGCACCTTGGGGCTGCC
TAGGACGGTCAGCTTTGGTCCCCCGCCGATAGCCCT
+
383734444;242850.,.,.,./56<1/,/8;99959995>BB?C?AA@@;111-////////*/88892444,4;;;:
BCC:;9;89<1-,-,*,*,*****(((((****,*,
@5F82Q:00004:00010
GGGAAGACGATGGGCCCTTGGTGGAGGCGAGGAGACGGTGCCAGGTCTGGCCCGAGCCAAGCCGTATAGGAATAATAAT
TCAGGTCCCGGATTCCCCGCTCGCTCGATAAAATACCCGTACCTGCGGGGGGGT
bs-mbpr185:~ enkelejda$

```

Figure 2: Fastq file header.

### 3.1 Guide to data retrieval and download

Accession codes, thus the fastq files to which they refer, can be tricky to find and to link to the original biological sample sequenced and its information. Researchers are aware that good quality public datasets are often (if not exclusively) connected to an original publication. The raw sequences are often submitted to an online database upon acceptance for publication. We refer in this section to the articles that have uploaded immune repertoire sequences and/or germlines.

For example, the "Accession code. Sequence data, SRA: SRA061316." is reported in the original publication for the data from *DeKosky, BJ et al. High-throughput sequencing of the paired human immunoglobulin heavy and light chain repertoire. Nature Biotechnology. 31(2):166-169 (2013) [11].* Accessing the SRA at <https://www.ncbi.nlm.nih.gov/sra/> and searching for **SRA061316** as shown in Figure 3, will result in the sample run **SRR611539** as shown in Figure 4.

By following the instructions provided at <https://www.ncbi.nlm.nih.gov/books/NBK158899/>, the fastq file for the IgG repertoire of Donor 2 can be downloaded by:

1. Download and install the SRA Toolkit:  
`https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software`
2. Double-click on the .tar.gz file and the Archive Utility will unpack it. Alternatively, command-line: `$ tar -xzf sratoolkit.current-centos_linux64.tar.gz`

Secure <https://www.ncbi.nlm.nih.gov/sra/?term=SRA061316>

SRA  [Create alert](#) [Advanced](#)

Summary ▾

[Send results to Blast](#)

### Search results

Items: 5

- ☐ [CAP256 VH:VL sequencing time course](#)
  1. 2 ILLUMINA (Illumina HiSeq 2500) runs: 17.4M spots, 8.7G bases, 5.5Gb downloads  
Accession: SRX397134
- ☐ [IgG Repertoire - Donor 2](#)
  2. 1 ILLUMINA (Illumina MiSeq) run: 1.2M spots, 605.6M bases, 436.8Mb downloads  
Accession: SRX203003

Figure 3: SRA search.

#### **SRX203003: IgG Repertoire - Donor 2**

1 ILLUMINA (Illumina MiSeq) run: 1.2M spots, 605.6M bases, 436.8Mb downloads

**Submitted by:** THE UNIVERSITY OF TEXAS AT AUSTIN

**Study:** Homo sapiens Transcriptome or Gene expression

[PRJNA179099](#) • [SRP017087](#) • [All experiments](#) • [All runs](#)

[show Abstract](#)

**Sample:** IgG Repertoire - Donor 2

[SAMN01801748](#) • [SRS374258](#) • [All experiments](#) • [All runs](#)

**Organism:** [Homo sapiens](#)

#### **Library:**

*Instrument:* Illumina MiSeq

*Strategy:* AMPLICON

*Source:* TRANSCRIPTOMIC

*Selection:* RT-PCR

*Layout:* PAIRED

#### **Spot descriptor:**



**Runs:** 1 run, 1.2M spots, 605.6M bases, [436.8Mb](#)

Run	# of Spots	# of Bases	Size	Published
<a href="#">SRR611539</a>	1,206,350	605.6M	436.8Mb	2013-01-06

Figure 4: SRA result.

3. Open a terminal or command prompt and "cd" into the directory containing the toolkit executables (e.g., [download\_location]/sratoolkit[version]/bin/). Linux and OS X users should execute the following command to check the correct download, installation and configuration:

```
enkelejda$ ./fastq-dump -X 5 -Z SRR390728
```

Download the data:

```
bs-mbpr185:bin enkelejda$ ./fastq-dump SRR611539
Read 1206350 spots for SRR611539
Written 1206350 spots for SRR611539
bs-mbpr185:bin enkelejda$
```

## 4 Assembly of paired-end reads

PANDAsq is a program to align forward and reverse Illumina reads and reconstruct an overlapping sequence [12]. All documentation is available online: <https://github.com/neufeld/pandaseq>.

### 4.1 Software installation

Follow the instructions for your system: <https://github.com/neufeld/pandaseq/wiki/Installation>

### 4.2 Input

Pair forward (-f) and reverse (-r) reads in fastq files (R1 and R2) from the Terminal, outputting (-W) a fasta file:

```
$ pandaseq -f path/Sample_IgG1_R1.fastq -r path/Sample_IgG1_R2.fastq -w path/Sample_IgG1.fasta
```

Split fastq files:

```
$ split -l 1000000 path/Sample_IgG1.fasta path/Sample_IgG1/
```

Check how many lines there are in one file:

```
$ wc -l path/Sample_IgG1/aa 1000000 path/Sample_IgG1/aa
```

So there are 1,000,000 lines which correspond to 500,000 sequences because a fasta file, 2 lines/sequence.

### 4.3 Output

Fasta files are output from the pairing procedure with PANDAsq (Figure 5).

```

enkelejda — -bash — 80x24
Last login: Thu Apr 6 19:06:43 on ttys000
bs-mbpr185:~ enkelejda$ head -10 /Volumes/reddy/Current_Members/Individual_Folde
rs/Enkelejda/Data/SciRep-2014/dataset-2/dataset2_pandaseq_aa
>5F82Q:00004:00008
GGGGATGACCGATGGGCCCTTGGTGGAGGCTGAGACTGCCAAGGGCACCACAGGGCTAGGCATGCT
>5F82Q:00004:00009
CACCAGTGTGGCCTTGTGGCTTGAAGCTCCTCAGAGGAGGGCGGGAACAGAGTGACCGAGGGGGCACCTTGGGGCTGCC
TAGGACGGTCAGCTTTGGTCCCCCGCCGATAGCCCT
>5F82Q:00004:00010
GGGAAGACGATGGGCCCTTGGTGGAGGCGAGGAGACGGTGCCAGGTCTGGCCCGAGCCAAGCCGTATAGGAATAATAAT
TCAGGTCCCGGATTCCCGCTCGCTCGATAAAATACCCGTACCTGCGGGGGGGT
>5F82Q:00004:00012
CACCAGTGTGGCCTTGTGGCTTGAAGCTCCTCAGAGGAGGGCGGGAACAGAGTGACCGAGGGGGCACCTTGGGGCTGAC
TAGGACGGTCACCTTTGGTCCCTCCGCCGAAACACTCTTCAGGCTGTGTCACAGTGTATATAACTCACGCCGCTTCGA
CTCCTCGGACTGGACGCCCAGTGATTGGGCCCAGGGGAGGCTGGATAGCAGCCCCGTCGAAGGGGCGGGGAGTGGAAGG
ACCACCACCCACCCCTTTTCCAGTCAAAACCTGCAAAACCGCAGTGGTTA
>5F82Q:00004:00020
GGGGAAGACCGATGGGCCCTTGGTGGAGGCGAGGAGACGGTGACCAGGGTTCCTGGCCCCAAGGGTGAACACCTCC
ACTTCTGCTGAACGACCACTCCAATGAGGGAGCCAGGCCTCTCGACAATAATAAAATGCCGTGTCTGCGACCGTCAAA
GAGGTCATCTGCAGGGAACCTGATTTTGAAGTGTCTACTGAGTGCGACCGTACACTGAGGGTGAGAGCGGTGTGATG
CCAGCTCAGATTAGTGCCCTGT
bs-mbpr185:~ enkelejda$

```

Figure 5: Fasta file header.

## 5 Read annotation

Reads in fasta files are annotated through IMGT/HighV-QUEST [13, 14]. Login online to IMGT/HighV-QUEST: [www.imgt.org/HighV-QUEST/login.action](http://www.imgt.org/HighV-QUEST/login.action) (Figure 6).

**WELCOME!**  
to **IMGT/HighV-QUEST**

THE INTERNATIONAL IMMUNOGENETICS INFORMATION SYSTEM®



IMGT/HighV-QUEST version: 1.5.2 (17 January 2017) IMGT/HighV-QUEST version: 3.4.3 (5 December 2016) IMGT/HighV-QUEST reference directory release: 201648-4 (8 December 2016)

**Citing IMGT/HighV-QUEST:**  
Alamyar, et al. IMGT/HighV-QUEST: The IMGT® web portal for immunoglobulin (IG) or antibody and T cell receptor (TR) analysis from NGS high throughput and deep sequencing. *Immunome Res.* 8:12 (2012). LIGM-400. PMID:22647594. [PubMed](#)  
Alamyar E., et al., *Methods Mol. Biol.* 882:569-604 (2012). PMID:22655256. LIGM-404  
Li S., et al. IMGT/HighV-QUEST paradigm for T cell receptor (TR) clonotype, clonal expression evaluation diversity and next generation repertoire immunoprofiling. *Nat. Commun.* 4:2333 (2013). [Open access](#). PMID:23956977. LIGM-419  
Giudicelli V., et al., *Autoimmun Infect Dis.* 1(1) (2015). doi:10.16966/aidoa.103. [Free Article](#) LIGM-448

Please login here

E-mail:

Password:

[Privacy & Terms](#)

You can change the two words to be typed by clicking on the upper blue button ("Get a new challenge"), they are not case sensitive.

- IMGT/HighV-QUEST is an online tool that gives IMGT/HighV-QUEST users the possibility to analyze huge number of rearranged immunoglobulin (IG) and T cell receptor (TR) sequences in one run.
- Once your job is launched and analysis results are available, you can use 'Analysis history' page or click on the link sent to you after each job launch to cope with your results.
- All users should be identified before using this platform. If you are a new user click on 'New user' under the login form to register, once you are registered you can use IMGT/HighV-QUEST provided that your account is activated by IMGT/HighV-QUEST administrator.

IMGT/HighV-QUEST is free for academics and under CNRS licence agreement for companies.

Figure 6: IMGT/HighV-QUEST login page.

## 5.1 Input to IMGT

Upload to IMGT fasta files (Figure 5) up to 500 000 sequences as shown in Figure ?? and check-in the boxes to output results in CSV files as shown in Figure 7.

WELCOME!  
to **IMGT/HighV-QUEST**

THE INTERNATIONAL IMMUNOGENETICS INFORMATION SYSTEM®

Login: enkelejda@base.ethz.ch IMGT/HighV-QUEST Search page Analysis history Launch statistics Statistics history IMGT/StatClonotype **NEW!** Help Logout



IMGT/HighV-QUEST version: 1.5.2 (17 January 2017) IMGT/V-QUEST version: 3.4.3 (5 December 2016) IMGT/V-QUEST reference directory release: 201649-4 (8 December 2016)

**Citing IMGT/HighV-QUEST:**  
Alamyar, et al. IMGT/HighV-QUEST: The IMGT® web portal for immunoglobulin (IG) or antibody and T cell receptor (TR) analysis from NGS high throughput and deep sequencing. Immunome Res. 8:1-2 (2012). LIGM-400 PMID:22647994 **DOI**  
Alamyar E., et al., Methode Mol. Biol. 882:569-604 (2012). PMID:22652556 LIGM-404  
Li S., et al. IMGT/HighV-QUEST paradigm for T cell receptor IMGT clonotype, clonal expression evaluation diversity and next generation repertoire immunoprofiling. Nat. Commun. 4:2333 (2013). Open access. PMID:23995877 LIGM-419  
Giudicelli V., et al., Autoimmun Infect Dis. 1(1) (2015). doi:10.1096/aidoa.103. **Free Article** LIGM:448

🔗 To launch statistical analysis on completed jobs (now with IMGT clonotype (AA)), click on 'Launch statistics' on the menu bar.

Analysis title:  (50 characters or less)

Species:

Receptor type or locus:

Sequences are from a single individual:

Give the path access to a local file (in simple text format) containing your sequences in **FASTA format** (from up to sequences)  
 No file chosen

🔗 For IMGT/HighV-QUEST submissions >150 000 sequences, the individual files are not provided.

Send me an e-mail notification: ☐ when analysis is queued ☐ when analysis is submitted ☒ when analysis is completed [Check all](#) | [None](#)

Figure 7: IMGT/HighV-QUEST display results.

## 5.2 Explore output IMGT

The results of IMGT/HighV-QUEST consist of annotated reads [13]. Briefly, each nucleotide sequence is translated to a.a., V-, (D- for heavy chains) and J- genes are assigned to each read, and regions are defined such as framework region 1 (FR1), FR2, FR3 and FR4, and complementarity region 1 (CDR1), CDR2 and CDR3; mutations are tracked and further read-characteristics are calculated [15].

Title	User	Status	Submission date	Nb of sequences	IMGT/V-QUEST reference directory		Actions
					Species	Receptor type (or locus)	
SC44_bnab_25months	Enkelejda MIHO	completed	2016-10-14 2:53:10 <a href="#">CET</a>	64568	Homo_sapiens	IG	(file not found)
SC44_bnab_11months	Enkelejda MIHO	completed	2016-10-14 2:52:10 <a href="#">CET</a>	21406	Homo_sapiens	IG	(file not found)

Figure 8: IMGT/HighV-QUEST analysis history.

Results output from IMGT appears in the History tab and can be downloaded (Figure 8). Files must be removed from IMGT and saved elsewhere as the results will be eliminated after. Output from IMGT files consists in 9 CSV files (Figure 9; CSV: comma separated values files, where data is saved in a table structured format). These CSV that can be open and explored for example with Microsoft Excel. However, the quantity of data in the files (that for 500,000 sequences have 500,000 rows, one row for each sequence) often does not allow a fast exploration and analysis. We use thus R to readin and analyze the data.



Name	Size	Kind
1_Summary.txt	439.3 MB	Plain Text
2_IMGT-gapped-nt-sequences.txt	405.7 MB	Plain Text
3_Nt-sequences.txt	446.8 MB	Plain Text
4_IMGT-gapped-AA-sequences.txt	163 MB	Plain Text
5_AA-sequences.txt	146.5 MB	Plain Text
6_Junction.txt	135.5 MB	Plain Text
7_V-REGION-mutation-and-AA-change-table.txt	557.9 MB	Plain Text
8_V-REGION-nt-mutation-statistics.txt	134.2 MB	Plain Text
9_V-REGION-AA-change-statistics.txt	109.3 MB	Plain Text
10_V-REGION-mutation-hotspots.txt	313.6 MB	Plain Text
11_Parameters.txt	502 bytes	Plain Text
README.txt	1 KB	Plain Text

Figure 9: IMGT/HighV-QUEST result files.

```
### Read-in IMGT CSV output. Set "eval=TRUE" to run the code.

# Insert path to 1_Summary.txt files output from IMGT/High-VQUEST
# Files to read in are CSV IMGT output files from dataset2 and dataset4
files1 <- c("IMGT/dataset2_pandaseq_aa/1_Summary.txt",
            "IMGT/dataset2_pandaseq_ab/1_Summary.txt",
            "IMGT/dataset2_pandaseq_ac/1_Summary.txt",
            "IMGT/dataset2_pandaseq_ad/1_Summary.txt",
            "IMGT/dataset2_pandaseq_ae/1_Summary.txt",
            "IMGT/dataset2_pandaseq_af/1_Summary.txt",
            "IMGT/dataset2_pandaseq_ag/1_Summary.txt",
            "IMGT/dataset4_pandaseq_aa/1_Summary.txt",
            "IMGT/dataset4_pandaseq_ab/1_Summary.txt",
            "IMGT/dataset4_pandaseq_ac/1_Summary.txt",
            "IMGT/dataset4_pandaseq_ad/1_Summary.txt",
            "IMGT/dataset4_pandaseq_ae/1_Summary.txt",
            "IMGT/dataset4_pandaseq_af/1_Summary.txt",
            "IMGT/dataset4_pandaseq_ag/1_Summary.txt")

# Read in selected columns: apply the function read.table
# to the list of files save in the variable "files1".
# "character" instructs the read in,
# "NULL" skips the column.
ds_files1 <- lapply(1:length(files1), function(x) read.table(files1[x],
                    colClasses = c("NULL", "NULL", rep("character", 2), "NULL",
                                   "character", "NULL", "NULL", "NULL", "character",
                                   "NULL", "NULL", "NULL", "character", rep("NULL", 4),
                                   rep("NULL", 2), rep("NULL", 9)),
                    sep="\t", header=TRUE, strip.white=TRUE,
                    na.strings=TRUE, fill=TRUE, nrows=500001))

# Output of the columns that have been read-in
```

```

colnames(ds_files1[[1]])
# [1] "Functionality", "V.GENE.and.allele", "V.REGION.identity..",
# "J.GENE.and.allele", "D.GENE.and.allele"

# Label the donor for each sequence in each CSV row of the read-in files
hiv_iavi17 <- rep(c("HIV-1 IAVI donor 17"),
  times = c(nrow(ds_files1[[1]]) + nrow(ds_files1[[2]]) +
    nrow(ds_files1[[3]]) + nrow(ds_files1[[4]]) +
    nrow(ds_files1[[5]]) + nrow(ds_files1[[6]]) +
    nrow(ds_files1[[7]])))

healthy_donor1 <- rep(c("Healthy donor 1"),
  times = c(nrow(ds_files1[[8]]) + nrow(ds_files1[[9]]) +
    nrow(ds_files1[[10]]) + nrow(ds_files1[[11]]) +
    nrow(ds_files1[[12]]) + nrow(ds_files1[[13]]) +
    nrow(ds_files1[[14]])))

Donor <- c(hiv_iavi17, healthy_donor1)

# Bind the all the rows with "rbind"
summary_file <- do.call("rbind", ds_files1)

# Change the name to the column "V.REGION.identity.."
colnames(summary_file)[3] <- c("V.REGION.identity")

files3 <- c("IMGT/dataset2_pandaseq_aa/3_Nt-sequences.txt",
  "IMGT/dataset2_pandaseq_ab/3_Nt-sequences.txt",
  "IMGT/dataset2_pandaseq_ac/3_Nt-sequences.txt",
  "IMGT/dataset2_pandaseq_ad/3_Nt-sequences.txt",
  "IMGT/dataset2_pandaseq_ae/3_Nt-sequences.txt",
  "IMGT/dataset2_pandaseq_af/3_Nt-sequences.txt",
  "IMGT/dataset2_pandaseq_ag/3_Nt-sequences.txt",
  "IMGT/dataset4_pandaseq_aa/3_Nt-sequences.txt",
  "IMGT/dataset4_pandaseq_ab/3_Nt-sequences.txt",
  "IMGT/dataset4_pandaseq_ac/3_Nt-sequences.txt",
  "IMGT/dataset4_pandaseq_ad/3_Nt-sequences.txt",
  "IMGT/dataset4_pandaseq_ae/3_Nt-sequences.txt",
  "IMGT/dataset4_pandaseq_af/3_Nt-sequences.txt",
  "IMGT/dataset4_pandaseq_ag/3_Nt-sequences.txt")

ds_files3 <- lapply(1:length(files3), function(x) read.csv(files3[x],
  colClasses = c(rep("NULL", 6), "character", "NULL",
    rep("character", 8), rep("NULL", 7),
    "character", rep("NULL", 16),
    rep("character", 2), rep("NULL", 72)),
  sep="\t", header=TRUE, strip.white=TRUE,
  na.strings=TRUE, fill=TRUE, nrows=500001))

nt_sequences_file <- do.call("rbind", ds_files3)

colnames(nt_sequences_file)
# [1] "V.D.J.REGION" "V.REGION" "FR1.IMGT"
# "CDR1.IMGT" "FR2.IMGT" "CDR2.IMGT" "FR3.IMGT"

```

```

# [8] "CDR3.IMG" "JUNCTION" "D.REGION" "J.REGION"
# "FR4.IMG"

colnames(nt_sequences_file)[1:12] <- c("V.D.J.REGION.nt", "V.REGION.nt",
"FR1.IMG.nt", "CDR1.IMG.nt", "FR2.IMG.nt", "CDR2.IMG.nt", "FR3.IMG.nt",
"CDR3.IMG.nt", "JUNCTION.nt", "D.REGION.nt", "J.REGION.nt", "FR4.IMG.nt")

files5 <- c("IMGT/dataset2_pandaseq_aa/5_AA-sequences.txt",
"IMGT/dataset2_pandaseq_ab/5_AA-sequences.txt",
"IMGT/dataset2_pandaseq_ac/5_AA-sequences.txt",
"IMGT/dataset2_pandaseq_ad/5_AA-sequences.txt",
"IMGT/dataset2_pandaseq_ae/5_AA-sequences.txt",
"IMGT/dataset2_pandaseq_af/5_AA-sequences.txt",
"IMGT/dataset2_pandaseq_ag/5_AA-sequences.txt",
"IMGT/dataset4_pandaseq_aa/5_AA-sequences.txt",
"IMGT/dataset4_pandaseq_ab/5_AA-sequences.txt",
"IMGT/dataset4_pandaseq_ac/5_AA-sequences.txt",
"IMGT/dataset4_pandaseq_ad/5_AA-sequences.txt",
"IMGT/dataset4_pandaseq_ae/5_AA-sequences.txt",
"IMGT/dataset4_pandaseq_af/5_AA-sequences.txt",
"IMGT/dataset4_pandaseq_ag/5_AA-sequences.txt")

ds_files5 <- lapply(1:length(files5), function(x) read.table(files5[x],
colClasses = c(rep("NULL", 6), "character", "NULL",
rep("character", 10)),
sep="\t", header=TRUE, strip.white=TRUE,
na.strings=TRUE, fill=TRUE, nrow=101))

aa_sequence_file <- do.call("rbind", ds_files5)

colnames(aa_sequence_file)
# [1] "V.D.J.REGION" "V.REGION" "FR1.IMG" "CDR1.IMG"
# "FR2.IMG" "CDR2.IMG" "FR3.IMG"
# [8] "CDR3.IMG" "JUNCTION" "J.REGION" "FR4.IMG" "Donor"

files8 <- c("IMGT/dataset2_pandaseq_aa/8_V-REGION-nt-mutation-statistics.txt"
,
"IMGT/dataset2_pandaseq_ab/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset2_pandaseq_ac/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset2_pandaseq_ad/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset2_pandaseq_ae/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset2_pandaseq_af/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset2_pandaseq_ag/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset4_pandaseq_aa/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset4_pandaseq_ab/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset4_pandaseq_ac/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset4_pandaseq_ad/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset4_pandaseq_ae/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset4_pandaseq_af/8_V-REGION-nt-mutation-statistics.txt",
"IMGT/dataset4_pandaseq_ag/8_V-REGION-nt-mutation-statistics.txt")

ds_files8 <- lapply(1:length(files8), function(x) read.table(files8[x],
colClasses = c(rep("NULL", 7), rep("character",3), rep("NULL", 15),

```

```

rep("character", 3), rep("NULL", 15), rep("character", 3),
rep("NULL", 15), rep("character", 3), rep("NULL", 15),
rep("character", 3), rep("NULL", 15), rep("character", 3),
rep("NULL", 15), rep("character", 3), rep("NULL", 12)),
sep="\t", header=TRUE, strip.white=TRUE,
na.strings=TRUE, fill=TRUE, nrow=500001))

nt_mutation_file <- do.call("rbind", ds_files8)

colnames(nt_mutation_file)
# [1] "V.REGION.Nb.of.mutations" "V.REGION.Nb.of.silent.mutations"
# "V.REGION.Nb.of.nonsilent.mutations" [4] "FR1.IMGT.Nb.of.mutations"
# "FR1.IMGT.Nb.of.silent.mutations" "FR1.IMGT.Nb.of.nonsilent.mutations"
# [7] "CDR1.IMGT.Nb.of.mutations" "CDR1.IMGT.Nb.of.silent.mutations"
# "CDR1.IMGT.Nb.of.nonsilent.mutations" [10] "FR2.IMGT.Nb.of.mutations"
# "FR2.IMGT.Nb.of.silent.mutations" "FR2.IMGT.Nb.of.nonsilent.mutations"
# [13] "CDR2.IMGT.Nb.of.mutations" "CDR2.IMGT.Nb.of.silent.mutations"
# "CDR2.IMGT.Nb.of.nonsilent.mutations" [16] "FR3.IMGT.Nb.of.mutations"
# "FR3.IMGT.Nb.of.silent.mutations" "FR3.IMGT.Nb.of.nonsilent.mutations"
# [19] "CDR3.IMGT.Nb.of.mutations" "CDR3.IMGT.Nb.of.silent.mutations"
# "CDR3.IMGT.Nb.of.nonsilent.mutations"

files9 <- c("IMGT/dataset2_pandaseq_aa/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset2_pandaseq_ab/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset2_pandaseq_ac/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset2_pandaseq_ad/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset2_pandaseq_ae/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset2_pandaseq_af/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset2_pandaseq_ag/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset4_pandaseq_aa/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset4_pandaseq_ab/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset4_pandaseq_ac/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset4_pandaseq_ad/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset4_pandaseq_ae/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset4_pandaseq_af/9_V-REGION-AA-change-statistics.txt",
"IMGT/dataset4_pandaseq_ag/9_V-REGION-AA-change-statistics.txt")

ds_files9 <- lapply(1:length(files9), function(x) read.table(files9[x],
colClasses = c(rep("NULL", 5), "character", "NULL", "character",
rep("NULL", 12), "character", "NULL", "character", rep("NULL", 12),
"character", "NULL", "character", rep("NULL", 12), "character",
"NULL", "character", rep("NULL", 12), "character", "NULL",
"character", rep("NULL", 12), "character", "NULL", "character",
rep("NULL", 12), "character", "NULL", "character", rep("NULL", 11)),
sep="\t", header=TRUE, strip.white=TRUE,
na.strings=TRUE, fill=TRUE, nrow=101))

aa_change_file <- do.call("rbind", ds_files9)

colnames(aa_change_file)
# [1] "V.REGION.Nb.of.AA" "V.REGION.Nb.of.AA.changes" "FR1.IMGT.Nb.of.AA"
# [4] "FR1.IMGT.Nb.of.AA.changes" "CDR1.IMGT.Nb.of.AA" "CDR1.IMGT.Nb.of.AA.changes"
# [7] "FR2.IMGT.Nb.of.AA" "FR2.IMGT.Nb.of.AA.changes" "CDR2.IMGT.Nb.of.AA"

```

```

# [10] "CDR2.IMGT.Nb.of.AA.changes" "FR3.IMGT.Nb.of.AA" "FR3.IMGT.Nb.of.AA.changes"
# [13] "CDR3.IMGT.Nb.of.AA" "CDR3.IMGT.Nb.of.AA.changes"

### Extracted parameters from IMGT files
# nt_sequences_files
# aa_sequence_files
# nt_mutation_files
# aa_change_files

# Bind the the aa files (all rows together) to the "Donor" column
# that indicates the donor for each sequence-row
imgt_files <- cbind(summary_file, aa_sequence_file, aa_change_file, Donor)

# How many columns are there?
ncol(imgt_files)
#[1] 31

### Save your data for faster and access without re-running read-in
# save(imgt_files, file = "imgt_files.RData")

```

## 6 Filtering

### 6.1 Filtering CDR3 singletons, short CDR3, stop codons

Briefly, selection and extraction of features (numbers, IGHV genes, etc) are performed. Sequences with missing values, with stop codons, undetermined a.a. are filtered out and productive sequences that appear more than twice in a dataset and are longer than 2 a.a. are filtered in. Then data is processed further to match mutations and other parameters to unique CDR3s.

```

# Start timing of the code
start <- proc.time()

### Filter and preprocess IMGT CSV output

# Load in data extracted from IMGT/High-VQUEST CSV result files
load("imgt_files.RData")

# Match samples. Check the function by running: unique(imgt_files$Donor)
# and levels(factor(imgt_files$Donor))
match_stage <- match(unique(imgt_files$Donor), levels(factor(imgt_files$Donor)))

# Count annotated total sequences per sample
annotated_seq_initial <- data.frame(summary(imgt_files$Donor)[match_stage])

# Select columns
selection <- imgt_files[,c("Functionality", "V.GENE.and.allele",
    "D.GENE.and.allele", "J.GENE.and.allele", "V.D.J.REGION",
    "V.REGION", "JUNCTION", "J.REGION", "FR1.IMGT", "FR2.IMGT",
    "FR3.IMGT", "FR4.IMGT", "CDR1.IMGT", "FR1.IMGT", "CDR2.IMGT",
    "CDR3.IMGT", "V.REGION.Nb.of.AA", "V.REGION.Nb.of.AA.changes",
    "FR1.IMGT.Nb.of.AA.changes", "FR2.IMGT.Nb.of.AA.changes",
    "FR3.IMGT.Nb.of.AA.changes", "CDR1.IMGT.Nb.of.AA.changes",

```

```

        "CDR2.IMGT.Nb.of.AA.changes", "CDR3.IMGT.Nb.of.AA.changes",
        "Donor")]

# How many NA in data?
sum(is.na(selection))
# [1] 0

# How many sequences in total?
nrow(selection)
# [1] 6455246

selection_df <- data.frame(summary(selection$Donor)[match_stage])

# Select for productive reads according to IMGT functionality
selection_productive <- selection$Functionality=="productive"
productive <- selection[(selection_productive),] # nrow(productive) [1] 195242

sum(is.na(productive))
# [1] 0

nrow(productive)
#[1] 441540

productive_df <- data.frame(summary(productive$Donor)[match_stage])

# Add lengths of regions as additional columns of the data frame
productive$V.REGION.Length <- nchar(productive[, "V.REGION"])
productive$V.D.J.REGION.Length <- nchar(productive[, "V.D.J.REGION"])
productive$JUNCTION.Length <- nchar(productive[, "JUNCTION"])
productive$J.REGION.Length <- nchar(productive[, "J.REGION"])
productive$FR1.Length <- nchar(productive[, "FR1.IMGT"])
productive$FR2.Length <- nchar(productive[, "FR2.IMGT"])
productive$FR3.Length <- nchar(productive[, "FR3.IMGT"])
productive$FR4.Length <- nchar(productive[, "FR4.IMGT"])
productive$CDR1.Length <- nchar(productive[, "CDR1.IMGT"])
productive$CDR2.Length <- nchar(productive[, "CDR2.IMGT"])
productive$CDR3.Length <- nchar(productive[, "CDR3.IMGT"])

# Extract number-s of AA and AA changed (mutations) from IMGT columns
productive$V.REGION.Nb.of.AA.changes <- as.numeric(
  str_extract(productive[, "V.REGION.Nb.of.AA.changes"], "[0-9]{1,3}"))

productive$FR1.Nb.of.AA.changes <- as.numeric(
  str_extract(productive[, "FR1.IMGT.Nb.of.AA.changes"], "[0-9]{1,3}"))

productive$FR2.Nb.of.AA.changes <- as.numeric(
  str_extract(productive[, "FR2.IMGT.Nb.of.AA.changes"], "[0-9]{1,3}"))

productive$FR3.Nb.of.AA.changes <- as.numeric(
  str_extract(productive[, "FR3.IMGT.Nb.of.AA.changes"], "[0-9]{1,3}"))

productive$CDR1.Nb.of.AA.changes <- as.numeric(
  str_extract(productive[, "CDR1.IMGT.Nb.of.AA.changes"], "[0-9]{1,3}"))

```

```

productive$CDR2.Nb.of.AA.changes <- as.numeric(
  str_extract(productive[, "CDR2.IMGT.Nb.of.AA.changes"], "[0-9]{1,3}")

productive$CDR3.IMGT.Nb.of.AA.changes <- as.numeric(
  str_extract(productive[, "CDR3.IMGT.Nb.of.AA.changes"], "[0-9]{1,3}")

# Extract V(D)J genes
V.Gene.and.allele <- str_extract(productive$V.GENE.and.allele,
  pattern = "[[:alnum:]]{1,6}-(.*?)\\*[0-9]{1,2}")
productive$V.Gene.and.allele <- V.Gene.and.allele

# Check output of extraction
sample(table(V.Gene.and.allele), 3)
# V.Gene.and.allele
# IGHV3-30*10 IGHV7-4-1*02 IGLV1-51*02
#           6           228           5021

V.Gene.Subgroup <- str_extract(productive$V.GENE.and.allele,
  pattern = "[[:alnum:]]{1,6}-(.*?)[:alnum:]{1,2}")
productive$V.Gene.Subgroup <- V.Gene.Subgroup

V.Gene <- str_extract(productive$V.GENE.and.allele,
  pattern = "IG[A-Z]V[0-9]{1,2}")
productive$V.Gene <- V.Gene

J.Gene.and.allele <- str_extract(productive$J.GENE.and.allele,
  pattern = "[[:alnum:]]{1,6}\\*[0-9]{1,2}")
productive$J.Gene.and.allele <- J.Gene.and.allele

J.Gene <- str_extract(productive$J.GENE.and.allele,
  pattern = "IG[A-Z]J[0-9]{1,2}")
productive$J.Gene <- J.Gene

productive2 <- productive

# From productive2 variable, extract only heavy chains (productive)
V.Gene.and.allele <- str_extract(productive$V.GENE.and.allele, "IG[H]V")
productive$HV.Gene.and.allele <- V.Gene.and.allele

# From productive2 variable, extract only light chains (productive2)
V.Gene.and.allele <- str_extract(productive2$V.GENE.and.allele, "IG[K-L]V")
productive2$V.Gene.and.allele <- V.Gene.and.allele
productive2$HV.Gene.and.allele <- NULL

row.has.na2 <- apply(productive2, 1, function(x){any(is.na(x))})

sum(row.has.na2)
# [1] 291131

# Take out the rows with NA (from VH assignment)
productive_lv <- productive2[!row.has.na2,]

nrow(productive_lv)

```



```

# [1] 150409

nrow(productive)
# [1] 441540

row.has.na <- apply(productive, 1, function(x){any(is.na(x))})

productive_hv <- productive[!row.has.na,]
nrow(productive_hv) # Number of productive heavy chains
# [1] 73474

# Extract D gene
D.Gene.and.allele <- str_extract(productive_hv$D.GENE.and.allele,
                                pattern = "[[:alnum:]]{1,6}-(.*?)\\\[0-9]{1,2}")
productive_hv$D.Gene.and.allele <- D.Gene.and.allele

D.Gene.Subgroup <- str_extract(productive_hv$D.GENE.and.allele,
                                pattern = "[[:alnum:]]{1,6}-(.*?)[[:alnum:]]{1,2}")
productive_hv$D.Gene.Subgroup <- D.Gene.Subgroup

D.Gene <- str_extract(productive_hv$D.GENE.and.allele,
                      pattern = "IG[A-Z]D[0-9]{1,2}")
productive_hv$D.Gene <- D.Gene

# Select columns after V(D)J extraction
extracted_hc <- productive_hv[,c("Functionality", "V.Gene.and.allele",
                                "V.Gene.Subgroup", "V.Gene", "D.Gene.and.allele",
                                "D.Gene.Subgroup", "D.Gene", "J.Gene.and.allele",
                                "J.Gene", "V.REGION", "V.D.J.REGION", "JUNCTION",
                                "J.REGION", "FR1.IMGT", "FR2.IMGT", "FR3.IMGT",
                                "FR4.IMGT", "CDR1.IMGT", "CDR2.IMGT", "CDR3.IMGT",
                                "V.REGION.Nb.of.AA.changes", "FR1.IMGT.Nb.of.AA.changes",
                                "FR2.IMGT.Nb.of.AA.changes", "FR3.IMGT.Nb.of.AA.changes",
                                "CDR1.IMGT.Nb.of.AA.changes", "CDR2.IMGT.Nb.of.AA.changes",
                                "CDR3.IMGT.Nb.of.AA.changes", "Donor")]

extracted_lc <- productive_lv[,c("Functionality", "V.Gene.and.allele",
                                "V.Gene.Subgroup", "V.Gene", "J.Gene.and.allele", "J.Gene",
                                "V.REGION", "V.D.J.REGION", "JUNCTION", "J.REGION", "FR1.IMGT",
                                "FR2.IMGT", "FR3.IMGT", "FR4.IMGT", "CDR1.IMGT", "CDR2.IMGT",
                                "CDR3.IMGT", "V.REGION.Nb.of.AA.changes", "FR1.IMGT.Nb.of.AA.changes",
                                "FR2.IMGT.Nb.of.AA.changes", "FR3.IMGT.Nb.of.AA.changes",
                                "CDR1.IMGT.Nb.of.AA.changes", "CDR2.IMGT.Nb.of.AA.changes",
                                "CDR3.IMGT.Nb.of.AA.changes", "Donor")]

# Find rows with NA
row.has.na_hc <- apply(extracted_hc, 1, function(x){any(is.na(x))})

# How many rows have NA?
sum(row.has.na_hc)
# [1] 487

# To check the NAs in the data:

```



```

head(extracted_hc[row.has.na_hc,])
# Note that there was no D gene assigned, where the NA are.

# Exclude (!) rows with NA in the heavy chains data frame
nona_hc <- extracted_hc[!row.has.na_hc,]
nrow(nona_hc) # [1] 72987

# Exclude (!) rows with NA in the light chains data frame
row.has.na_lc <- apply(extracted_lc, 1, function(x){any(is.na(x))})
nona_lc <- extracted_lc[!row.has.na_lc,]

nrow(nona_lc) # [1] 150409

### Some data present "*" in annotated sequences, which are stop codons.
### To filter out VDJ with '*'
# s <- nona_hc$V.D.J.REGION
# my_pattern='\\*'
# strings_without_asterisk <- s[!grep(my_pattern, s)]
# with_asterisk <- grep(my_pattern, s)
# selection <- initial_filtered[-c(with_asterisk),]

### Some data present "X" in an annotated sequences, which indicates
### undetermined aminoacid.
### Exclusion of sequences that have a "X" in the V region or in the CDR3 sequence
# nona_c3X <- str_detect(nona_hc$CDR3.IMGT, pattern = c("X")) #sum(nona_c3X) # [1] 0
# initial <- nona_hc[!nona_c3X,]

initial_hc <- nona_hc

initial_lc <- nona_lc

# Additional CDR3.Length for heavy chains
initial_hc$FR1.Length <- nchar(initial_hc$FR1.IMGT)
initial_hc$FR2.Length <- nchar(initial_hc$FR2.IMGT)
initial_hc$FR3.Length <- nchar(initial_hc$FR3.IMGT)
initial_hc$FR4.Length <- nchar(initial_hc$FR4.IMGT)
initial_hc$CDR1.Length <- nchar(initial_hc$CDR3.IMGT)
initial_hc$CDR2.Length <- nchar(initial_hc$CDR3.IMGT)
initial_hc$CDR3.Length <- nchar(initial_hc$CDR3.IMGT)

# Additional CDR3.Length for light chains
initial_lc$FR1.Length <- nchar(initial_lc$FR1.IMGT)
initial_lc$FR2.Length <- nchar(initial_lc$FR2.IMGT)
initial_lc$FR3.Length <- nchar(initial_lc$FR3.IMGT)
initial_lc$FR4.Length <- nchar(initial_lc$FR4.IMGT)
initial_lc$CDR1.Length <- nchar(initial_lc$CDR3.IMGT)
initial_lc$CDR2.Length <- nchar(initial_lc$CDR3.IMGT)
initial_lc$CDR3.Length <- nchar(initial_lc$CDR3.IMGT)

# Numbers of preprocessed sequences at the different stages
selection_df <- data.frame(summary(selection$Donor)[match_stage])

```

```

productive_hc_df <- data.frame(summary(productive_hv$Donor)[match_stage])
productive_lc_df <- data.frame(summary(productive_lv$Donor)[match_stage])

initial_hc_df <- data.frame(summary(initial_hc$Donor)[match_stage])
initial_lc_df <- data.frame(summary(initial_lc$Donor)[match_stage])

# Counts of CDR3 per donor. Apply table function, to CDR3 sequences according donor
# Calculates the frequency of unique CDR3s
cdr3_count_pre_hc <- lapply(tapply(initial_hc$CDR3.IMG,
                                   initial_hc$Donor, function(x)
                                   sort(table(x), decreasing = TRUE)), function(x) x)

cdr3_count_pre_lc <- lapply(tapply(initial_lc$CDR3.IMG, initial_lc$Donor,
                                   function(x) sort(table(x), decreasing = TRUE)), function(x) x)

# How many unique CDR3 are in list 1?
length(cdr3_count_pre_hc[[1]])
# [1] 4264

# How many unique CDR3 are each donor?
sapply(cdr3_count_pre_hc, length)[match_stage]
# HIV-1 IAVI donor 17      Healthy donor 1
#                3161          4264

sapply(cdr3_count_pre_lc, length)[match_stage]
# HIV-1 IAVI donor 17      Healthy donor 1
#                4203          8093

## Add for table
cdr3_count_pr_hc <- sapply(cdr3_count_pre_hc, length)
cdr3_count_pr_lc <- sapply(cdr3_count_pre_lc, length)

cdr3_count_pr_hc_df <- data.frame(cdr3_count_pr_hc[match_stage])
cdr3_count_pr_lc_df <- data.frame(cdr3_count_pr_lc[match_stage])

# Include only CDR3 that appear more than once in data (exclude singletons)
cdr3_exclusion_hc <- lapply(1:length(cdr3_count_pre_hc), function(x)
                           names(which(cdr3_count_pre_hc[[x]]>1))) #

# How many CDR3 are there after exclusion of singletons in heavy chains?
sapply(cdr3_exclusion_hc, length)
# [1] 1862 1557

cdr3_exclusion_lc <- lapply(1:length(cdr3_count_pre_lc), function(x)
                           names(which(cdr3_count_pre_lc[[x]]>1)))

sapply(cdr3_exclusion_lc, length)
# [1] 3773 2377

# Filter-in from the "initial" stage only the CDR3 that have passed checks
cdr3_exclusion_final_hc <- unlist(lapply(1:length(levels(initial_hc$Donor)), function(x)
  initial_hc$CDR3.IMG[initial_hc$Donor==levels(initial_hc$Donor)[x]] %in%
  cdr3_exclusion_hc[[x]])[match_stage])

```

```

length(cdr3_exclusion_final_hc)
# [1] 72987

cdr3_exclusion_final_lc <- unlist(lapply(1:length(levels(initial_lc$Donor)), function(x)
  initial_lc$CDR3.IMG_T[initial_lc$Donor==levels(initial_lc$Donor)[x]] %in%
  cdr3_exclusion_lc[[x]])[match_stage])

length(cdr3_exclusion_final_hc)
# [1] 72987

cdr3_included_hc <- initial_hc[cdr3_exclusion_final_hc,]

length(cdr3_included_hc[[1]])
# [1] 68981

tapply(cdr3_included_hc$CDR3.IMG_T, cdr3_included_hc$Donor, length)
# Healthy donor 1 HIV-1 IAVI donor 17
#          31800          37181

cdr3_included_lc <- initial_lc[cdr3_exclusion_final_lc,]

tapply(cdr3_included_lc$CDR3.IMG_T, cdr3_included_lc$Donor, length)
# Healthy donor 1 HIV-1 IAVI donor 17
#          60094          84169

## Add for table, exclude singletons (CDR3 that appear once in data)
cdr3_included_hc_df <- data.frame(summary(cdr3_included_hc$Donor)[match_stage])
cdr3_included_lc_df <- data.frame(summary(cdr3_included_lc$Donor)[match_stage])

# Filter for CDR3 that are longer than 2 a.a.
cdr3_length_hc <- nchar(str_trim(cdr3_included_hc$CDR3.IMG_T))>2

length(cdr3_length_hc)
# [1] 68981

cdr3_length_lc <- nchar(str_trim(cdr3_included_lc$CDR3.IMG_T))>2

length(cdr3_length_lc)
# [1] 144263

# All CDR3s that have passed filters but not unique
cdr3_hc <- cdr3_included_hc[cdr3_length_hc,]

tapply(cdr3_hc$CDR3.IMG_T, cdr3_hc$Donor, length)
# Healthy donor 1 HIV-1 IAVI donor 17
#          31800          37181

cdr3_lc <- cdr3_included_lc[cdr3_length_lc,]

#including only cdr3>2aa
cdr3_hc_df <- data.frame(summary(cdr3_hc$Donor)[match_stage])
cdr3_lc_df <- data.frame(summary(cdr3_lc$Donor)[match_stage])

```

```

# Preprocess in unique CDR3s
cdr3_processed <- lapply(tapply(cdr3_hc$CDR3.IGMT, cdr3_hc$Donor,
                              function(x) sort(table(x), decreasing = TRUE)),
                              function(y) y)

# How many unique CDR3s in each dataset?
sapply(cdr3_processed, length)
# Healthy donor 1 HIV-1 IAVI donor 17
#      1862      1557

cdr3_hc <- cdr3_hc[,c("Functionality", "V.Gene.and.allele",
                     "V.Gene.Subgroup", "V.Gene", "J.Gene.and.allele",
                     "J.Gene", "V.REGION", "V.D.J.REGION", "JUNCTION",
                     "J.REGION", "FR1.IGMT", "FR2.IGMT", "FR3.IGMT",
                     "FR4.IGMT", "CDR1.IGMT", "CDR2.IGMT", "CDR3.IGMT",
                     "V.REGION.Nb.of.AA.changes", "FR1.IGMT.Nb.of.AA.changes",
                     "FR2.IGMT.Nb.of.AA.changes", "FR3.IGMT.Nb.of.AA.changes",
                     "CDR1.IGMT.Nb.of.AA.changes", "CDR2.IGMT.Nb.of.AA.changes",
                     "CDR3.IGMT.Nb.of.AA.changes", "FR1.Length", "FR2.Length",
                     "FR3.Length", "FR4.Length", "CDR1.Length", "CDR2.Length",
                     "CDR3.Length", "Donor")]

nrow(cdr3_hc)
# [1] 68981

## Add for table
cdr3_df <- data.frame(summary(cdr3_hc$Donor)[match_stage])

cdr3_processed_df <- data.frame(summary(cdr3_processed)[match_stage])
productive_hc_df

# Build table of numbers of sequences through preprocessing steps
processing_seq <- cbind(selection_df, productive_df, productive_hc_df,
                        initial_hc_df, cdr3_included_hc_df,
                        cdr3_df, cdr3_processed_df)

# Set names of columns
colnames(processing_seq) <- c("All Paired", "All Productive", "Productive HC",
                             "NO Missing, X", "NO Singletons", "CDR3 > 2a.a.",
                             "Unique")

### Write a CSV file of the preprocessed results
write.csv(processing_seq, "processing_seq.csv")

### Make a plot
seqs_df <- data.frame(rownames(processing_seq), processing_seq)

# Eliminate rownames
rownames(seqs_df) <- NULL

# Select columns to show in plot
seqs_df <- seqs_df[,c(1,2,4,8)]
colnames(seqs_df)[1] <- c("Sample")

```

```

# Melt data
melt_seq <- melt(seqs_df, id.vars="Sample")

print(melt_seq)
#       Sample      variable  value
# 1 HIV-1 IAVI donor 17 All.Paired 3104454
# 2   Healthy donor 1 All.Paired 3350792
# 3 HIV-1 IAVI donor 17 Productive.HC 38994
# 4   Healthy donor 1 Productive.HC 34480
# 5 HIV-1 IAVI donor 17      Unique 1557
# 6   Healthy donor 1      Unique 1862

plot_seq <- ggplot(melt_seq, aes(x=variable, y=as.numeric(value)))
plot_seq <- plot_seq + geom_bar(stat="identity", aes(fill=variable), position="dodge") +
  facet_wrap(~Sample, nrow=3) +
  theme_bw() +
  labs(x= "Processing stages", y="Sequence count (log10)") +
  theme(plot.title = element_text(face="bold", size=rel(1), hjust=0),
        plot.margin = unit(c(2, 2, 2, 2), "points"),
        strip.text = element_text(size = rel(0.75))) +
  theme(strip.background = element_rect(fill = scales::alpha("blue", 0.3))) +
  scale_fill_discrete(name="Processing stage") +
  theme(axis.ticks = element_blank(), axis.text.x = element_blank()) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
               labels = trans_format("log10", math_format(10^.x)))

# pdf("figure/processed_seqs.pdf", width=7, height=7)
plot_seq
# dev.off()

# Prints a table that can be imported in LaTeX
print(xtable(processed_seq, caption = "Processed sequences",
             label="tab:process-seqs", digits = 2),
      table.placement="h", include.rownames=TRUE, size = "tiny")

# Stop the clock (in seconds). Code takes ~ 2 minutes filtering.
proc.time() - start
# user      system    elapsed
# 105.194    7.846    119.172

```

	All Paired	All Productive	Productive HC	NO Missing, X	NO Singletons	CDR3 > 2a.a.	Unique
HIV-1 IAVI donor 17	3104454	195242	38994	38785	37181	37181	1557
Healthy donor 1	3350792	246298	34480	34202	31800	31800	1862

Table 1: Processed sequences

## 6.2 Processing

Unique CDR3 are matched to mutations, germline genes (V, D, J) and regions (FRs, CDRs, V region). Results are saved in .RData files that can be loaded and continue other analysis and visualization.

```

process_time <- proc.time()

# Preprocess heavy chains
# Repeat for light chains with: cdr3 <- cdr3_lc
cdr3 <- cdr3_hc

# Mutations by CDR3
mutations_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$V.REGION.Nb.of.AA.changes[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      median_mutations <- median(as.numeric(x), na.rm = TRUE)
      median_mutations
    })

# Generate CDR3 table in order to match with CDR3 mutation-table:
# mutations_by_cdr3_pre
# Match by CDR3
cdr3_table <- tapply(cdr3$CDR3.IMGT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  # order in descending order by CDR3 abundance
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

mutations_by_cdr3 <- lapply(1:length(mutations_by_cdr3_pre), function(x) {
  mutations <- match(cdr3_table[[x]], names(mutations_by_cdr3_pre[[x]]))
  mutations_by_cdr3_pre[[x]][mutations]})

mutations_by_cdr3_stage <- mutations_by_cdr3[match_stage]

# FR1 mutations
fr1_mutations_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$FR1.IMGT.Nb.of.AA.changes[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      median_mutations <- median(as.numeric(x), na.rm = TRUE)
      median_mutations
    })

cdr3_table <- tapply(cdr3$CDR3.IMGT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

fr1_mutations_by_cdr3 <- lapply(1:length(fr1_mutations_by_cdr3_pre), function(x) {
  fr1_mutations <- match(cdr3_table[[x]], names(fr1_mutations_by_cdr3_pre[[x]]))
  fr1_mutations_by_cdr3_pre[[x]][fr1_mutations]})

fr1_mutations_by_cdr3_stage <- fr1_mutations_by_cdr3[match_stage]

# FR2 mutations
fr2_mutations_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)

```

```

    tapply(cdr3$FR2.IMGT.Nb.of.AA.changes[cdr3$Donor==levels(cdr3$Donor)[y]],
           cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
median_mutations <- median(as.numeric(x), na.rm = TRUE)
median_mutations
}))

cdr3_table <- tapply(cdr3$CDR3.IMGT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

fr2_mutations_by_cdr3 <- lapply(1:length(fr2_mutations_by_cdr3_pre), function(x) {
  fr2_mutations <- match(cdr3_table[[x]], names(fr2_mutations_by_cdr3_pre[[x]]))
  fr2_mutations_by_cdr3_pre[[x]][fr2_mutations]})

fr2_mutations_by_cdr3_stage <- fr2_mutations_by_cdr3[match_stage]

# FR3 mutations
fr3_mutations_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$FR3.IMGT.Nb.of.AA.changes[cdr3$Donor==levels(cdr3$Donor)[y]],
         cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
median_mutations <- median(as.numeric(x), na.rm = TRUE)
median_mutations
}))

cdr3_table <- tapply(cdr3$CDR3.IMGT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

fr3_mutations_by_cdr3 <- lapply(1:length(fr3_mutations_by_cdr3_pre), function(x) {
  fr3_mutations <- match(cdr3_table[[x]], names(fr3_mutations_by_cdr3_pre[[x]]))
  fr3_mutations_by_cdr3_pre[[x]][fr3_mutations]})

fr3_mutations_by_cdr3_stage <- fr3_mutations_by_cdr3[match_stage]

# CDR1 mutations
cdr1_mutations_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$CDR1.IMGT.Nb.of.AA.changes[cdr3$Donor==levels(cdr3$Donor)[y]],
         cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
median_mutations <- median(as.numeric(x), na.rm = TRUE)
median_mutations
}))

cdr3_table <- tapply(cdr3$CDR3.IMGT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

cdr1_mutations_by_cdr3 <- lapply(1:length(cdr1_mutations_by_cdr3_pre), function(x) {
  cdr1_mutations <- match(cdr3_table[[x]], names(cdr1_mutations_by_cdr3_pre[[x]]))
  cdr1_mutations_by_cdr3_pre[[x]][cdr1_mutations]})

```

```

cdr1_mutations_by_cdr3_stage <- cdr1_mutations_by_cdr3[match_stage]

# CDR2 mutations
cdr2_mutations_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$CDR2.IGMT.Nb.of.AA.changes[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      median_mutations <- median(as.numeric(x), na.rm = TRUE)
      median_mutations
    })

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

cdr2_mutations_by_cdr3 <- lapply(1:length(cdr2_mutations_by_cdr3_pre), function(x) {
  cdr2_mutations <- match(cdr3_table[[x]], names(cdr2_mutations_by_cdr3_pre[[x]]))
  cdr2_mutations_by_cdr3_pre[[x]][cdr2_mutations]})

cdr2_mutations_by_cdr3_stage <- cdr2_mutations_by_cdr3[match_stage]

# CDR3 MUTATIONS
cdr3_mutations_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$CDR3.IGMT.Nb.of.AA.changes[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      median_mutations <- median(as.numeric(x), na.rm = TRUE)
      median_mutations
    })

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

cdr3_mutations_by_cdr3 <- lapply(1:length(cdr3_mutations_by_cdr3_pre), function(x) {
  cdr3_mutations <- match(cdr3_table[[x]], names(cdr3_mutations_by_cdr3_pre[[x]]))
  cdr3_mutations_by_cdr3_pre[[x]][cdr3_mutations]})

cdr3_mutations_by_cdr3_stage <- cdr3_mutations_by_cdr3[match_stage]

# CDR3 abundance
cdr3_abundance <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  abundance <- table(x)
  # order in descending order by CDR3 abundance
  abundance <- abundance[order(abundance, decreasing = TRUE)]
  abundance
})

cdr3_abundance_stage <- cdr3_abundance[match_stage]

# CDR3 frequency

```



```

cdr3_frequency <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  abundance <- table(x)
  abundance <- abundance[order(abundance, decreasing = TRUE)]
  frequency <- abundance/sum(abundance) *100
  frequency
})

cdr3_frequency_stage <- cdr3_frequency[match_stage]

# V region by CDR3 and donor
vregion_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$V.REGION[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x) {
    # cat(.) # Uncomment this to see a dot printed for each list processed
    v.region <- table(x)
    names(v.region)[which.max(v.region)]
  })

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

vregion_by_cdr3 <- lapply(1:length(vregion_by_cdr3_pre), function(x) {
  vregion <- match(cdr3_table[[x]], names(vregion_by_cdr3_pre[[x]]))
  vregion_by_cdr3_pre[[x]][vregion]})

vregion_by_cdr3_stage <- vregion_by_cdr3[match_stage]

# VDJ region by CDR3 and donor
vdjregion_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$V.D.J.REGION[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
    # cat(.)
    vdj.region <- table(x)
    names(vdj.region)[which.max(vdj.region)]
  })

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

vdjregion_by_cdr3 <- lapply(1:length(vdjregion_by_cdr3_pre), function(x) {
  vdjregion <- match(cdr3_table[[x]], names(vdjregion_by_cdr3_pre[[x]]))
  vdjregion_by_cdr3_pre[[x]][vdjregion]})

vdjregion_by_cdr3_stage <- vdjregion_by_cdr3[match_stage]

# FR1 by CDR3 and donor
fr1_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$FR1.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){

```

```

fr1.region <- table(x)
names(fr1.region)[which.max(fr1.region)]
}))

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  # order in descending order by CDR3 abundance
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

fr1_by_cdr3 <- lapply(1:length(fr1_by_cdr3_pre), function(x) {
  fr1 <- match(cdr3_table[[x]], names(fr1_by_cdr3_pre[[x]]))
  fr1_by_cdr3_pre[[x]][fr1]})

fr1_by_cdr3_stage <- fr1_by_cdr3[match_stage]

# FR2 by CDR3 and donor
fr2_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$FR2.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
    fr2.region <- table(x)
    names(fr2.region)[which.max(fr2.region)]
  })
}))

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

fr2_by_cdr3 <- lapply(1:length(fr2_by_cdr3_pre), function(x) {
  fr2 <- match(cdr3_table[[x]], names(fr2_by_cdr3_pre[[x]]))
  fr2_by_cdr3_pre[[x]][fr2]})

fr2_by_cdr3_stage <- fr2_by_cdr3[match_stage]

# FR3 BY CDR3 AND DONOR
fr3_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$FR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
    # cat(.)
    fr3.region <- table(x)
    names(fr3.region)[which.max(fr3.region)]
  })
}))

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

fr3_by_cdr3 <- lapply(1:length(fr3_by_cdr3_pre), function(x) {
  fr3 <- match(cdr3_table[[x]], names(fr3_by_cdr3_pre[[x]]))
  fr3_by_cdr3_pre[[x]][fr3]})

```

```

fr3_by_cdr3_stage <- fr3_by_cdr3[match_stage]

# CDR1 by CDR3 and donor
cdr1_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$CDR1.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      cdr1.region <- table(x)
      names(cdr1.region)[which.max(cdr1.region)]
    })
))

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

cdr1_by_cdr3 <- lapply(1:length(cdr1_by_cdr3_pre), function(x) {
  cdr1 <- match(cdr3_table[[x]], names(cdr1_by_cdr3_pre[[x]]))
  cdr1_by_cdr3_pre[[x]][cdr1]})

cdr1_by_cdr3_stage <- cdr1_by_cdr3[match_stage]

# CDR2 by CDR3 and donor
cdr2_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$CDR2.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      cdr2.region <- table(x)
      names(cdr2.region)[which.max(cdr2.region)]
    })
))

cdr3_table <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  names(cdr3.table)})

cdr2_by_cdr3 <- lapply(1:length(cdr2_by_cdr3_pre), function(x) {
  cdr2 <- match(cdr3_table[[x]], names(cdr2_by_cdr3_pre[[x]]))
  cdr2_by_cdr3_pre[[x]][cdr2]})

cdr2_by_cdr3_stage <- cdr2_by_cdr3[match_stage]

# V gene assignment to unique CDR3 analysed
vgene_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$V.Gene[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      v.gene <- table(x)
      names(v.gene)[which.max(v.gene)]
    })
))

vgene_by_cdr3 <- lapply(1:length(vgene_by_cdr3_pre), function(x) {
  table.vgene <- match(cdr3_table[[x]], names(vgene_by_cdr3_pre[[x]]))
  vgene_by_cdr3_pre[[x]][table.vgene]
})

```

```

})

vgene_by_cdr3_stage <- vgene_by_cdr3[match_stage]

# V gene subgroup assignment to unique CDR3 analysed
vgene_subgr_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$V.Gene.Subgroup[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      v.gene.sub <- table(x)
      names(v.gene.sub)[which.max(v.gene.sub)]
    })
})

vgene_subgr_by_cdr3 <- lapply(1:length(vgene_subgr_by_cdr3_pre), function(x) {
  match_by_cdr3 <- match(cdr3_table[[x]], names(vgene_subgr_by_cdr3_pre[[x]]))
  vgene_subgr_by_cdr3_pre[[x]][match_by_cdr3]
})

vgene_subgr_by_cdr3_stage <- vgene_subgr_by_cdr3[match_stage]

# V gene and allele assignment to unique CDR3 analysed
vgene_allele_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$V.Gene.and.allele[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      v.gene.allele <- table(x)
      names(v.gene.allele)[which.max(v.gene.allele)]
    })
})

vgene_allele_by_cdr3 <- lapply(1:length(vgene_allele_by_cdr3_pre), function(x) {
  match_by_cdr3 <- match(cdr3_table[[x]], names(vgene_allele_by_cdr3_pre[[x]]))
  vgene_allele_by_cdr3_pre[[x]][match_by_cdr3]
})

vgene_allele_by_cdr3_stage <- vgene_allele_by_cdr3[match_stage]

# J gene assignment to unique cdr3s analysed
jgene_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)
  tapply(cdr3$J.Gene[cdr3$Donor==levels(cdr3$Donor)[y]],
    cdr3$CDR3.IMGT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
      v.gene <- table(x)
      names(v.gene)[which.max(v.gene)]
    })
})

jgene_by_cdr3 <- lapply(1:length(jgene_by_cdr3_pre), function(x) {
  jgene <- match(cdr3_table[[x]], names(jgene_by_cdr3_pre[[x]]))
  jgene_by_cdr3_pre[[x]][jgene]
})

jgene_by_cdr3_stage <- jgene_by_cdr3[match_stage]

# J gene and allele assignment to unique cdr3s analysed
jgene_allele_by_cdr3_pre <- lapply(1:length(levels(cdr3$Donor)), function(y)

```

```

tapply(cdr3$J.Gene.and.allele[cdr3$Donor==levels(cdr3$Donor)[y]],
       cdr3$CDR3.IGMT[cdr3$Donor==levels(cdr3$Donor)[y]], function(x){
  j.gene.allele <- table(x)
  names(j.gene.allele)[which.max(j.gene.allele)]
}))

jgene_allele_by_cdr3 <- lapply(1:length(jgene_allele_by_cdr3_pre), function(x) {
  j.gene.allele <- match(cdr3_table[[x]], names(jgene_allele_by_cdr3_pre[[x]]))
  jgene_allele_by_cdr3_pre[[x]][j.gene.allele]
})

jgene_allele_by_cdr3_stage <- jgene_allele_by_cdr3[match_stage]

# CDR3 rank
cdr3_table_rank <- tapply(cdr3$CDR3.IGMT, cdr3$Donor, function(x) {
  cdr3.table <- table(x)
  # Order in descending order by CDR3 abundance
  cdr3.table <- cdr3.table[order(cdr3.table, decreasing = TRUE)]
  cdr3.table/sum(cdr3.table) *100
})

cdr3_table_rank_stage <- cdr3_table_rank[match_stage]

save(cdr3_table_rank_stage, file="cdr3_table_rank_stage.RData")

#####
##### PARAMETERS BY CDR3 AA #####

# Put all the values into a list
working_list <- list()

for(k in 1:length(cdr3_table_rank_stage)){
  working_list[[k]] <- data.frame(cdr3 = names(cdr3_table_rank_stage[[k]]),
                                cdr3_rank = c(1:length(cdr3_table_rank_stage[[k]])),
                                cdr3_frequency = as.numeric(cdr3_frequency_stage[[k]]),
                                cdr3_abundance = as.numeric(cdr3_abundance_stage[[k]]),
                                vregion_mutations = mutations_by_cdr3_stage[[k]],
                                fr1_mutations = fr1_mutations_by_cdr3_stage[[k]],
                                fr2_mutations = fr2_mutations_by_cdr3_stage[[k]],
                                fr3_mutations = fr3_mutations_by_cdr3_stage[[k]],
                                cdr1_mutations = cdr1_mutations_by_cdr3_stage[[k]],
                                cdr2_mutations = cdr2_mutations_by_cdr3_stage[[k]],
                                cdr3_mutations = cdr3_mutations_by_cdr3_stage[[k]],
                                vgeneallele = vgene_allele_by_cdr3_stage[[k]],
                                vgene = vgene_by_cdr3_stage[[k]],
                                vgene_subgroup = vgene_subgr_by_cdr3_stage[[k]],
                                jgeneallele = jgene_allele_by_cdr3_stage[[k]],
                                jgene = jgene_by_cdr3_stage[[k]],
                                vregion_by_cdr3 = vregion_by_cdr3_stage[[k]],
                                fr1_by_cdr3 = fr1_by_cdr3_stage[[k]],
                                fr2_by_cdr3 = fr2_by_cdr3_stage[[k]],
                                fr3_by_cdr3 = fr3_by_cdr3_stage[[k]],

```

```

        cdr1_by_cdr3 = cdr1_by_cdr3_stage[[k]],
        cdr2_by_cdr3 = cdr2_by_cdr3_stage[[k]],
        cdr1_length = nchar(cdr1_by_cdr3_stage[[k]]),
        cdr2_length = nchar(cdr2_by_cdr3_stage[[k]]),
        cdr3_length = nchar(names(cdr3_table_rank_stage[[k]])),
        fr1_length = nchar(fr1_by_cdr3_stage[[k]]),
        fr2_length = nchar(fr2_by_cdr3_stage[[k]]),
        fr3_length = nchar(fr3_by_cdr3_stage[[k]]))}

names(working_list) <- names(cdr3_table_rank_stage)

# Stop the clock (in seconds). Code takes less than 2 minutes for processing.
proc.time() - process_time
# user      system    elapsed
# 17.364      0.621     27.921

# save(working_list, file = "working_list.RData")

# This procedure was followed for each sample separately for the workshop.
# The minimal code for one dataset is provided with the materials of this course:
# "initial_analysis.R"

```

## 7 Immune repertoire analysis

Immune repertoire analysis [16] is performed on complex systems, with hundreds of thousands or more sequencing (depending on sequencing depth, technological and biological sampling). It is a common challenge to choose appropriate graphics for the visualization of large-scale and high-dimensional data. The following sections will cover some of the general analysis (CDR3 frequency, showing clonal expansion), somatic hypermutations (SHM), CDR3 lengths, V(D)J germline analysis, and a few mining techniques to order to select clones of interest. Clonality processing (where CDR3 clones are grouped usually by same V and J genes and CDR3 length) is performed to correct for PCR/sequencing errors [16]. For the reader reference, a new clonotyping method through IMGT is also available: IMGT/StatClonotype: Statistical analysis from IMGT/HighV-QUEST output ([17]). Here, analysis of clones based on exact CDR3 amino acid sequences (100% identity, Figure 10) or after clustering in clonotypes with 90% a.a. identity (e.g., Figure 11) is shown.

### 7.1 Diversity

CDR3 clonal frequencies in each repertoire are profiles of its clonal selection and expansion. Although to-date exist several methods to calculate diversity a repertoire [18], here the focus is on CDR3 frequency plots and CDR3 cumulative frequency (Figure 10). Of note that this analysis is based on exact amino acid sequences (100% identity)

```

# Load in data calculated previously
load("cdr3_table_rank_stage.RData")

### Plot frequency and cumulative frequency
require(scales)

# Get titles for plot

```

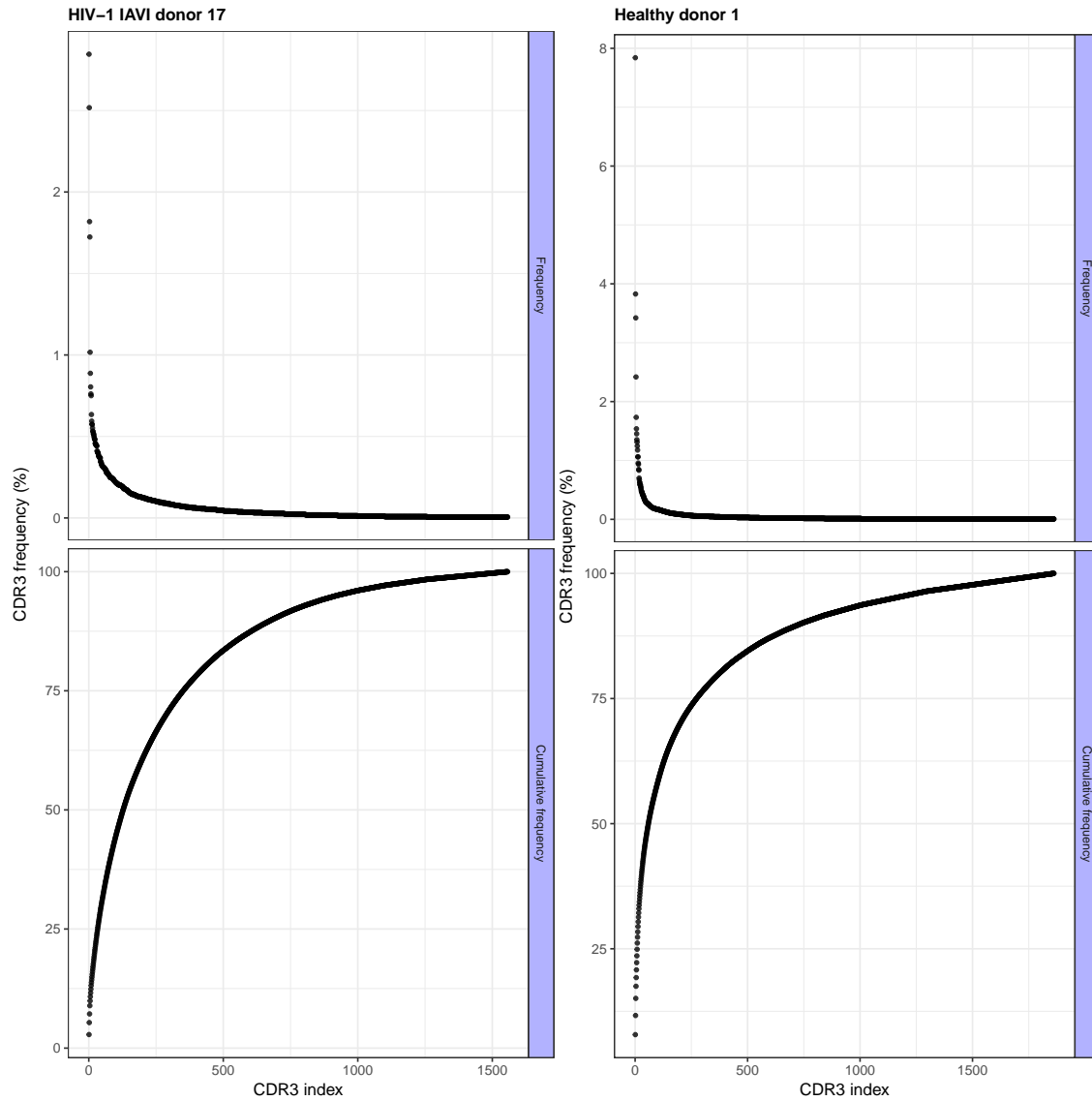


Figure 10: CDR3 frequency and cumulative frequency.

```
titles <- c(levels(cdr3_hc$Donor)[match_stage])

for (i in 1:length(cdr3_table_rank_stage)){

  cdr3_freq <- data.frame(as.numeric(cdr3_table_rank_stage[[i]]))
  cdr3_freq$index <- 1:nrow(cdr3_freq)
  colnames(cdr3_freq)[1] <- "V1"
  cdr3_freq <- rbind(cdr3_freq, data.frame(V1=cumsum(cdr3_freq$V1),
                                             index=1:nrow(cdr3_freq)))
  cdr3_freq$facet <- factor(rep(factor(c("Frequency", "Cumulative frequency")),
                                each=length(cdr3s_table_rank_stages[[i]]),
                                levels = c("Frequency", "Cumulative frequency")))

  rank_plots[[i]] <- ggplot(cdr3_freq, aes(x=index, y= V1))
```

```

rank_plots[[i]] <- rank_plots[[i]] + geom_point(size=1, alpha = 0.8) +
  facet_grid(facet ~., scales = "free") +
  labs(x= "CDR3 index", y="CDR3 frequency (%)", title = titles[i]) +
  theme_bw() +
  theme(plot.title = element_text(face="bold", size=rel(1), hjust=0),
        plot.margin = unit(c(2, 2, 2, 2),"points"),
        strip.text = element_text(size = rel(0.75)),
        legend.position = "none") +
  theme(strip.background = element_rect(fill = scales::alpha("blue", 0.3)))
}

# pdf("figure/rankplots.pdf", width = 10, height=10)
pushViewport(viewport(layout=grid.layout(1,2), height=1,width=1))
vplayout<-function(x,y)viewport(layout.pos.row=x,layout.pos.col=y)
print(rank_plots[[1]],vp=vplayout(1,1))
print(rank_plots[[2]],vp=vplayout(1,2))
# dev.off()

# CDR3 frequency plot
cdr3_freq_hc <- data.frame(as.numeric(cdr3_table_rank_stage$`HIV-1 IAVI donor 17`))
cdr3_freq_hc$index <- 1:nrow(cdr3_freq_hc)
colnames(cdr3_freq_hc)[1] <- "V1"

freq_plots <- ggplot(cdr3_freq_hc, aes(x=index, y= V1))
freq_plots <- freq_plots + geom_point(size=1, alpha = 0.8) +
  labs(x= "CDR3 index", y= "CDR3 frequency (%)",
       title = "CDR3 frequency in HIV-1 IAVI donor 17") +
  theme_bw() +
  theme(plot.title = element_text(face="bold", size=rel(1), hjust=0),
        plot.margin = unit(c(2, 2, 2, 2),"points"),
        strip.text = element_text(size = rel(0.75)),
        legend.position = "none") +
  theme(strip.background = element_rect(fill = scales::alpha("blue", 0.3)))

# pdf("figure/frequency_iavi17.pdf", width = 7, height=7)
print(freq_plots)
# dev.off()

```

## 7.2 Somatic hypermutations (SHM)

Somatic hypermutations distributions in 6 samples are calculated after clonotyping for 90% a.a. identity. The clonotyping code is not shown because it is not in the scope of this workshop to go into detail into the computational methods to perform clonotyping. The precalculated data is though provided.

```

load("clon90")

mean_vmutations <- tapply(clon90$mean_mutations, clon90$L1, function(x)
  mean(x, na.rm=TRUE))

median_vmutations <- tapply(clon90$mean_mutations, clon90$L1, function(x)
  median(x, na.rm=TRUE))

```



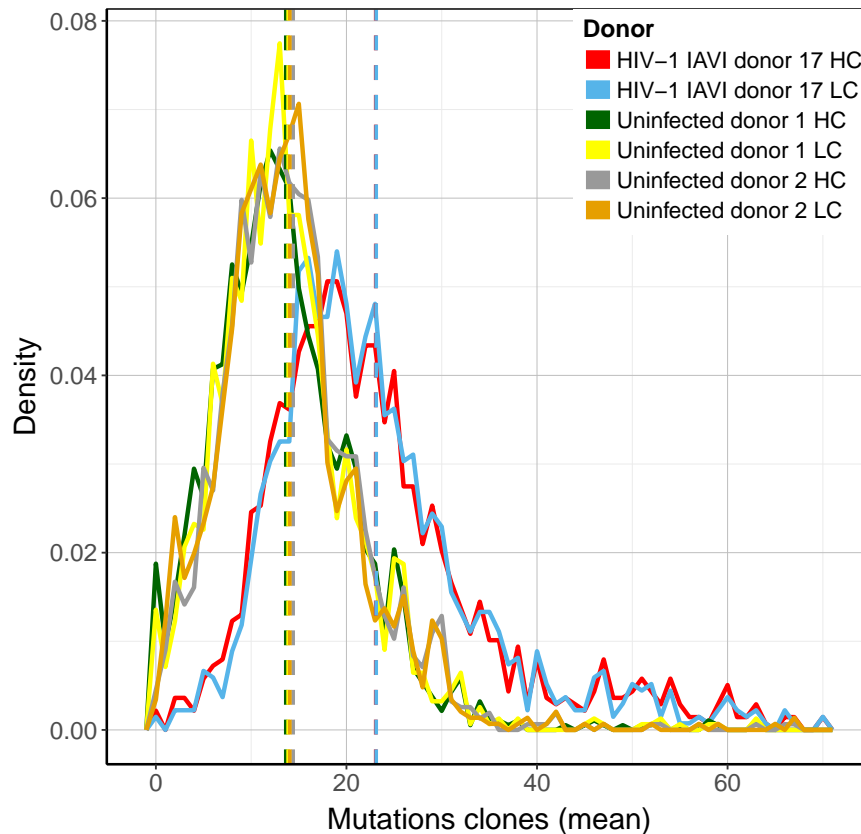


Figure 11: Somatic hypermutation density in clonotypes with 90% CDR3 a.a. identity.

```
# pdf("figure/density_shm_clon90.pdf") # Uncomment to produce plot
ggplot(clon90) +
  geom_freqpoly(binwidth=1, aes(x=mean_mutations, y=..density.., colour=L1), size=1.2) +
  scale_color_manual("Donor", values=c("red",
                                         "#56B4E9", "darkgreen", "yellow", "#999999", "#E69F00"),
                    labels=c("HIV-1 IAVI donor 17 HC", "HIV-1 IAVI donor 17 LC",
                              "Uninfected donor 1 HC", "Uninfected donor 1 LC",
                              "Uninfected donor 2 HC", "Uninfected donor 2 LC")) +
  theme(axis.text.x = element_text(angle=360, size=5)) +
  theme_bw(base_size=12, base_family = "Helvetica") +
  labs (title = "", y="Density", x="Mutations clones (mean)") +
  theme(strip.background = element_rect(fill="white",
                                         linetype="solid", color="white"),
        strip.text=element_text(face="bold", size=rel(1.2), hjust=0.5, vjust=1)) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(plot.title=element_text(family="Helvetica",
                                 face="bold", size=rel(1.5), hjust=-0.035, vjust=3.5)) +
  theme(axis.text=element_text(family="Helvetica", size=rel(1.2)),
        axis.title=element_text(family="Helvetica", size=rel(1.2)),
        legend.text=element_text(family="Helvetica", size=rel(1.1)),
        legend.title=element_text(family="Helvetica", face="bold", size=rel(1.2))) +
```

```

theme(axis.title.x = element_text(vjust=-0.5, size=rel(1.2)),
      axis.title.y = element_text(vjust=1, size=rel(1.2))) +
theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
      axis.line=element_line(size = 0.7, color = "black"),
      text = element_text(size = 14)) +
theme(plot.margin = unit(c(0, 0.2, 0.5, 0.2),"cm")) +
guides(colour = guide_legend(override.aes = list(size=5))) +
theme(strip.text.x = element_text(size = 12, colour = "black", angle = 0)) +
theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
      axis.line = element_line(size = 0.7, color = "black"),
      text = element_text(size = 12))+ theme(legend.position=c(0.805,0.85)) +
geom_vline(xintercept=mean_vmutations, linetype="dashed", size=1,
          color=c("red", "#56B4E9", "darkgreen", "yellow", "#999999", "#E69F00"))
# dev.off()

```

### 7.3 CDR3 length distribution

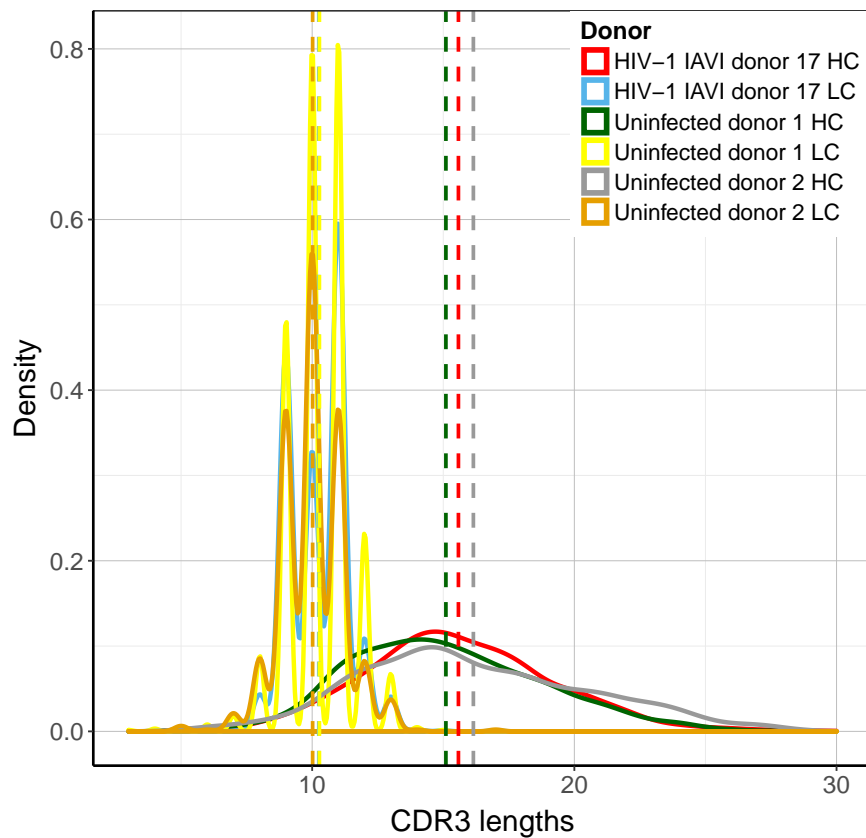


Figure 12: CDR3 lengths density in clonotypes with 90% CDR3 a.a. identity.

```

load("clon90")

mean_cdr3_lengths <- tapply(clon90$cdr3_length, clon90$L1, function(x)
  mean(x, na.rm=TRUE))

```

```

median_cdr3_lengths <- tapply(clon90$cdr3_length, clon90$L1, function(x)
  median(x, na.rm=TRUE))

pdf("figure/geom_density_cdr3_lengths_clon90.pdf")
ggplot(clon90, aes(x=cdr3_length, color=L1)) + geom_density(size=1.2) +
  scale_color_manual("Donor", values=c("red", "#56B4E9", "darkgreen",
    "yellow", "#999999", "#E69F00"),
    labels=c("HIV-1 IAVI donor 17 HC", "HIV-1 IAVI donor 17 LC",
      "Uninfected donor 1 HC", "Uninfected donor 1 LC",
      "Uninfected donor 2 HC", "Uninfected donor 2 LC")) +
  theme(axis.text.x = element_text(angle=360, size=5)) +
  theme_bw(base_size=12, base_family = "Helvetica") +
  labs (title = "", y="Density", x="CDR3 lengths") +
  theme(strip.background = element_rect(fill="white", linetype="solid", color="white"),
    strip.text=element_text(face="bold", size=rel(1.2), hjust=0.5, vjust=1)) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(plot.title=element_text(family="Helvetica", face="bold", size=rel(1.5),
    hjust=-0.035, vjust=3.5)) +
  theme(axis.text=element_text(family="Helvetica", size=rel(1.2)),
    axis.title=element_text(family="Helvetica", size=rel(1.2)),
    legend.text=element_text(family="Helvetica", size=rel(1.1)),
    legend.title=element_text(family="Helvetica", face="bold", size=rel(1.2))) +
  theme(axis.title.x = element_text(vjust=-0.5, size=rel(1.2)),
    axis.title.y = element_text(vjust=1, size=rel(1.2))) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
    axis.line=element_line(size = 0.7, color = "black"),
    text = element_text(size = 14)) +
  theme(plot.margin = unit(c(0, 0.2, 0.5, 0.2), "cm")) +
  guides(colour = guide_legend(override.aes = list(size=5))) +
  theme(strip.text.x = element_text(size = 12, colour = "black", angle = 0)) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
    axis.line = element_line(size = 0.7, color = "black"),
    text = element_text(size = 12)) + theme(legend.position=c(0.805,0.85)) +
  geom_vline(xintercept=mean_cdr3_lengths, linetype="dashed", size=1,
    color=c("red", "#56B4E9", "darkgreen", "yellow", "#999999", "#E69F00"))
dev.off()

```

## 7.4 V(D)J gene and allele frequency

The following code produces the V gene frequency shown in Figure 13. In the same way, D and J gene frequencies can be visualized. Figure 14 shows the V-J combinations obtained using the R package RCircos [19].

```

load("working_hiv1_iavi17.RData")

# Germline genes - V gene frequency analysis

vgene_freq <- prop.table(table(working_hiv1_iavi17$vgene_subgroup))

vgene_plot <- ggplot(data.frame(vgene_freq), aes(x=Var1, y=Freq*100)) +
  theme_bw() + geom_bar(stat="identity", width=0.5, fill="dodgerblue4") +

```

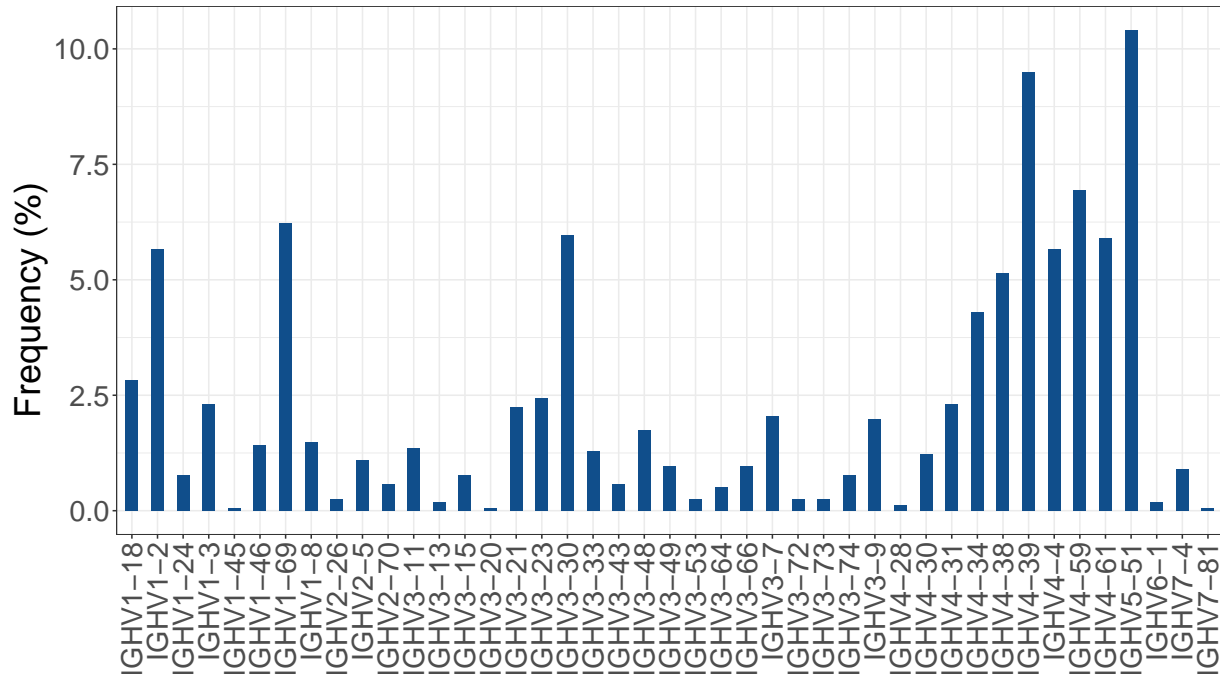


Figure 13: V gene frequency in HIV-1 IAVI donor 17.

```
labs(x="", y="Frequency (%)") +
  theme(axis.title.x = element_text(vjust=-0.5),
        axis.title.y = element_text(vjust=1)) +
  theme(axis.text.x=element_text(angle=90, size=20, hjust=1, vjust=0.5)) +
  theme(axis.title.y = element_text(colour="black", size=25, vjust=1.5, hjust=0.4),
        axis.text.y = element_text(angle=0, vjust=0.5, size=20)) +
  theme(plot.margin = unit(c(12, 12, 12, 12),"points"))

pdf("figure/vgene_frequency.pdf", width=12)
vgene_plot
dev.off()
```

## 7.5 Mining antibody repertoires: selection techniques

The following code shows how to select for top frequency CDR3 in a dataset, and how to subset for different parameters.

```
# Load data
load("working_hiv1_iavi17.RData")

# As we have ordered our dataframes by CDR3 ranks,
# we can subset the 100 most frequent clones by
top_frequency_cdr3 <- as.character(head(working_hiv1_iavi17[["cdr3"]], 100))

# In order to select all the columns of the top 100 most frequent
# CDR3s in the original dataset:
```

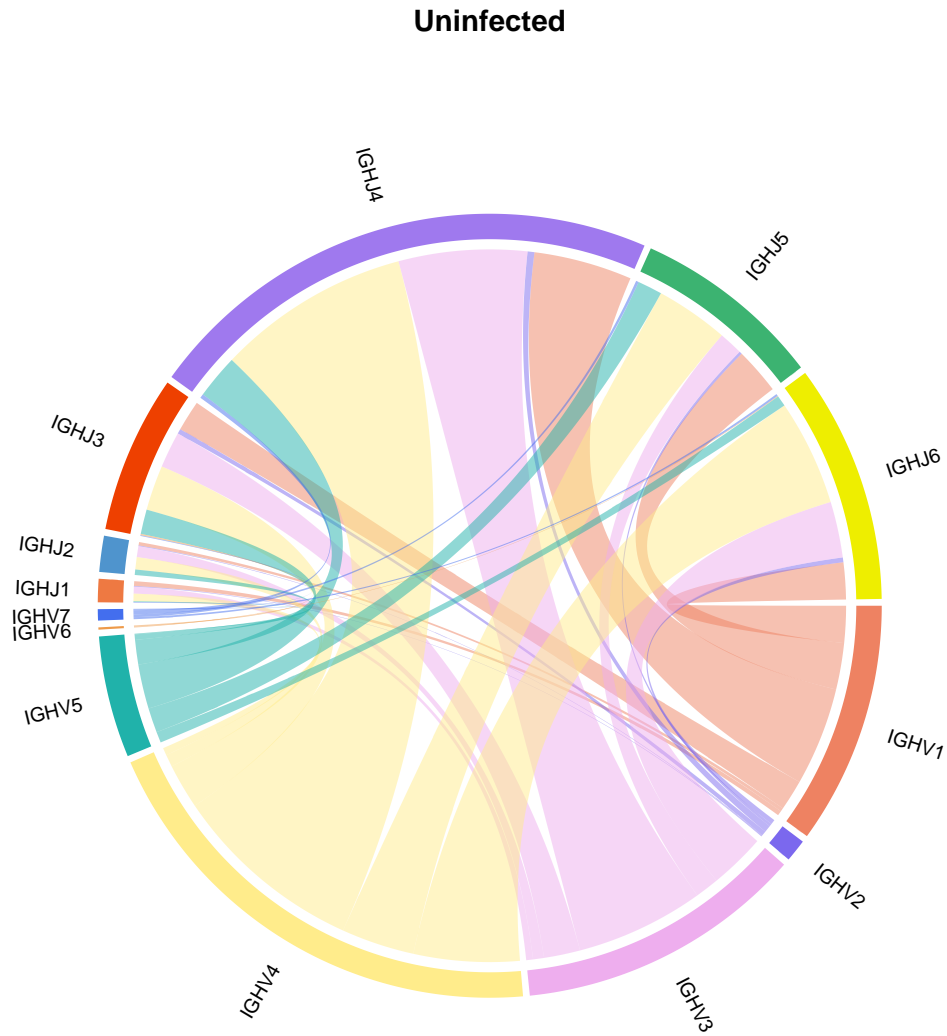


Figure 14: V-J combinations in HIV-1 IAVI donor 17.

```
all_top_cdr3 <- working_hiv1_iavi17[top_frequency_cdr3,]

# Calculate the median mutations in this subset of CDR3s
median(all_top_cdr3$vregion_mutations)
# [1] 21

# FR2 for CDR3 that have V gene IGHV1-2 in the top 100 most frequent CDR3s
data.frame(fr2=all_top_cdr3[vgene_subgroup=="IGHV1-2",]$fr2)
#
# AKGLNRYSDWSFGT    VHWVRQAPGQGLEYMGW
```

```

# ARAWHPYYFDY      LHWVRQAPGQGLEWLGW
# ARENYFDY          IHWVRQAPGQGFEWMGR
# ARDLCTGGSCYYFDY  MYWVRQAPGQGPEWMGR
# ARGKQLRLGDLSPDFDY MHWVRHVPGQGLEWMGW
# ARDYCSGGTCRLPHY  TTRYVLLYCSSRGFVLA

# A faster way is by using the subset function.
# We subset the fr3 in the original dataframe and V genes
# by the clones that have CDR3 longer than 27 a.a.
subset(working_hiv1_iavi17, cdr3_length > 27, select = c(fr3, vgene_subgroup))
#                                     fr3 vgene_subgroup
# TRELIVEEPPNCGTANCDVNIVSYEFDL    KYAQRFGSGVTMTRDMSISTGFIEVERLTSDDTAVYYC    IGHV1-2
# VRDGAYGCSGASCYFGALGNFVYYYMDV    FYADSVKGRFTISRHNANNSLDLHINSLRAEDTAVYYC    IGHV3-11
# ARSYCSSPSCYMGYVDPEDYYSYYMDV    YYADSVKGRFTISRDNKDSLFLQMDSLRVDDTAVYFC    IGHV3-48
# ARGRREQLLLPLKATRPRTVESERNNWFD    NYKSSLKSRVIISVDSTNQFSLRLSSVTAVDTAVYYC    IGHV4-34
# ARGRKSKGAKDFWSDMGFYSYFYDMDV    NYNPSLKSRTLLVDPSKNQFSLKMTSVTAADTAVYYC    IGHV4-34

# Select for the "CDR3" with all its characteristics (V gene, CDR3 frequency, etc)
working_hiv1_iavi17[working_hiv1_iavi17$cdr3 == "VRDGAYGCSGASCYFGALGNFVYYYMDV",]

# Find sequence in dataset
which(working_hiv1_iavi17$cdr3 == "VRDGAYGCSGASCYFGALGNFVYYYMDV")
# [1] 210

# Find similar sequences
cdr3_distance <- stringDist(as.character(working_hiv1_iavi17$cdr3),
                           method = "levenshtein")

# Assign 1 to all similar sequences (max. 1 a.a. different)
# and zero to all sequences that differ by more than 1 a.a.
cdr3_matrix <- ifelse(as.matrix(cdr3_distance) == 1, 1, 0)
colnames(cdr3_matrix) <- as.character(working_hiv1_iavi17$cdr3)
rownames(cdr3_matrix) <- as.character(working_hiv1_iavi17$cdr3)

which(cdr3_matrix[rownames(cdr3_matrix)=="ARRGIAGPDYYSYHGLDV"]==1)
# [1] 52 106 122 1186 1436 1437 1438 1552

```

## 7.6 Saving output and reproducibility

As shown in the previous sections, analyzed data can be saved as RData and reloaded for further analysis (a short summary of examples follows). knitr code assures that when the input data is changed, the following analysis and calculations are updated. Thus, reports with updated values in text will be produced.

```

# Example loading preserved data

load("working_hiv1_iavi17.RData")

# Example of saving results as RData

subset_fr3 <- subset(working_hiv1_iavi17, cdr3_length > 27, select = c(fr3, vgene_subgroup))

save(subset_fr3, file="subset_fr3.RData")

```

```

# Example of writing results as CSV

top_cdr3 <- as.character(head(working_hiv1_iavi17[["cdr3"]], 100))

write.csv(top_cdr3, "top_cdr3.csv")

# Example of loading and reading-in previous results

load("subset_fr3.RData")

# Print the first row of the data

print(subset_fr3[1,])

##
## TRELIVEEPPNCGTANCDVNIVSYEFDL KYAQRFSQSGVTMTRDMSISTGFIEVERLTSDDTAVYYC fr3 vgene_subgroup IGHV1-2
nrow(subset_fr3)
## [1] 5

```

For example, the number of FR3 subset with the following code can be incorporated in the text as (5) and this value will automatically be updated if the value from the input data changes.

```

working_list <- list(working_hiv1_iavi17, working_hiv1_iavi17_lc)

# What is the maximal length of CDR3s?
max(sapply(working_list, function(x) max(x[, "cdr3_length"])))
# [1] 30

# Group all CDR3 of length 7 a.a.
length11 <- lapply(working_list, function(x) x[x$cdr3_length=="11",])

# Find the distance between all CDR3 of length 7 a.a.
dist_11 <- lapply(length11, function(x) stringDist(as.character(row.names(x)),
method = "levenshtein"))

# Cluster the CDR3
hc11 <- lapply(dist_11, function(x) hclust(x))

# Find distance of 90% a.a. similarity of CDR3s of length 7
h11_dist <- nchar(as.character(row.names(length7[[1]])))[1] -
0.9*nchar(as.character(row.names(length7[[1]])))[1]

# Group CDR3s of 90% a.a. similarity
orig_into_clust11 <- lapply(1:length(hc11), function(x)
tapply(as.character(row.names(length7[[x]])),
cutree(hc11[[x]], h = h11_dist), function(x) x ))

# orig_into_clust11 is a list of length 2.

# orig_into_clust11[[1]] contains CDR3s grouped by length =11 a.a.
# and 90% identical from HIV-1 IAVI donor 17

# orig_into_clust11[[2]] from the Healthy donor

```

```
# How many groups of CDR3 that share 90% sequence identity to one another?
length(orig_into_clust11[[1]])
# [1] 68

# How is the second group composed?
orig_into_clust11[[1]][2]
# $`2`
# [1] "ARAWHPYYFDY" "ARAWRPYYFDY"
```



## Part III

# Advanced Topics

## 8 Align to database and identify sequence of interest

To identify sequences of interest from alignment with quality scores, a query (e.g, vdj\_sequences.fasta) and database of reference sequences (database.fasta) can be built. The alignment can be performed using blastp running the following line in Terminal after building query and database fasta files.

```
$ /Users/enkelejda/ncbi-blast-2.2.31+/bin/blastp
-query /Users/enkelejda/ncbi-blast-2.2.31+/data/vdj_sequences.fasta
-db /Users/enkelejda/ncbi-blast-2.2.31+/data/database.fasta
-out /Users/enkelejda/ncbi-blast-2.2.31+/data/alignment.txt -outfmt 7
```

Top aligned sequences can be then selected using an adaption of the methods explained here. For example, following this procedure and building a broadly neutralizing antibody (bNAbs) database, we found the top VDJ aligned sequence shown in Figure 15 from the HIV-1 IAVI donor 17 that has developed bNAbs.

After aligning the full VDJ sequences to the in-house bNAb database [20], top-hit CDR3 , Figure 15, was retrieved. Mutations with respect to the hit bNAb sequence are shown in blue. CD regions are underlined, framework regions are not underlined.

bNAb 10-1074 QVQLQESGPGLVKPSSETLSVTCSVSGDSMNYY  
VDJ836 query AAMAGTSGPGLVKPSETLSVTCSVSGDSMNYY

bNAb 10-1074 WTWIRQSPGKGLEWIGYISDRPSATYNPSLNSRVVISRDTSKNQLSLKLSVTPADTAVYYC  
VDJ836 query WTWIRQSPGKGLEWIGYISDRPSATYNPSLNSRVVISRDTSKNQLSLKLSVTPADTAVYYC

bNAb 10-1074 RIYGVVSFGEFFYYYSMDVWGKGTITVSSATARRGQ  
VDJ836 query ATARRGQRIYGVVSFGEFFYYYSMDVWGKGTITVSS

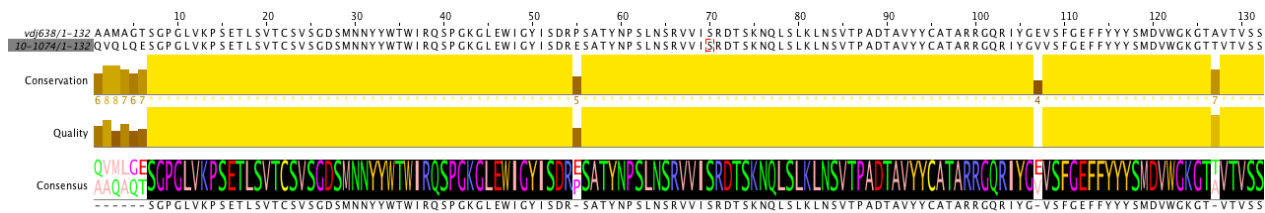


Figure 15: bNAb selection from top aligned sequence from high-throughput sequencing data to broadly neutralizing database.

## 9 Visualization of high-dimensional data

Data from immune repertoires can be high-dimensional with different parameters measured for each clone in the repertoire. Sometimes it is useful to understand more than one parameter (e.g., more than the CDR3 frequency) and the relation of this to other parameters (e.g., somatic hypermutations and CDR3 lengths). This would allow to answer questions like: Are highest-frequency clones more or less mutated than the rest of the clones in the repertoire? How is the mutation in these highly mutated clones related to their CDR3 length? Are clones with a longer CDR3 more mutated?

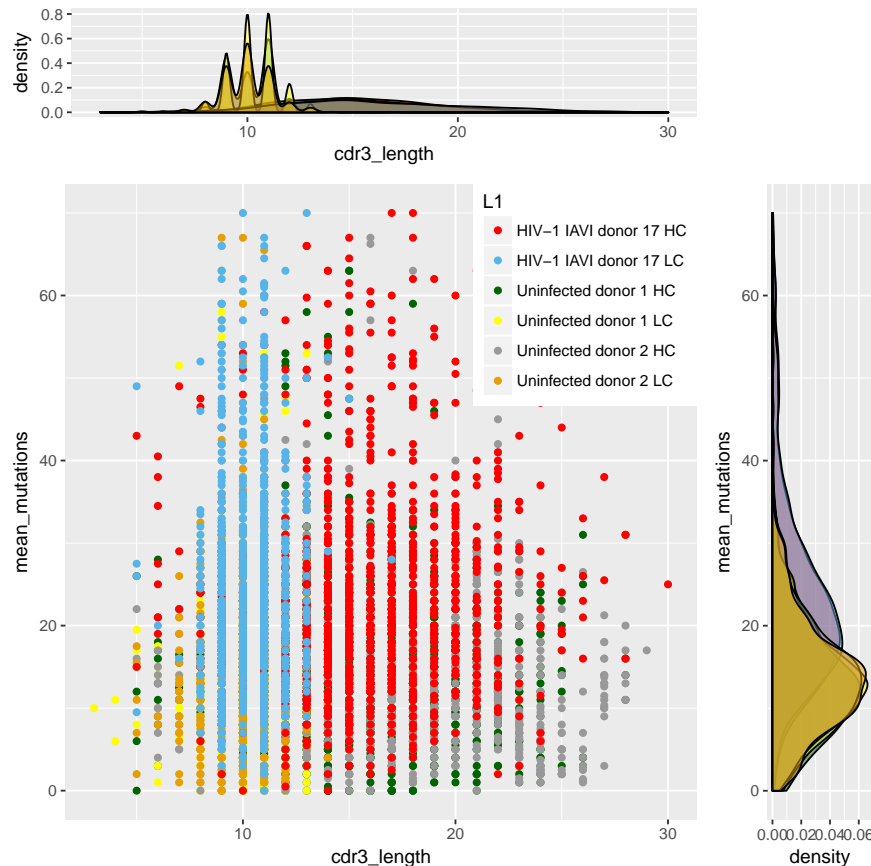


Figure 16: CDR3 frequency, lengths and somatic hypermutation density in clonotypes with 90% CDR3 a.a. identity.

In Figure 16 the CDR3 frequencies, lengths and somatic hypermutations are shown for all 6 samples based on clonotypes of 90% CDR3 identity, the parameters for CDR3 clones based on 100% CDR3 identity are shown in Figure 17 .

```
# Load clones based on 90 % CDR3 a.a. identity
load("clon90")

library(e1071)

skewness(clon90[clon90$L1=="HIV-1 IAVI donor 17 HC",]$mean_mutations)
# [1] 1.301486

#placeholder plot - prints nothing at all
empty <- ggplot()+geom_point(aes(1,1), colour="white") +
  theme(
    plot.background = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
```

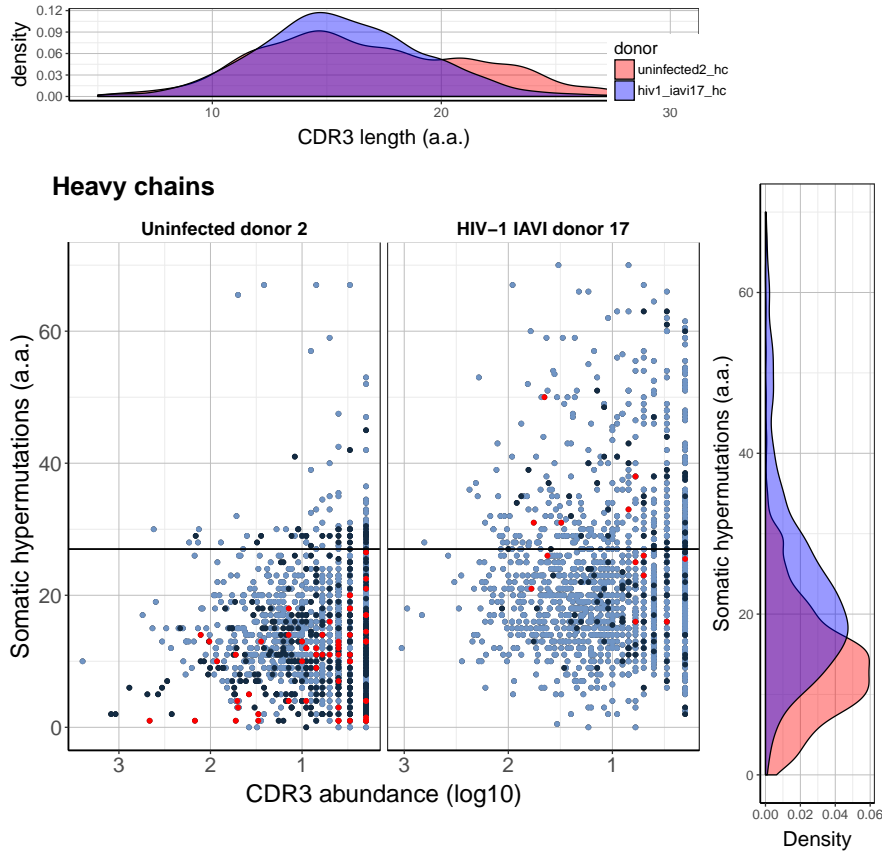


Figure 17: CDR3 frequency, lengths and somatic hypermutation density in clonotypes with 90% CDR3 a.a. identity.

```

axis.text.x = element_blank(),
axis.text.y = element_blank(),
axis.ticks = element_blank()
)

#scatterplot of x and y variables
scatter <- ggplot(clon90,aes(cdr3_length, mean_mutations)) +
geom_point(aes(color=L1)) +
scale_color_manual(values = c("red", "#56B4E9", "darkgreen",
                              "yellow", "#999999", "#E69F00")) +
theme(legend.position=c(1,1),legend.justification=c(1,1))
# scatter <- hc_abundance_mutations_plot

#marginal density of x - plot on top
plot_top <- ggplot(clon90,aes(cdr3_length, fill=L1)) +
geom_density(alpha=.5) +
scale_fill_manual(values = c("red", "#56B4E9", "darkgreen",
                              "yellow", "#999999", "#E69F00")) +
theme(legend.position = "none")

#marginal density of y - plot on the right

```

```

plot_right <- ggplot(clon90,aes(mean_mutations, fill=L1)) +
  geom_density(alpha=.5) +
  coord_flip() +
  scale_fill_manual(values = c("red", "#56B4E9", "darkgreen",
                                "yellow","#999999", "#E69F00")) +
  theme(legend.position = "none")

# Arrange the plots together, with appropriate
# height and width for each row and column
pdf("figure/cdr3_length_mean_mutations.pdf")
grid.arrange(plot_top, empty, scatter, plot_right,
              ncol=2, nrow=2, widths=c(4, 1), heights=c(1, 4))
dev.off()

# Make plot from clones based on 100 % CDR3 identity

load("working_uninfected2.RData")
load("working_hiv1_iavi17.RData")

working_list <- list(working_uninfected2, working_hiv1_iavi17)

abundance_mutations <- rbind(working_uninfected2, working_hiv1_iavi17)
abundance_mutations$donor <- c(rep("uninfected2_hc",
                                   times=nrow(working_uninfected2)),
                              rep("hiv1_iavi17_hc",
                                   times=nrow(working_hiv1_iavi17)))

abundance_mutations$donor <- factor(abundance_mutations$donor)
levels(abundance_mutations$donor) <- levels(factor(abundance_mutations$donor))

# Separated Heavy Chains and Light Chains

hc_abundance_mutations <- abundance_mutations

hc_abundance_mutations <- rbind(abundance_mutations[abundance_mutations$donor
                                                       == "uninfected2_hc",],
                                abundance_mutations[abundance_mutations$donor ==
                                                       "hiv1_iavi17_hc",])

hc_abundance_mutations$donor <- factor(hc_abundance_mutations$donor,
                                       levels=c("uninfected2_hc", "hiv1_iavi17_hc"))

hc_abundance_mutations$cdr3category <- factor()

for(i in 1:length(hc_abundance_mutations$cdr3_length)){

  if(hc_abundance_mutations$cdr3_length[i] <21) {
    hc_abundance_mutations$cdr3category[i] <- "<21"}

  if(hc_abundance_mutations$cdr3_length[i] >=21 &
     hc_abundance_mutations$cdr3_length[i]<=25) {
    hc_abundance_mutations$cdr3category[i] <- "21-25"
  }
}

```

```

}

if(hc_abundance_mutations$cdr3_length[i] >=26) {
  hc_abundance_mutations$cdr3category[i] <- "26-30"
}
}

samples <- list("Uninfected donor 2", "HIV-1 IAVI donor 17")

sample_labeller <- function(variable,value){
  return(samples[value])
}

nrow(hc_abundance_mutations[hc_abundance_mutations$donor == "uninfected2_hc",])
# [1] 1862

hc_uninf2 <- hc_abundance_mutations[hc_abundance_mutations$donor == "uninfected2_hc",]

nrow(hc_uninf2[hc_uninf2$vregion_mutations >28,])
# [1] 88

nrow(hc_abundance_mutations[hc_abundance_mutations$donor == "hiv1_iavi17_hc",])
# [1] 1557

percentage_clones_above_bNAbs_mutations_hc <- data.frame(uninfected2_hc=
nrow(hc_uninf2[hc_uninf2$vregion_mutations >28,])/
  nrow(hc_abundance_mutations[hc_abundance_mutations$donor
    == "uninfected2_hc",])*100,
hiv1_iavi17_hc=nrow(hc_hiv1[hc_hiv1$vregion_mutations>28,])/
  nrow(hc_abundance_mutations[hc_abundance_mutations$donor ==
    "hiv1_iavi17_hc",])*100)

print(percentage_clones_above_bNAbs_mutations_hc)
#   uninfected2_hc hiv1_iavi17_hc
#      4.726101      23.50674

hc_df_layer_1 <- hc_abundance_mutations[hc_abundance_mutations$cdr3category == "<21",]
hc_df_layer_2 <- hc_abundance_mutations[hc_abundance_mutations$cdr3category == "21-25",]
hc_df_layer_3 <- hc_abundance_mutations[hc_abundance_mutations$cdr3category == "26-30",]

reverselog_trans <- function(base = exp(1)) {
  trans <- function(x) -log(x, base)
  inv <- function(x) base^(-x)
  trans_new(paste0("reverselog-", format(base)), trans, inv,
    log_breaks(base = base),
    domain = c(1e-100, Inf))
}

hc_abundance_mutations_plot <- ggplot(hc_abundance_mutations,
  aes(x=cdr3_abundance, y=vregion_mutations))

```

```

hc_abundance_mutations_plot <- hc_abundance_mutations_plot + facet_grid(~donor,
                                scales = "free", labeller=sample_labeller) + geom_point() +
  geom_point(data=hc_df_layer_1, aes(x=cdr3_abundance, y=vregion_mutations), colour="#7095c5") +
  geom_point(data=hc_df_layer_2, aes(x=cdr3_abundance, y=vregion_mutations), colour="#132b43") +
  geom_point(data=hc_df_layer_3, aes(x=cdr3_abundance, y=vregion_mutations), colour="red") +
  theme(axis.text.x = element_text(angle=360,size=5)) +
  theme_bw(base_size=12, base_family = "Helvetica") + xlab("CDR3 abundance (log10)") +
  labs (title = "Heavy chains", color = "CDR3 length", y="Somatic hypermutations (a.a.)") +
  theme(strip.background = element_rect(fill="white", linetype="solid", color="white"),
        strip.text=element_text(face="bold", size=rel(1.2), hjust=0.5, vjust=1)) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(strip.background=element_rect(fill = "white")) +
  # 27 is the median of SHM in bNAbs
  geom_hline(aes(yintercept=27), colour = "black", size=0.7) +
  theme(plot.title=element_text(family="Helvetica",
                                face="bold", size=rel(1.5), hjust=-0.035, vjust=3.5)) +
  theme(axis.text=element_text(family="Helvetica", size=rel(1.2)),
        axis.title=element_text(family="Helvetica", size=rel(1.2)),
        legend.text=element_text(family="Helvetica", size=rel(1.1)),
        legend.title=element_text(family="Helvetica", face="bold", size=rel(1.2))) +
  theme(axis.title.x = element_text(vjust=-0.5, size=rel(1.2)),
        axis.title.y = element_text(vjust=1, size=rel(1.2))) +
  scale_x_continuous(breaks = c(1,10, 100,1000, 10000), trans = reverselog_trans(10),
                     labels = trans_format("log10", math_format(.x))) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
        axis.line=element_line(size = 0.7, color = "black"),
        text = element_text(size = 14)) +
  theme(plot.margin = unit(c(1, 0.2, 1.5, 0.2),"cm")) +
  theme(legend.position=c(0.1,-0.25), legend.justification=c(0,0),
        legend.direction = "horizontal") +
  guides(colour = guide_legend(override.aes = list(size=5))) +
  theme(strip.text.x = element_text(size = 15, colour = "black", angle = 0)) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
        axis.line = element_line(size = 0.7, color = "black"),
        text = element_text(size = 14))

### Uncomment the following to produce the scatter plot only
# pdf("figure/scatter_cdr3_length_mutations_frequency.pdf")
# hc_abundance_mutations_plot
# dev.off()

#scatterplot of x and y variables
scatter <- hc_abundance_mutations_plot

# marginal density of x - plot on top
plot_top <- ggplot(hc_abundance_mutations, aes(x=cdr3_length, fill=donor)) +
  geom_density(alpha=.4) +
  scale_fill_manual(values = c("red", "blue", "#999999")) +
  theme(legend.position = "none") + theme_bw(12) +
  theme(axis.text.x = element_text(angle=360, size=5)) +
  theme_bw(base_size=12, base_family = "Helvetica") +
  labs (title = "", color = "", x="CDR3 length (a.a.)") +
  theme(strip.background = element_rect(fill="white", linetype="solid", color="white"),

```

```

    strip.text=element_text(face="bold", size=rel(1.2), hjust=0.5, vjust=1)) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(plot.title=element_text(family="Helvetica", face="bold", size=rel(1.5),
                                hjust=-0.035, vjust=2.5)) +
  theme(axis.title.x = element_text(vjust=-0.5, size=rel(1.2)),
        axis.title.y = element_text(vjust=1, size=rel(1.2))) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
        axis.line=element_line(size = 0.7, color = "black"),
        text = element_text(size = 14)) +
  theme(legend.position=c(0,0), legend.justification=c(-4.2,0.1),
        legend.direction = "vertical") +
  guides(colour = guide_legend(override.aes = list(size=5))) +
  theme(strip.text.x = element_text(size = 15, colour = "black", angle = 0)) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
        axis.line = element_line(size = 0.7, color = "black"),
        text = element_text(size = 14))

#marginal density of y - plot on the right
plot_right <- ggplot(hc_abundance_mutations, aes(x=vregion_mutations, fill=donor)) +
  geom_density(alpha=.4) +
  coord_flip() +
  scale_fill_manual(values = c("red", "blue", "#999999")) + theme_bw(12) +
  theme(axis.text.x = element_text(angle=360, size=5)) +
  theme_bw(base_size=12, base_family = "Helvetica") +
  labs (title = "", color = "", x="Somatic hypermutations (a.a.)", y="Density") +
  theme(strip.background = element_rect(fill="white", linetype="solid", color="white"),
        strip.text=element_text(face="bold", size=rel(1.2), hjust=0.5, vjust=1)) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(strip.background=element_rect(fill = "white")) +
  theme(plot.title=element_text(family="Helvetica", face="bold",
                                size=rel(1.5), hjust=-0.035, vjust=3.5)) +
  theme(axis.title.x = element_text(vjust=-0.5, size=rel(1.2)),
        axis.title.y = element_text(vjust=1, size=rel(1.2))) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
        axis.line=element_line(size = 0.7, color = "black"),
        text = element_text(size = 14)) +
  theme(panel.grid.major = element_line(size = 0.2, color = "grey"),
        axis.line = element_line(size = 0.7, color = "black"),
        text = element_text(size = 14)) + theme(legend.position = "none")

#arrange the plots together, with appropriate height and width for each row and column
pdf("figure/cdr3_length_mutations_frequency_raw.pdf", width=10, height=10)
grid.arrange(plot_top, empty, scatter, plot_right, ncol=2, nrow=2,
              widths=c(4, 1), heights=c(1, 4))
dev.off()

```

## 10 The architecture of natural antibody repertoires

The architecture of antibody repertoires was previously revealed through network analysis [21]. Briefly, networks (graphs) were constructed from the sparse triangle matrix of pairwise Levenshtein distances (LD) between CDR3s. For small samples (up to 30 000 unique CDR3 sequences) such a calculation can be

performed on a single computer. However, due to the  $N^2$  complexity of required calculations, computing the pairwise matrix for samples of > 100 000 unique CDR3 sequences becomes prohibitively expensive. For generating and outputting the larger graphs imNet tool in python can be used [21].

For smaller networks the following code and igraph [22] package can be used. In Figure 18 the CDR3 network from HIV-1 IAVI donor 17 that has developed broadly neutralizing antibodies is shown.

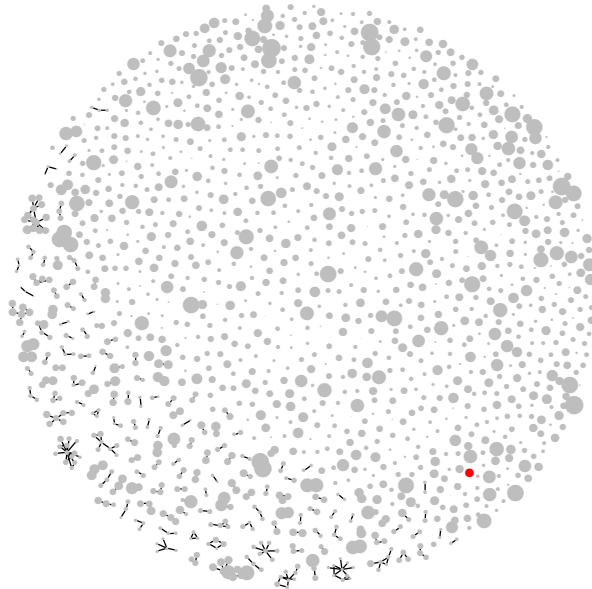


Figure 18: Network of CDR3 clones. Clone with best VDJ alignment to bNAbs database is shown in red.

```
load("working_hiv1_iavi17.RData")

library(igraph)

cdr3 <- as.character(working_hiv1_iavi17$cdr3)
cdr3_dist <- stringDist(cdr3, method = "levenshtein")
cdr3_mat <- as.matrix(cdr3_dist)
cdr3_bol <- cdr3_mat
cdr3_bol[cdr3_bol<=1] <- 1
cdr3_bol[cdr3_bol>1] <- 0
colnames(cdr3_bol) <- cdr3
rownames(cdr3_bol) <- cdr3

# CDR3 corresponding to the top aligned VDJ with the bNAb database
as.character(working_hiv1_iavi17[working_hiv1_iavi17$vdj_region ==
"AAMAGTSGPGLVKPSETLSVTCVSGDSMNYYWTWIRQSPGKGLEWIGYISDRPSATYNPSLNSR
VVISRDTSKNQLSLKLSVTPADTAVYYCATARRGQRIYGEVSFGEFFYYYSMDVWGKGTAVTVSS",]$cdr3)
```



```

# "ATARRGQRIYGEVSFGEFFYYYSMDV"

# The entire row
working_hiv1_iavi17[working_hiv1_iavi17$vdj_region ==
"AAMAGTSGPGLVKPSETLSVTCVSGDSMNYYWTWIRQSPGKGLEWIGYISDRPSATYNPSLNSRVV
ISRDTSKNQLSLKLNSVTPADTAVYYCATARRGQRIYGEVSFGEFFYYYSMDVWGKGTAVTVSS",]

cdr3_graph <- igraph::simplify(graph.adjacency(cdr3_bol, weighted=T, mode = "undirected"))

# Name of the first node of the graph
V(cdr3_graph)$name[1]
# [1] "VRSWSGHLDPDL"

nodesize <- ifelse(V(cdr3_graph)$name ==
as.character(working_hiv1_iavi17[working_hiv1_iavi17$vdj_region ==
"AAMAGTSGPGLVKPSETLSVTCVSGDSMNYYWTWIRQSPGKGLEWIGYISDRPSATYNPSLNSR
VVISRDTSKNQLSLKLNSVTPADTAVYYCATARRGQRIYGEVSFGEFFYYYSMDVWGKGTAVTVSS",]$cdr3),
3, working_hiv1_iavi17$vregion_mutations/11)

# Mark in red the top hit (best aligned) with bNAb database sequences
nodecolor <- ifelse(V(cdr3_graph)$name ==
as.character(working_hiv1_iavi17[working_hiv1_iavi17$vdj_region ==
"AAMAGTSGPGLVKPSETLSVTCVSGDSMNYYWTWIRQSPGKGLEWIGYISDRPSATYNPSLNSR
VVISRDTSKNQLSLKLNSVTPADTAVYYCATARRGQRIYGEVSFGEFFYYYSMDVWGKGTAVTVSS",]$cdr3),
"red", "grey")

pdf("figure/cdr3_network.pdf")
set.seed(11)
plot(cdr3_graph, vertex.frame.color=NA,
      layout=layout_with_fr(cdr3_graph, grid="nograd", niter=200),
      vertex.label=NA, vertex.color=nodecolor, edge.width=1,
      vertex.size=nodesize, edge.color = "black")
dev.off()

### With the following you can extract R code from Rnw file
# f <- 'bioinformatic_analysis_of_immune_repertoires_em.Rnw'
# knitr::purl(f)

```

## 11 Acknowledgements

To Professor Sai T. Reddy goes my gratitude for his patient guidance. I am thankful for the opportunity to prepare and present this material as part of the bioinformatics immunue repertoire analysis for the training seminar "Next-generation sequencing for antibody repertoire discovery and engineering" at PEGS, Boston, 2017.

I am grateful to Dr. Victor Greiff who contributed in the past 3 years R code and advice on the analysis of immune repertoires and tracking available public data through literature.

## References

- [1] Dmitriy A Bolotin, Stanislav Poslavsky, Igor Mitrophanov, Mikhail Shugay, Ilgar Z Mamedov, Ekaterina V Putintseva, and Dmitriy M Chudakov. Mixcr: software for comprehensive adaptive immunity profiling. *Nat. Methods*, 12(5):380–381, 2015.
- [2] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer, 2009.
- [3] Hadley Wickham. *ggplot2*. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011.
- [4] Yihui Xie. *knitr: A general-purpose package for dynamic report generation in R*. 2012.
- [5] Linling He, Devin Sok, Parisa Azadnia, Jessica Hsueh, Elise Landais, Melissa Simek, Wayne C Koff, Pascal Poignard, Dennis R Burton, and Jiang Zhu. Toward a more accurate view of human b-cell repertoire by next-generation sequencing, unbiased repertoire capture and single-molecule barcoding. *Scientific reports*, 4:6778, 2014.
- [6] Marie-Paule Lefranc, Véronique Giudicelli, Patrice Duroux, Joumana Jabado-Michaloud, Géraldine Folch, Safa Aouinti, Emilie Carillon, Hugo Duvergey, Amélie Houles, Typhaine Paysan-Lafosse, et al. Imgt®, the international immunogenetics information system® 25 years on. *Nucleic acids research*, page gku1056, 2015.
- [7] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.
- [8] Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic acids research*, 41(D1):D36–D42, 2013.
- [9] A. M. Eroshkin, A. LeBlanc, D. Weekes, K. Post, Z. Li, A. Rajput, S. T. Butera, D. R. Burton, and A. Godzik. bNAber: database of broadly neutralizing HIV antibodies. *Nucleic Acids Research*, 42(D1):D1133–D1139, January 2014.
- [10] Mark B Swindells, Craig T Porter, Matthew Couch, Jacob Hurst, KR Abhinandan, Jens H Nielsen, Gary Macindoe, James Hetherington, and Andrew CR Martin. abysis: Integrated antibody sequence and structure management, analysis, and prediction. *Journal of Molecular Biology*, 429(3):356–364, 2017.
- [11] Brandon J. DeKosky, Gregory C. Ippolito, Ryan P. Deschner, Jason J. Lavinder, Yariv Wine, Brandon M. Rawlings, Navin Varadarajan, Claudia Giesecke, Thomas DÄürner, Sarah F. Andrews, Patrick C. Wilson, Scott P. Hunicke-Smith, C. Grant Willson, Andrew D. Ellington, and George Georgiou. High-throughput sequencing of the paired human immunoglobulin heavy and light chain repertoire. *Nature Biotechnology*, 2013.
- [12] Andre P. Masella, Andrea K. Bartram, Jakub M. Truszkowski, Daniel G. Brown, and Josh D. Neufeld. PANDAsseq: paired-end assembler for illumina sequences. *BMC Bioinformatics*, 13(1):31, February 2012.
- [13] Eltaf Alamyar, Véronique Giudicelli, Shuo Li, Patrice Duroux, and Marie-Paule Lefranc. Imgt/highv-quest: the imgt® web portal for immunoglobulin (ig) or antibody and t cell receptor (tr) analysis from ngs high throughput and deep sequencing. *Immunome research*, 8(1):26, 2012.
- [14] Xavier Brochet, Marie-Paule Lefranc, and Véronique Giudicelli. Imgt/v-quest: the highly customized and integrated system for ig and tr standardized vj and vdj sequence analysis. *Nucleic acids research*, 36(suppl 2):W503–W508, 2008.
- [15] Marie-Paule Lefranc. Immunoinformatics of the v, c, and g domains: Imgt® definitive system for ig, tr and igsf, mh, and mhsf. *Immunoinformatics*, pages 59–107, 2014.
- [16] Victor Greiff, Enkelejda Miho, Ulrike Menzel, and Sai T. Reddy. Bioinformatic and statistical analysis of adaptive immune repertoires. *Trends in Immunology*, 36(11), 2015.

- [17] Safa Aouinti, Véronique Giudicelli, Patrice Duroux, Dhafer Malouche, Sofia Kossida, and Marie-Paule Lefranc. Imgt/statclonotype for pairwise evaluation and visualization of ngs ig and tr imgt clonotype (aa) diversity or expression from imgt/highv-quest. *Frontiers in Immunology*, 7, 2016.
- [18] Victor Greiff, Pooja Bhat, Skylar C Cook, Ulrike Menzel, Wenjing Kang, and Sai T Reddy. A bioinformatic framework for immune repertoire diversity profiling enables detection of immunological status. *Genome medicine*, 7(1):49, 2015.
- [19] Hongen Zhang, Paul Meltzer, and Sean Davis. Rcircos: an r package for circos 2d track plots. *BMC bioinformatics*, 14(1):244, 2013.
- [20] Enkelejda Miho. Signatures of broadly neutralizing status in antibody repertoire sequences. In preparation, 2017.
- [21] Enkelejda Miho, Victor Greiff, Sai T Reddy, et al. The fundamental principles of antibody repertoire architecture revealed by large-scale network analysis. *bioRxiv*, page 124578, 2017.
- [22] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.