abseqR: reporting and data analysis functionalities for Rep-Seq datasets of antibody libraries

Jia Hong Fong and Monther Alhamdoosh

15 April 2025

Package

abseqR 1.26.0

Contents

1 Introduction

A plethora of high throughput sequencing (HTS) analysis pipelines are available as open source tools to analyze and validate the quality of Rep-seq [1] datasets. OmicTools provides a summary of repertoire sequencing tools that implements different techniques and algorithms in analyzing and visualizing datasets from B-cell receptors (BCR) and T-cell receptors (TCR). However, high throughput analysis pipelines of antibody library sequencing datasets are scarce.

`AbSeq` is a comprehensive bioinformatic pipeline for the analysis of sequencing datasets generated from antibody libraries and *abseqR* is one of its packages. The `AbSeq` suite is implemented as a set of functions in `R` and `Python` that can be used together to provide insights into the quality of antibody libraries. abseqPy processes paired-end or single-end *FASTA/FASTQ* files generated from NGS sequencers and converts them into CSV and HDF files. *abseqR* visualizes the output of abseqPy and generates a self-contained HTML report. Furthermore, *abseqR* provides additional functionalities to explicitly compare multiple samples and perform further downstream analyses.

abseqR provides the following functionalities:

- Visualizations: the output from abseqPy is summarized succintly into static and interactive plots. The plots are also stored in `Rdata` object files that provide flexibility for users to easily customize the aesthetics of any plot generated by *abseqR*.
- Interactive reports: the plots generated by *abseqR* can be collated and presented in a self-contained HTML document for convenience and ease of sharing.
- Sample comparison: *abseqR* overloads the `+` operator via the S4 classes `AbSeqCRep` and `AbSeqRep` to compare multiple samples with each other. The `comparative reports` include additional plots, for example, sample similarity clustering, overlapping clonotypes, etc. The usual plots are also generated for all the compared samples by adding an extra layer of `aes`thetic.

1.1 AbSeq core analyses

`AbSeq` includes, but is not limited to, merging paired-end reads, annotating V-(D)-J germlines, calculating unique clonotypes, analyzing primer specificity, facilitating the selection of best restriction enzymes, predicting frameshifts, identifying functional clones, and calculating diversity indices and estimations. These analyses are seamlessly extrapolated to analyze multiple library repertoires simultaneously when multiple samples are present. Figure 1 depicts the complete `AbSeq` workflow. Sequencing files are taken as input to be annotated and analyzed by abseqPy before they are further analyzed and visualized by *abseqR*.

[1] Repertoire sequencing.

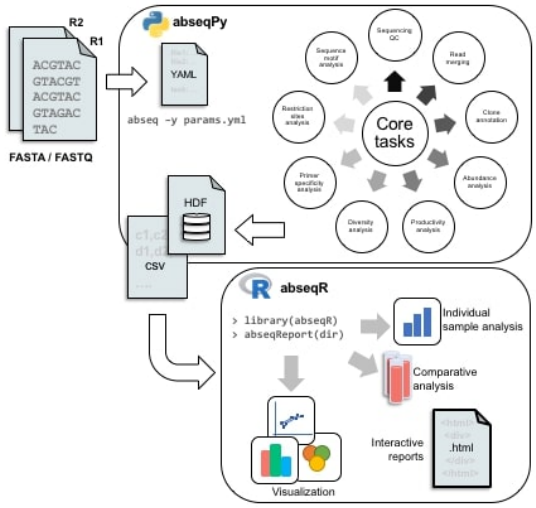

Figure 1: AbSeq workflow. Comprehensive analyses and visualizations are generated from input sequencing files using abseqPy and abseqR.

2 Installation

abseqR can be installed from bioconductor.org or its GitHub repository at https://github.com/malhamdoosh/abseqR.

2.1 Bioconductor

To install *abseqR* via the `BiocManager`, type in R console:

```
if (!require("BiocManager"))
    install.packages("BiocManager")
BiocManager::install("abseqR")
```

2.2 GitHub

To install the development version of *abseqR* from GitHub, type in R console:

```
if (!require("BiocManager"))
    install.packages("BiocManager")
BiocManager::install("malhamdoosh/abseqR")
```

2.3 System prerequisites

abseqR requires pandoc version *1.19.2.1* or higher to render the HTML reports. If `pandoc` cannot be detected while executing *abseqR*, the HTML report will **not** be generated. *abseqR* is a cross-platform library and will work on any major operating system [2].

[2]The performace offered by BiocParallel may differ across different operating systems.

2.4 R package dependencies

abseqR depends on several packages from the CRAN and Bioconductor repositories:

- *RColorBrewer* provides colour schemes for maps and graphics. To install it, type in R console `install.packages("RColorBrewer")`
- *VennDiagram* provides a set of functions to generate Venn diagrams. To install it, type in R console `install.packages("VennDiagram")`
- *circlize* is a visualization tool used to summarize the distributions of associations between V-J gene segments. To install it, type in R console `install.packages("circlize")`
- *flexdashboard* is a package that provides a template for RMarkdown that resembles a grid oriented dashboard and is used to generate the HTML reports. To install it, type in R console `install.packages("flexdashboard")`
- *ggplot2* is an implementation of the "Grammar of Graphics" in R. It is used extensively to generate plots. To install it, type in R console `install.packages("ggplot2")`
- *ggcorrplot* is used to visualize a correlation matrix using *ggplot2*. To install it, type in R console `install.packages("ggcorrplot")`
- *ggdendro* provides a set of tools for drawing dendrograms and tree plots using *ggplot2*. To install it, type `install.packages("ggdendro")`
- *grid* is used to arrange plots. It has been integrated into the base R package.
- *gridExtra* provides functions to work with "grid" graphics and used to arrange grid-based plots in the HTML reports. To install it, type in R console `install.packages("gridExtra")`
- *knitr* provides the capability to dynamically generate reports in R. To install it, type in R console `install.packages("knitr")`
- *plotly* is used to translate *ggplot2* graphs to interactive web-based plots. To install it, type in R console `install.packages("plotly")`
- *plyr* offers a set of tools used in this package to apply operations on subsets of data in manageable pieces. To install it, type in R console `install.packages("plyr")`
- *png* is used to read and display PNG images. To install it, type in R console `install.packages("png")`
- *reshape2* allows this package to restructure and aggregate dataframes. To install it, type in R console `install.packages("reshape2")`
- *rmarkdown* converts R Markdown documents into a variety of formats. To install it, type in R console `install.packages("rmarkdown")`
- *vegan* provides a suite of functions to calculate diversity and distance statistics between repertoires. To install it, type in R console `install.packages("vegan")`
- *BiocParallel* is a package from Bioconductor used to enable parallel computing. To install it, type in R console

```
if (!require("BiocManager"))
    install.packages("BiocManager")
BiocManager::install("BiocParallel")
```

3 Quick start

To leverage all the functionalities provided by *abseqR*, the main functions to note are `abseqR::abseqReport`, `abseqR::report`, and `+`. This section uses a small simulated dataset to walk through the use cases of each function.

3.1 Datasets

The example dataset is packaged with *abseqR*. For the sake of brevity, the dataset generation is described under the Appendices section.

Briefly, the dataset includes three samples, namely PCR1, PCR2, and PCR3, that was generated using *in silico* simulations. abseqPy was then used to analyze the datasets and the output directory argument `--outdir` specified in abseqPy was initiated with the value `"./ex/"`.

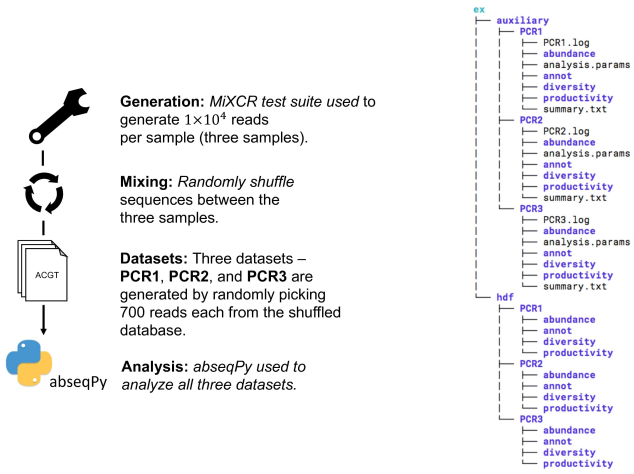

Figure 2: Generation of the three simulated datasets. The flowchart on the left shows the generation of the datasets in '"./ex/"' while the folder structure on the right shows the output generated by 'abseqPy' on the three datasets.

3.1.1 Fetch data files

The output of abseqPy on the simulated datasets is first fetched into a local directory as follows:

```
# substitute with any directory that you have read/write access to
sandboxDirectory <- tempdir()

# path to provided data (comes installed with abseqR)
exdata <- system.file("extdata", "ex", package = "abseqR")

# copy the provided data to sandboxDirectory
file.copy(exdata, sandboxDirectory, recursive = TRUE)
```

Then, the following commands can be executed in R console to verify that the three `PCR` datasets are fetched successfully:

```
# dataPath now holds the path to a local copy of the data
dataPath <- file.path(sandboxDirectory, "ex")
# the sample names can be found inside the auxiliary directory
list.files(path = file.path(dataPath, "auxiliary"))
## [1] "PCR1" "PCR2" "PCR3"
```

3.2 Basic analysis

After obtaining the datasets, the `abseqReport` function from *abseqR* is invoked to visualize the different analysis results as follows:

```
# This section will visualize all the datasets individually
# and compare PCR1 with PCR2 with PCR3

# Interim solution
if (Sys.info()["sysname"] == "Darwin") {
    BPPARAM <- BiocParallel::SerialParam()
} else {
    BPPARAM <- BiocParallel::bpparam()
}

# you should use report = 3 to generate a HTML report
samples <- abseqReport(dataPath,
                       compare = c("PCR1, PCR2, PCR3"),
                       report = 1,
                       BPPARAM = BPPARAM)

# ignore the message:
# "Sample output directory <path> is different from provided path
#   <path> assuming directory was moved"
# This warning message tells us that the directory has
# been moved (we copied the provided examples to "dataPath")
```

This creates plots for all samples included in `dataPath`. In addition, The `compare = c("PCR1, PCR2, PCR3")` argument specifies that samples `PCR1`, `PCR2`, and `PCR3` are explicitly compared against each other. Other possible values for `compare`, `report` , and `BPPARAM` will be discussed in detail in later sections (here, here, and here).

3.3 HTML reports' directory structure

Figure 3 shows the folder structure of `./ex/` after *abseqR* completes.

Invoking `abseqReport` generates plots in the same folder as the corresponding data files within the `auxiliary` directory. They are then collated together in an HTML document found in the `report` directory.

The `report` directory is structured as follows:

- `index.html` file is the entry point to browse AbSeq's HTML reports. It summarizes the `AbSeq` analysis and provides a convenient way for navigating individual and comparative analysis results. For example, within this file, there are links to the reports generated for `PCR1`, `PCR2`, `PCR3` and `PCR1 vs PCR2 vs PCR3`.

```
ex/
├── auxiliary
│   ├── PCR1
│   │   ├── abundance
│   │   ├── analysis.params
│   │   ├── annot
│   │   ├── diversity
│   │   ├── PCR1.log
│   │   ├── productivity
│   │   └── summary.txt
│   ├── PCR1_vs_PCR2_vs_PCR3
│   │   ├── abundance
│   │   ├── annot
│   │   ├── clonotype_analysis
│   │   ├── diversity
│   │   └── productivity
│   ├── PCR2
│   │   ├── abundance
│   │   ├── analysis.params
│   │   ├── annot
│   │   ├── diversity
│   │   ├── PCR2.log
│   │   ├── productivity
│   │   └── summary.txt
│   └── PCR3
│       ├── abundance
│       ├── analysis.params
│       ├── annot
│       ├── diversity
│       ├── PCR3.log
│       ├── productivity
│       └── summary.txt
├── hdf
│   └── README
└── report
    ├── html_files
    │   ├── PCR1.html
    │   ├── PCR1_vs_PCR2_vs_PCR3.html
    │   ├── PCR2.html
    │   └── PCR3.html
    └── index.html
```

Figure 3: Folder structure after 'abseqR' completes. The landing page 'index.html' provides a convenient way to navigate around the standalone HTML reports in 'html_files/'.

- `html_files` directory contains HTML files that are used build the individual and comparative reports. They can be accessed directly or via the main page `index.html`.

In conclusion, the individual sample reports in `html_files` can be shared as-is, but `index.html` **must** be accompanied by the `html_files` directory and thus it is recommended to share the entire `report` folder.

3.4 Comparative analysis

3.4.1 Comparative analysis using a single dataset

This section describes the possible values for `abseqReport`'s `compare` parameter. In the previous section, `abseqReport` was called with `compare = c("PCR1, PCR2, PCR3")`. This compares the three samples all together. However, it is also possible to compare only a subset of samples in the `dataPath` folder, multiple subsets of samples, or none at all.

The `compare` parameter accepts a vector of one or more strings. Each string denotes a comparison between samples separated by commas, for example, `compare = c("PCR1, PCR2, PCR3")`[3].

If sample comparison is not required, then the following can be simply invoked `samples <- abseqReport(dataPath)`.

Example of other combinations:

```
# Example of 1 comparison
# Analyze all samples, but only compare PCR1 with PCR2
pcr1.pcr2 <- abseqReport(dataPath,
                         compare = c("PCR1, PCR2"),
                         report = 0)

# Example of 2 comparisons
# Analyze all samples. In addition, compare:
#  * PCR1 with PCR2
#  * PCR2 with PCR3
multiComparison <- abseqReport(dataPath,
                               compare = c("PCR1, PCR2", "PCR2, PCR3"),
                               report = 0)
```

Note, `abseqReport` always returns S4 objects of the class **AbSeqRep** for each sample in the `dataPath` directory regardless of the value of the `compare` argument as illustrated next:

```
# compare = c("PCR1,PCR2")
names(pcr1.pcr2)

# compare = c("PCR1, PCR2", "PCR2 ,PCR3")
names(multiComparison)
## [1] "PCR1" "PCR2" "PCR3"
## [1] "PCR1" "PCR2" "PCR3"
```

The next subsection explains the motivation behind this behaviour.

3.4.2 Comparative analysis using multiple datasets

When constructing antibody libraries, users might be interested in comparing Ab repertoires from different stages of the construction process. Usually, each stage has its own sequencing runs and thus would be analyzed indepedent of others. The `report` function in *abseqR* was written to enable this behaviour as illustrated next.

Previously, the S4 objects of three samples of our toy example loaded into a variable named `samples`. As a hypothetical example, if the reports show an interesting observation between `PCR1` and `PCR3`, it might be of interest to isolate the two samples from the rest.

This code shows how the `+` operator can be used to create customized comparisons as follows:

```
# recall that samples is a named list where each element's name
# is the sample's own name (see names(samples))
```

[3]Trailing and leading whitespace between sample names are trimmed. That is, "PCR1,PCR2" is identical to "PCR1 , PCR2".

```
# use report = 3 if the results should be collated in a HTML document
pcr1.pcr3 <- samples[["PCR1"]] + samples[["PCR3"]]
refinedComparison <- report(pcr1.pcr3,
                            outputDir = file.path(sandboxDirectory,
                                                  "refined_comparison"),
                            report = 1)
```

Here, the `+` operator creates a new comparison between `PCR1` and `PCR3` of class *AbSeqCRep*. S4 objects of this class can be passed to the `report` function to generate a standalone HTML report of this particular comparison. Similar to `abseqReport`, this function returns S4 objects of the individual samples - `PCR1` and `PCR3`.

```
names(refinedComparison)
## [1] "PCR1" "PCR3"
```

Similarly, samples can be loaded from two different abseqPy's directories as illustrated in the following example:

```
# first abseqPy run on SRR dataset from control group
abseq --file1 fasta/SRR_ACGT_CTRL.fasta --outdir SRR_CTRL

# second abseqPy run on SRR dataset from experiment group
abseq --file1 fasta/SRR_ACGT_EXP.fasta --outdir SRR_EXP
```

analyzing these samples in *abseqR*:

```
SRRControl <- abseqReport("SRR_CTRL")
SRRExp <- abseqReport("SRR_EXP")
```

then comparing *all* samples in `SRRControl` with *all* samples in `SRRExp` can be done using `+` and `report`.

```
# short for SRRControl[[1]] + SRRControl[[2]] + ... + SRRExp[[1]] + ...
all.samples <- Reduce("+", c(SRRControl, SRRExp))
report(all.samples, outputDir = "SRRControl_vs_SRRExp")
```

Important: The sample names in `SRR_CTRL` and `SRR_EXP` *must* be unique.

In conclusion, the `+` operator along with the `report` function enables users to carry out a wide range of customized downstream analyses on the output of abseqPy.

4 Advanced Examples

4.1 Lazy loading

In the previous section, the `SRRControl` dataset was loaded using `SRRControl <- abseqReport("SRR_CTRL")`, which will **generate** all plots and reports by default. However, this dataset might have already been analyzed and users are interested in only loading the S4 objects of the samples. This can be efficiently carried out by using the `report=0` argument as follows:

```
# in essence, this is identical to SRRControl <- abseqReport("SRR_CTRL"),
# but will not regenerate plots and reports!
```

```
SRRControl.lazy <- abseqReport("SRR_CTRL", report = 0)

# exactly the same return value
stopifnot(names(SRRControl.lazy) == names(SRRControl))
```

4.2 Alternative reporting options

In the previous section, the `report` parameter of `abseqReport` was used to load the samples in `SRRControl` without actually plotting any data.

The `report` argument can accept one of four possible values as follows:

1. `abseqReport("SRR_CTRL", report = 0)` does not generate plots and HTLM reports and only returns a named list of S4 objects.
2. `abseqReport("SRR_CTRL", report = 1)` generates static plots in PNG format but **does not** generate HTML reports.
3. `abseqReport("SRR_CTRL", report = 2)` generates static plots in PNG format in addition to HTML reports.
4. `abseqReport("SRR_CTRL", report = 3)` generates static plots in PNG format and interactive plots in the HTML reports using plotly. This is the default behaviour when the `report` argument is not specified.

4.3 Parallelization

One of `abseqReport`'s parameters is `BPPARAM`, which is used to pass arguments into the `BiocParallel::bplapply` function for customizing parallelization. More information regarding the accepted values to `BPPARAM` can be found at BiocParallel's page.

Below is a simplified example of using 4 cores and serializing the loop.

```
# use 4 CPU cores
nproc <- 4
samples <- abseqReport(dataPath,
                       BPPARAM = BiocParallel::MulticoreParam(nproc))


# run sequentially - no multiprocessing
samples <- abseqReport(dataPath,
                       BPPARAM = BiocParallel::SerialParam())
```

5 Interpretation of report's figures

This section presents the plots generated by *abseqR* on the dataset discussed above and provides some insights on how to interpret them.

The visualizations and analyses can be broken down into 5 categories:

1. Sequence quality analysis
2. Abundance analysis

3. Productivity analysis
4. Diversity analysis
5. Comparative analysis

5.1 Sequence quality analysis

The plots described in this section can be found in the `Summary` and `Quality` tabs of the HTML report.

5.1.1 Sequence length

The sequence length distribution is a simple way of validating the quality of a sequencing run. The histogram is expected to show a large proportion of sequences falling within the expected lengths.

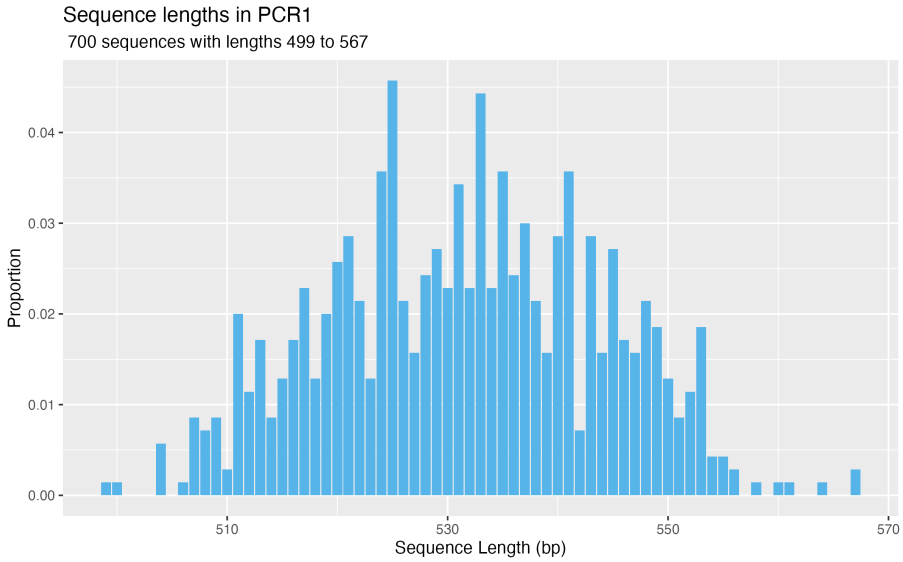

Figure 4: Sequence length distribution of PCR1. Histogram of (merged) sequence lengths.

Figure 4 plots sequence length (x-axis) against proportion (y-axis). A similar plot with outliers removed can be found in the `Summary` tab.

5.1.2 Alignment quality

abseqPy filters low quality sequences based on the quality of the sequence alignment against germline sequence databases and thus the following parameters can be used:

1. Alignment bitscore
2. Subject start index
3. Query start index
4. Alignment length

However, setting the optimal cut-off thresholds for these parameters is challenging. Stringent values could filter too many sequences while lenient values could retain low quality sequences.

The alignment quality heatmaps generated in the `Quality` tab of the HTML report shows the relationship between alignment lengths and these alignment parameters to help determine the percentage of sequences falling within a given range and thus inform the selection of cut-off thresholds.

For example, Figure 5 shows one of the 5 heatmaps: bitscore against V germline alignment length. abseqPy has some recommendations on the values of these parameters to retain good quality sequences.

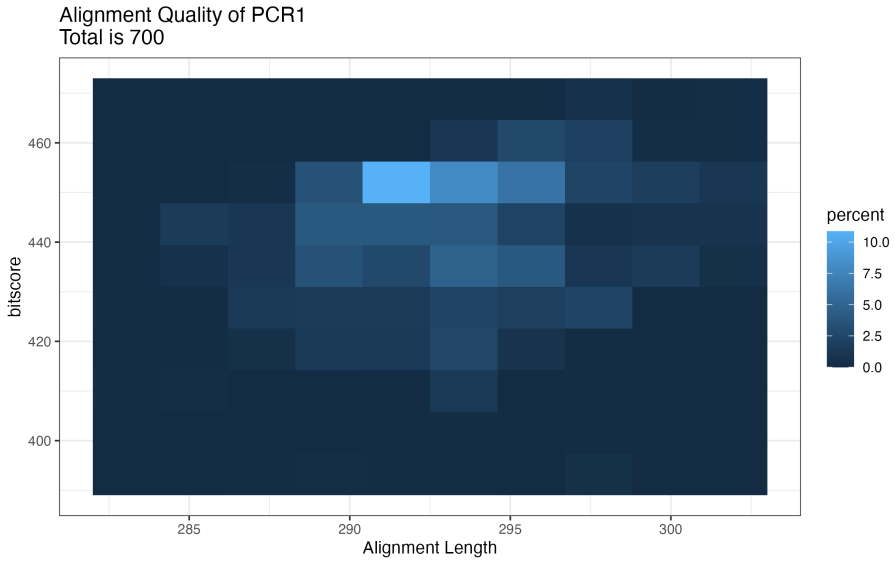

Figure 5: Bitscore vs V germline alignment length in PCR1. Heatmap of bitscore vs alignment length shows percentage of sequences in a given value range.

5.1.3 Insertions, deletions, and mismatches

Indels (Insertions or deletions) and mismatches can be used as an indicator of sequence quality.

Figure 6 shows the proportions of indels in `PCR1`. This graph plots the rate of indels (y-axis) in the V germlines of `PCR1` against the number of indels (x-axis). A similar plot for rate of mismatches in V germlines can be found in the same directory. A high rate of indels in the germline sequences might indicate a quality problem with the library because this would affect the functionality of the sequenced clones. However, this could be due to the sequencing quality depending on which sequencing technology is used. For example, long read sequencing technologies tend to produce more indel errors than short read sequencing technologies.

5.2 Abundance and association analysis

The plots generated in this section can be found in the `Abundance` tab of the HTML report.

5.2.1 V-(D)-J germline abundance

The proportions of V-(D)-J germlines is essential in some experiment designs. For example, it can be used to validate that the germline abundance of an in-house antibody library is in-line with the donor antibody repertoire. Experiments that artificially amplify certain germline families can also be validated similarly using this analysis.

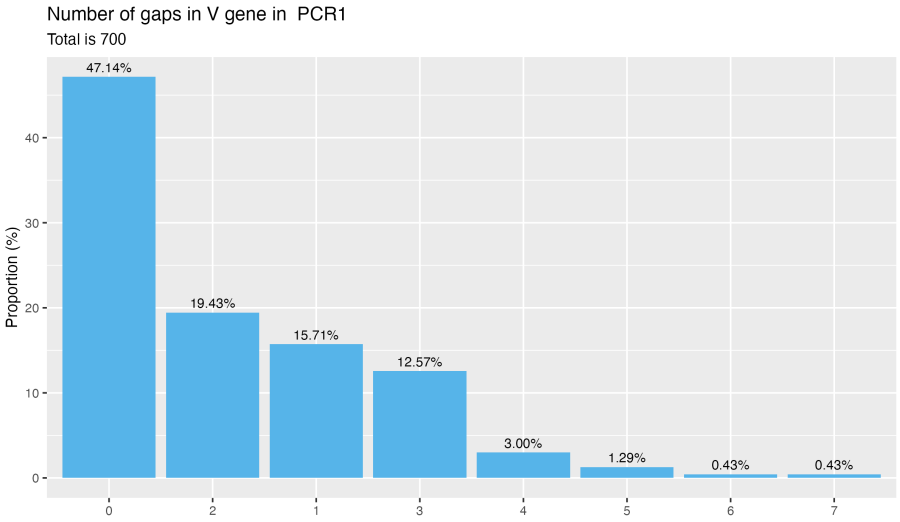

Figure 6: Rate of insertions and deletions in PCR1. The percentage of indels found within the V germlines of PCR1.

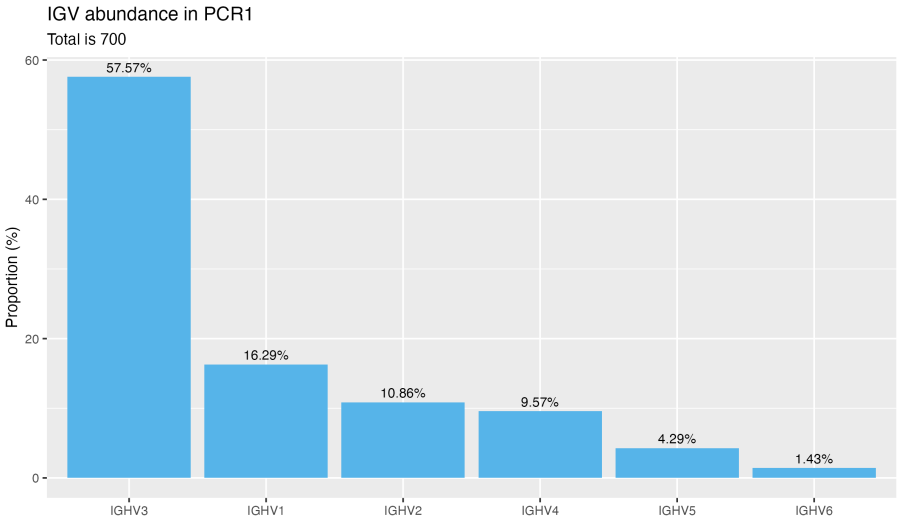

Figure 7: V family germline distribution in PCR1. The percentage of IGHV families after germline annotation using IgBLAST.

Figure 7 shows the distribution of IGHV families in `PCR1`. Similar plots can be generated for individual V germlines genes and for the D and J germlines.

5.2.2 V-J germline associations

This plot visualizes the recombination process of V and J germlines. Figure 8 summarizes the associations between V and J family germlines in a plot generated using *circlize*.

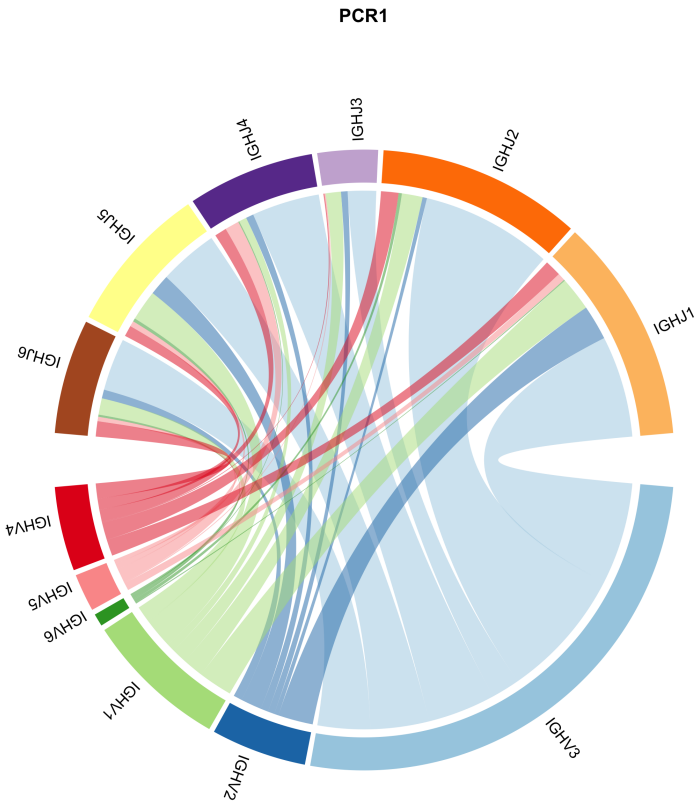

Figure 8: V-J family germline association in PCR1. Segment size represents germline proportion while the thickness of the arcs shows the proportion of associations between V and J germline families. This plot was heavily inspired by [VDJTools](https://github.com/mikessh/vdjtools/))

This plot can be used to check whether the Ab library is biased towards a particular combination of germline genes due to cloning errors.

5.3 Productivity analysis

The plots described in this section can be found in the `Productivity` tab in `PCR1`'s HTML report. The main factors affecting the productiveness of a clone by `AbSeq`'s interpretation are:

1. Concordance of the coding frames of V and J germline genes
2. Presence of stop codons
3. Presence of indels within a framework or complementarity-determining region

Any sequence that satisfies at least one of the above condition will be classified as unproductive and thus it is unlikely that it will express a functional antibody.

Figure 9 summarizes the productivity analysis results of `PCR1`. Factors that cause sequences to be non-functional are colour coded as follows:

1. Green – sequences *without* stop codons but have a frameshift
2. Cyan – sequences that are *in-frame* but contain at least one stop codon

3. Purple – sequences that contain at least one stop codon *and* have a frameshift

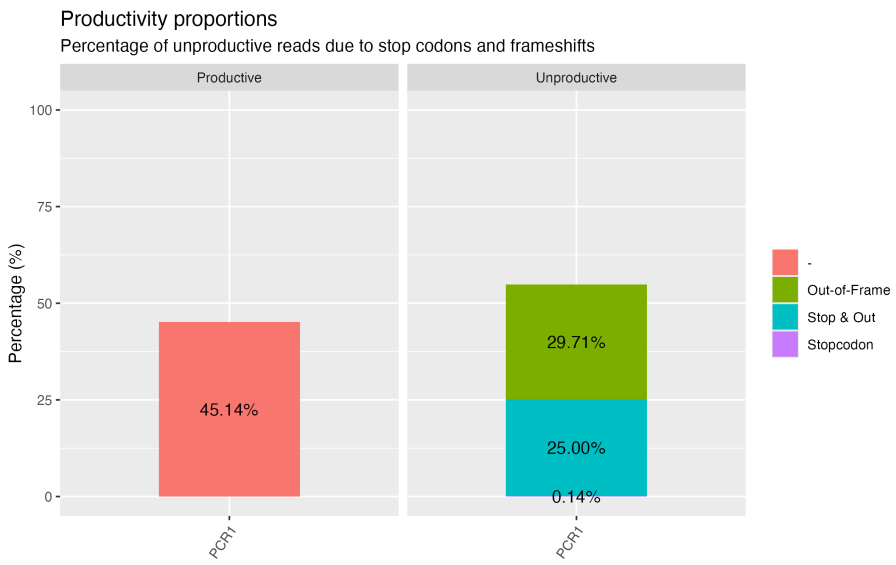

Figure 9: Productivity rate of sequences in PCR1. This plot shows the percentage of productive and unproductive sequences, the reason for unproductive sequences are colour coded.

A good antibody library should have as low unproductive clones as possible. Cloning strategies that are used to clone sequences from the donor libraries or used to construct the Ab library play a key role in this aspect of the library quality.

5.3.1 V-J frameshifts

Figure 10 shows the percentage of clones that are out-of-frame due to either misaligned V-J coding frames or to the presence of non-multiple of three-indels in one of the framework or complementarity-determining regions.

5.3.2 Stop codons

Figure 11 shows the hot spots for stop codons segregated by framework and complementarity-determining regions.

The figure above shows the percentage of stop codons in the FR and CDR regions of *out-of-frame* sequences. As discussed earlier, these stop codons may be introduced due to cloning or sequencing errors, hence a similar plot for *in-frame* sequences can also be found within the same tab.

5.3.3 Indels and mismatches

Some sequences are productive despite having indels and mismatches. This occurs when indels are multiple of three and mismatches do not introduce stop codons. The following figures show the proportion of indels and mismatches for each germline, framework region, and complementarity-determining region on *productive sequences* (unless specified otherwise).

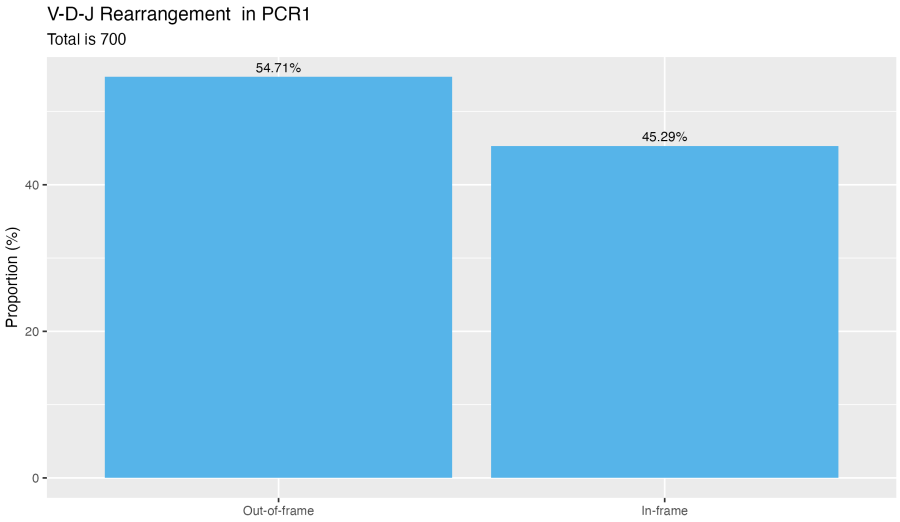

Figure 10: Rate of in-frame and out-of-frame sequences in PCR1. Misaligned V-J frame or non-multiple of 3 indels in either one of the framework or complementarity-determining regions causes a frameshift and therefore renders the sequence unproductive.

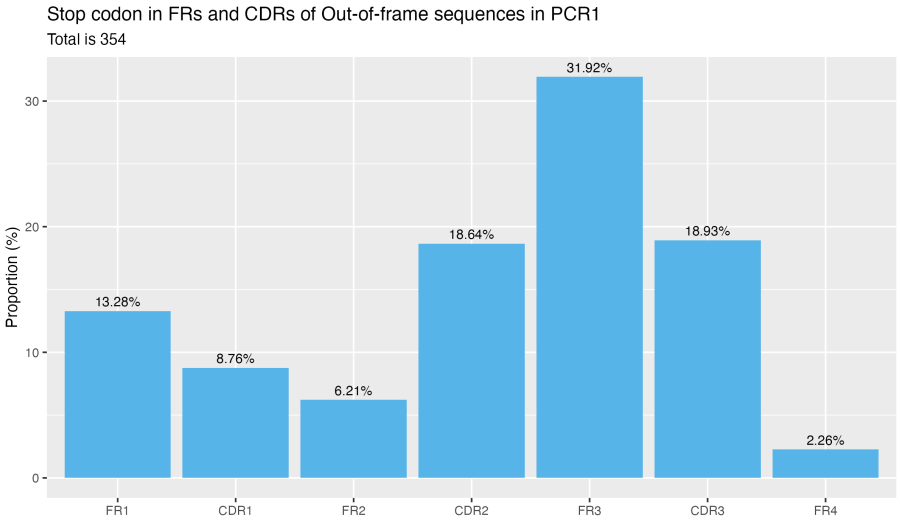

Figure 11: Proportion of stop codons in any given CDR or FR of out-of-frame sequences in PCR1. The percentages show the frequency of stop codons in a given region relative to the total number of stop codons present.

Figure 12, Figure 13, and Figure 14 plots the proportion of mismatches in productive sequences, indels in productive sequences, and indels in out-of-frame (unproductive) sequences for framework region 3 (FR3). The motivation behind these plots is to quickly identify the quality of productive sequences.

Ideally, the number of mismatches in framework regions and IGJ are expected to be low because they are relatively conserved regions. Similarly, the number of indels in productive sequences are expected to be low or some multiple of 3.

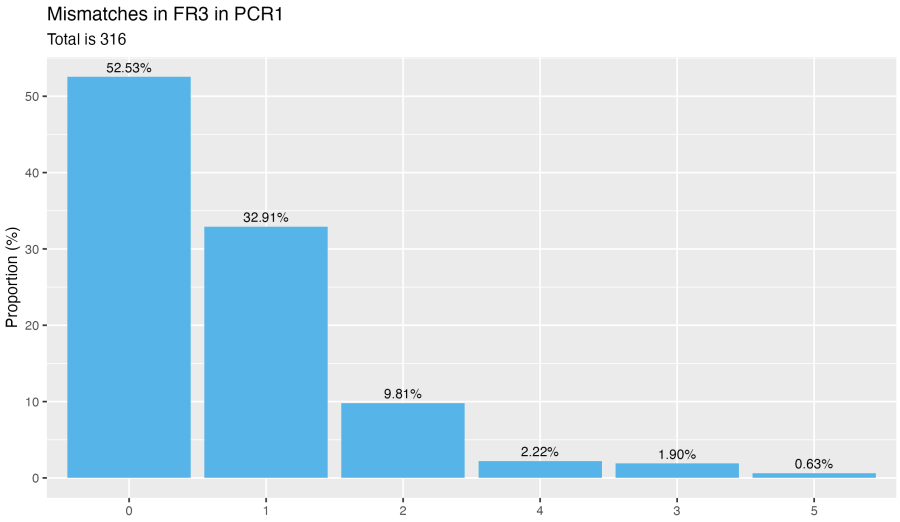

Figure 12: Proportion of mismatches in productive sequences of PCR1.

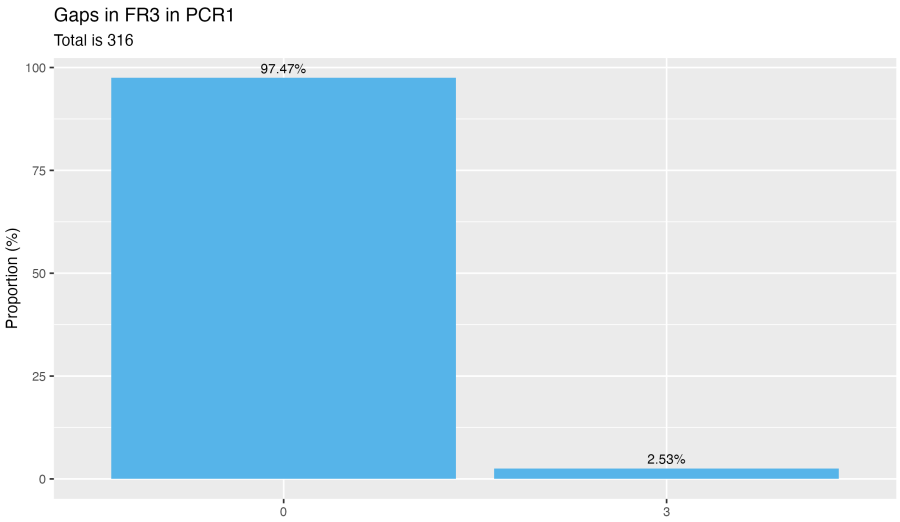

Figure 13: Proportion of indels in productive sequences of PCR1.

Similar plots are generated for other FR and CDR regions, IGV, IGD, and IGJ.

5.4 Diversity analysis

The plots described in this section can be found in the `Diversity` tab of the HTML report. `AbSeq` only conducts diversity analysis on clones that are productive.

5.4.1 Clonotype-based analysis

To estimate repertoire diversity, *abseqR* employs three commonly used techniques:

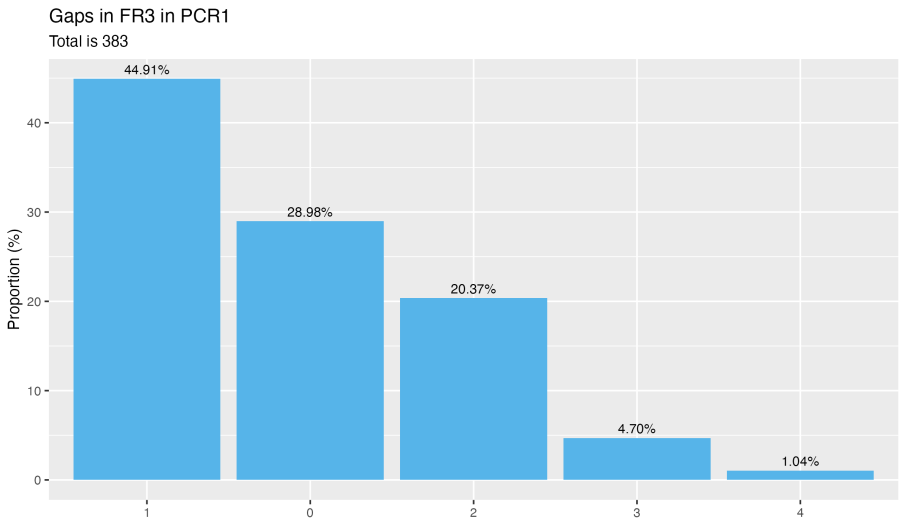

Figure 14: Proportion of indels in out-of-frame sequences of PCR1.

Duplication-level analysis in which the number of times a clone appears in the sequenced sample is calculated. Figure 15 plots the proportion of sequences (y-axis) that appear once (singletons), twice (doubletons), and at higher-orders (x-axis). The higher the percentage of singletones and doubletones, the more diverse the library would likely be.

Rarefaction analysis in which bootstrapping is used to estimate the *richness* of a library by calculating the proportion of unique sequences at different sample sizes. Figure 16 plots the number of deduplicated clonotypes (y-axis) againse sample sizes (x-axis). For each sample size, five samples are drawn and the mean with confidence intervals are calculated. In a highly diverse library, the percentage of unique clones should significantly increase as the sample size increases.

Capture-recapture analysis in Figure 17 plots the percentage of clonotypes recaptured (y-axis) in a capture-recapture experiment at different sample sizes (x-axis). For each sample size, the percentage of recaptured clonotypes is calculated for five repeated capture-recapture experiments and the mean and confidence intervals are reported.

In below figures, the complementarity-determining regions (CDRs) are used to define a "clonotype". Similar plots can be generated for the framework regions (FRs) and the entire variable domain sequences.

5.4.2 Spectratype analysis

Spectratypes are histograms of the clonotype lengths calculated based on the amino acid sequences. Figure 18 shows a CDR3 spectratype. Spectratypes for other FRs and CDRs are available in the same tab. In a good quality library, the framework regions would have quite conserved lengths while CDRs show high length diversity. CDR3 spectratype tends to follow a normal distribution in libraries cloned from naive repertoires.

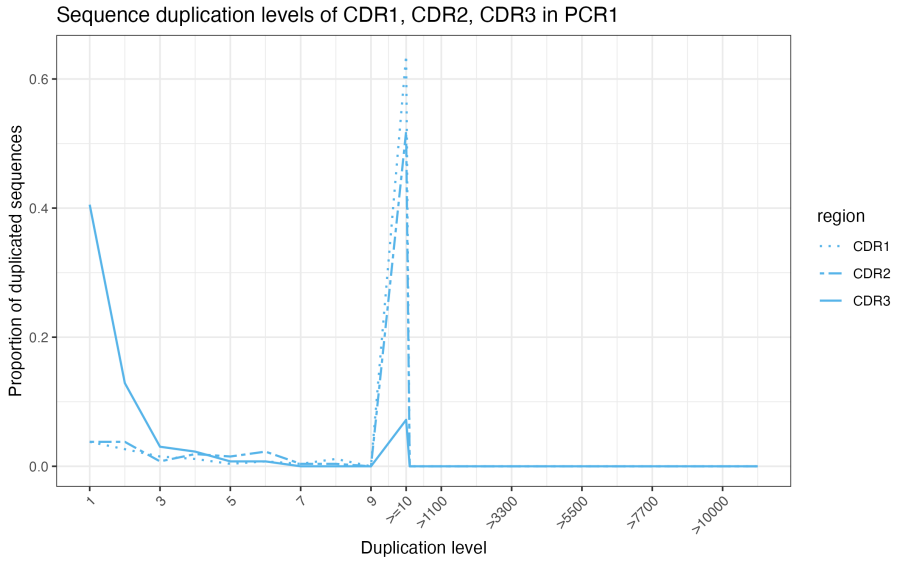

Figure 15: Proportion of duplicated clonotypes in PCR1. Higher-order duplication levels starting from 10 are binned.

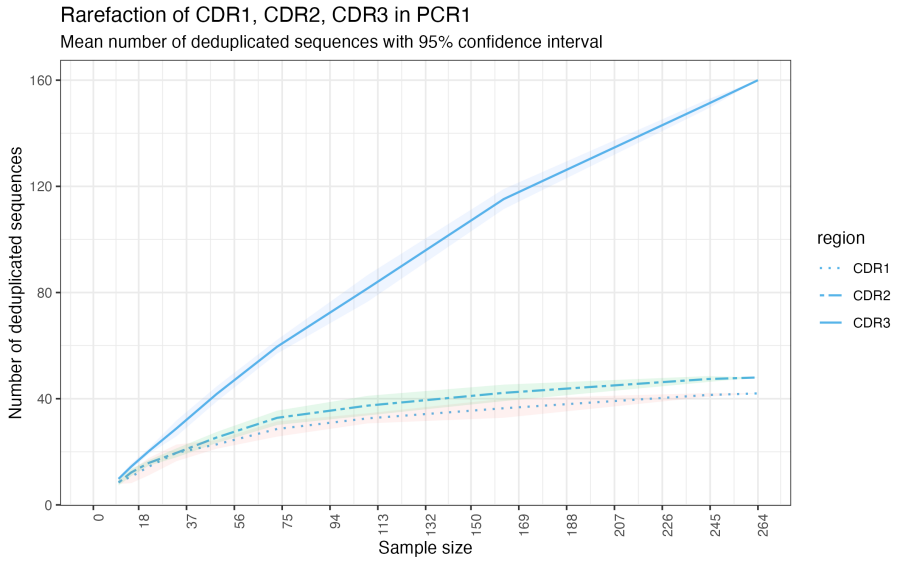

Figure 16: Rarefaction curve of clonotypes in PCR1. The number of deduplicated sequences are taken over the mean of 5 resampling rounds, where the shaded areas indicate 95% confidence interval.

5.4.3 Position-specific analysis

This analysis examines the diversity of Ab library at each amino acid position of the variable domain by estimating the utilization of each of the 20 amino acids at each position. Position-specific frequency matrices (PSFMs) are calculated by aligning all the sequences of a region of interest to anchor at the first position and then the frequency of each amino acid is calculated accordingly. Two PSFMs are calculated: (1) the unscaled PSFM, in which the frequencies are

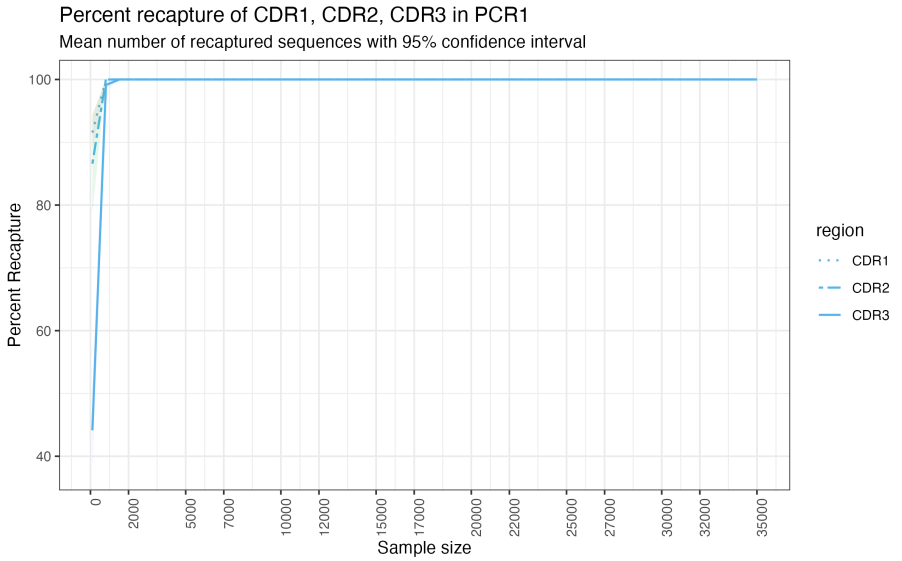

Figure 17: Capture-recapture of clonotypes in PCR1. The number of recaptured sequences are taken over the mean of 5 resampling rounds, where the shaded areas indicate 95% confidence interval.

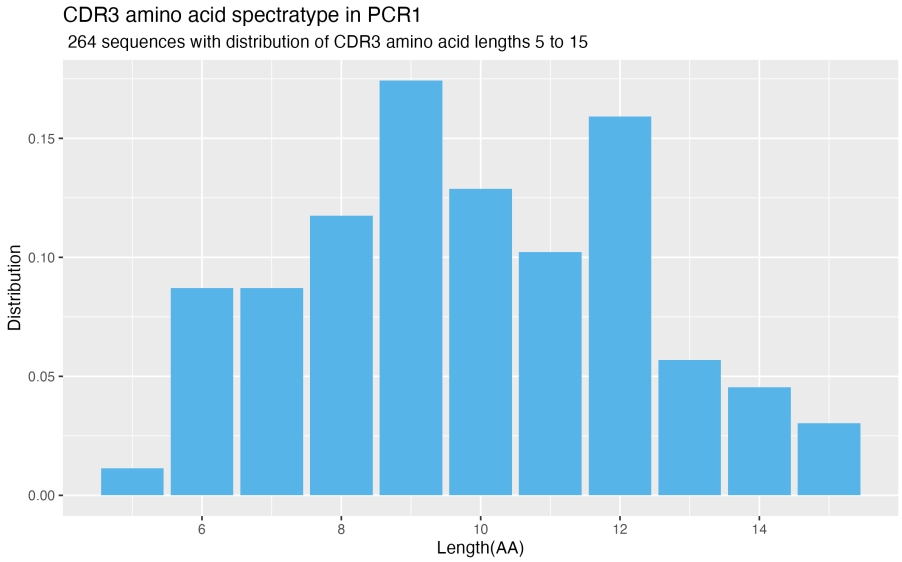

Figure 18: CDR3 amino acid spectratype for PCR1 with outliers removed.

calculated based on the total number of observed sequences per sample *at each position* and (2) the sacled PSFM, in which the frequencies are calculated based on the total number of observed sequences per sample.

Figure 19 shows the PSFM of CDR3 in `PCR1`. Amino acids are coloured based on their physiochemical properties. The left plot shows the unscaled composition logo and the right plot shows the scaled composition logo. Similar plots for other FRs and CDRs are available in the same tab.

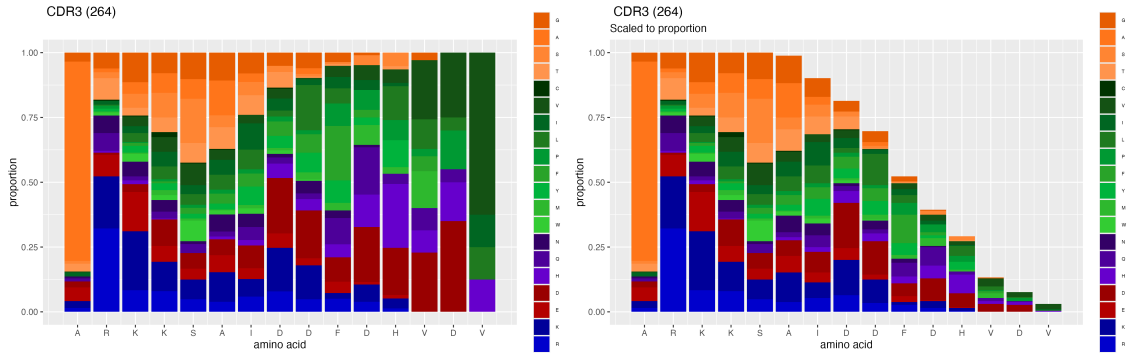

Figure 19: Amino acid composition logo for PCR1's CDR3. The unscaled (left) and the scaled (right) composition logos are colour coded and grouped by their physicochemical properties.

5.5 Comparative analysis

The plots described in this section can be found in the `Clonotypes` tab of `PCR1 vs PCR2 vs PCR3`'s HTML report.

Since comparative analysis deals with *overlapping* clonotypes, this analysis only applies when `compare` was supplied with at least one sample comparison. Earlier, the call to `abseqReport` had `compare = c("PCR1, PCR2, PCR3")`, therefore `PCR1`, `PCR2`, and `PCR3` are compared against each other.

Throughout this analysis, a clonotype is synonymous to its CDR3 amino acid sequence.

5.5.1 Over-representation analysis

Figure 20 offers a simple overview of the top 10 over-represented clones found in each sample. Since the clonotypes are colour coded, overlapping clonotypes can easily be identified within the top 10 of each samples. Note that the proportions are scaled relative to the top 10 clones in the respective samples.

This plot complements the scatter plot mentioned above. It displays the most abundant clonotypes in each sample with the amino acid sequence in the legend.

5.5.2 Overlapping analysis

5.5.2.1 Multi-sample analysis While Figure 20 is capable of showing overlapping clones, it is restricted to the top 10 over-represented clones from each sample. Figure 21 aims to overcome the restriction by using a venn diagram to visualize the number of overlapping (and non-overlapping) clones from each sample. Each number within the venn diagram shows the number of unique clonotypes that are overlapping (in an intersection) or are non-overlapping (not in any intersection). That is, by taking the sum of all the numbers in a sample segment, it becomes the number of unique clonotypes found in that sample.

Note that this venn diagram will **not** be plotted if there are more than 5 samples.

5.5.2.2 Two-sample analysis In order to visualize the correlation between any pair of samples, *abseqR* plots a scatter plot of every possible combination. Figure 22 shows one of them, plotting the clonotype frequencies in `PCR2` against `PCR1`.

The scatter plot:

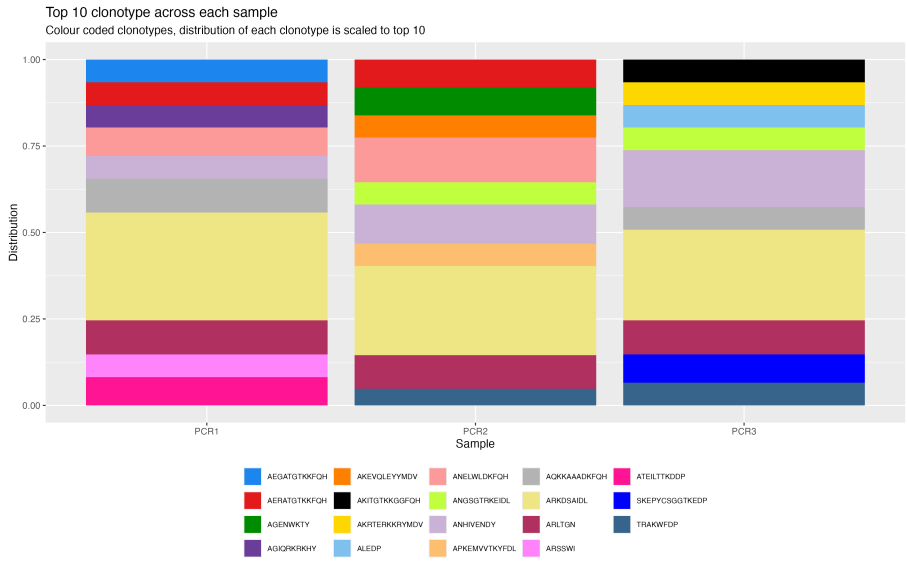

Figure 20: Top 10 clonotypes from PCR1, PCR2, and PCR3. The clonotype proportions displayed are scaled relative to the top 10 clones in each sample.

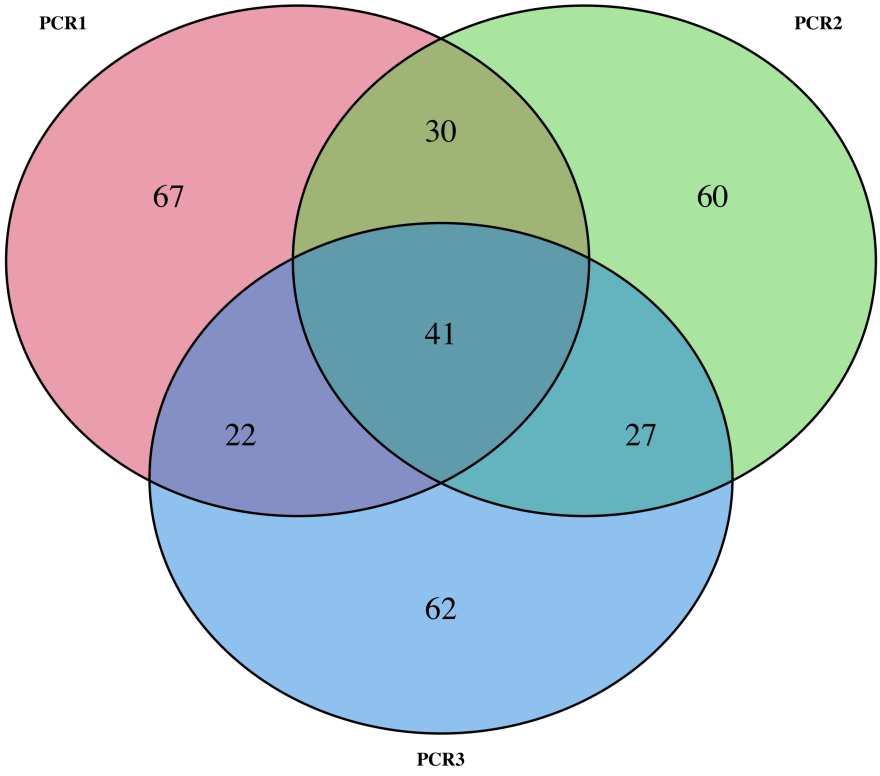

Figure 21: Number of unique overlapping clones found in PCR1, PCR2, and PCR3. The numbers in an intersection denotes the number of unique clones that are shared between the samples involved in the said intersection.

- has log_{10}-scaled clonotype frequencies
- has a point for each clonotype
  - the coordinate of each point denotes the $log_{10}(frequency)$ in the sample on the x and y axis respectively
  - point sizes are mapped to the mean frequency of a clonotype
- has marginal density plots colour coded as such:
  - blue: density of *overlapping clonotypes*
  - purple: density of *non-overlapping clonotypes*
  - grey: density of *all clonotypes*

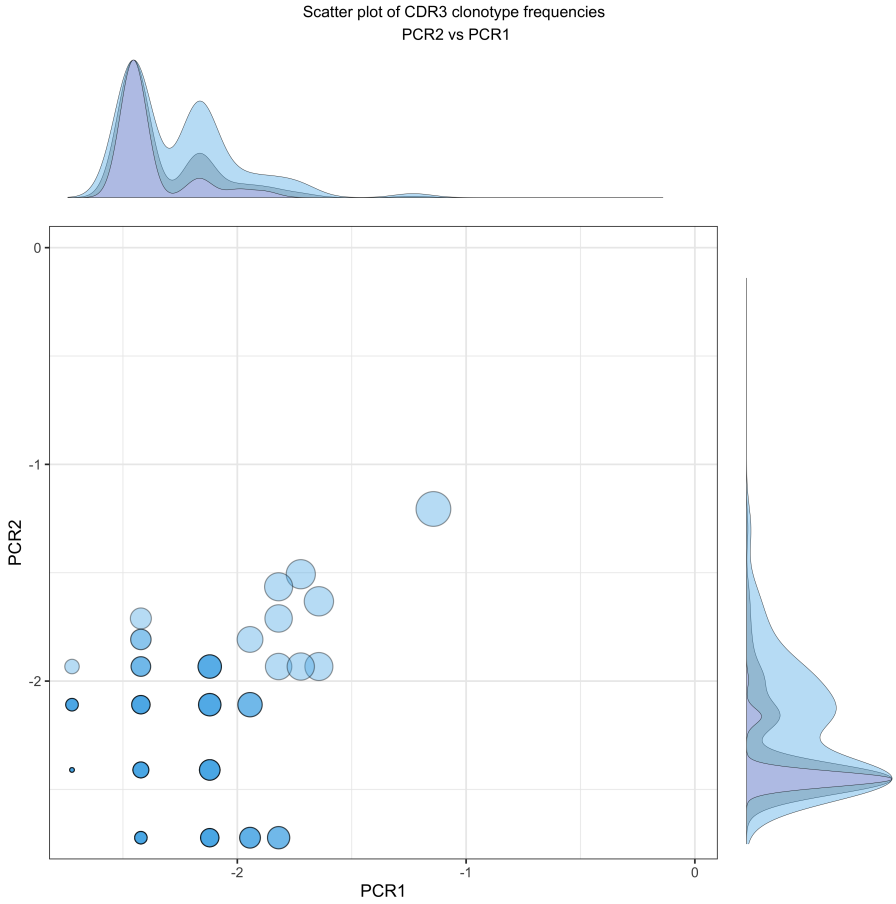

Figure 22: Log-scaled scatter plot of clonotype frequencies in PCR1 and PCR2. Point size denotes mean frequency of the 2 samples and marginal density plots are colour coded by overlapping (blue), non-overlapping (purple), and all (grey) clonotypes.

This plot is heavily inspired by VDJTools.

5.5.3 Correlation analysis

In addition to Figure 22, the linear correlation of clonotype frequencies between samples can be directly quantified using Pearson's correlation coefficient. Figure 23 shows the plot generated by *ggcorrplot* used to visualize pearson coefficients. A similar plot using Spearman's correlation coefficient (rank-based) is also available in the same directory.

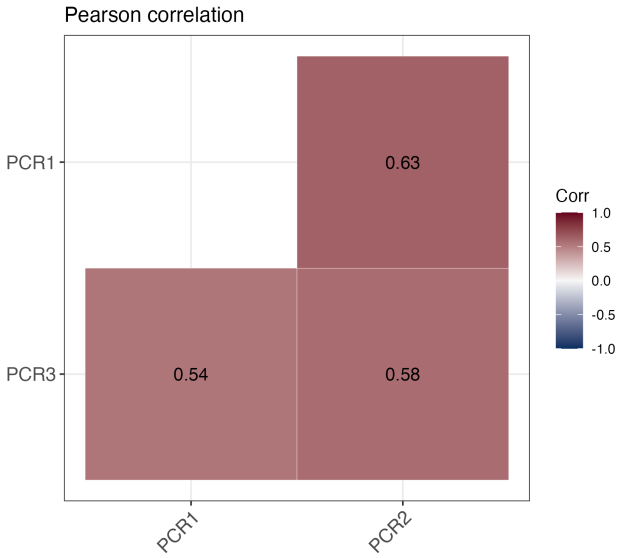

Figure 23: Pearson correlation between PCR1, PCR2, and PCR3. Values denote the pearson correlation coefficient of the clonotypes frequencies between the samples. These values will appear with a cross 'x' to signify an insignificant coefficient if the p-value of the coefficient is larger than 0.05.

5.5.4 Clustering analysis

The *vegan* package was used to calculate distances between samples. The distances between samples are calculated using its clonotype frequencies by applying methods from Morisita-Horn's overlap index, Jaccard index, and Dice's coefficient.

Figure 24 shows a dendrogram plotted using Morisita-Horn's overlap index. The length of each line denotes the distance between the 2 samples or clusters it is connected to. Other dendrograms using Jaccard and Dice's formula are available in the same directory.

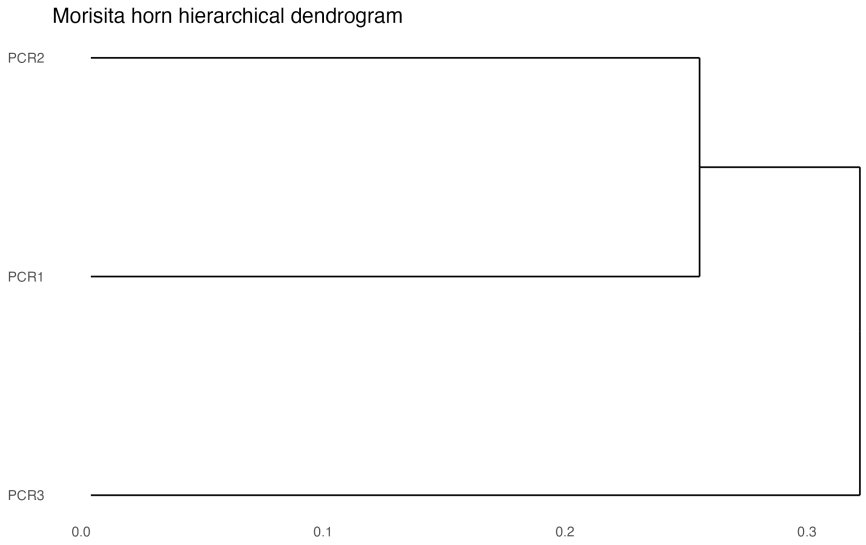

Figure 24: Morisita-Horn distances of PCR1, PCR2, and PCR3. The distances are calculated using clonotype frequencies and are visualized as the length of the lines connecting samples or clusters.

6 Appendices

6.1 Datasets

The datasets used in the above examples was obtained from a combination of synthetic sample datasets generated using MiXCR's program here. Firstly, three distinct samples were generated, each simulated with the following parameters in `MiXCR`:

Parameter	sample 1	sample 2	sample 3
reads	10000	10000	10000
clones	5000	5000	2000
seed	4228	2428	2842
conf	MiSeq-300-300	MiSeq-300-300	MiSeq-300-300
loci	IGH	IGH	IGH
species	hsa	hsa	hsa

Following that, an arbitrary number of sequences were randomly drawn from each of the three samples and randomly amplified. This process was repeated 3 times, resulting in a final repertoire of three samples, named PCR1, PCR2, and PCR3.

Finally, these three samples were analyzed by abseqPy. The command used to analyze these samples are as follows:

```
abseq -y params.yml
```

where the contents of `params.yml` is:

```
# params.yml
defaults:
    bitscore: 300
    sstart: 1-3
    alignlen: 250
    outdir: ex
    task: all
    threads: 1
---
file1: PCR1.fasta
name: PCR1
---
file1: PCR2.fasta
name: PCR2
---
file1: PCR3.fasta
name: PCR3
```

abseqPy's analysis output on these three samples are contained within the dataset described in this vignette.

6.2 Session Info

This vignette was rendered in the following environment:

```
## R version 4.5.0 RC (2025-04-04 r88126)
## Platform: aarch64-apple-darwin20
## Running under: macOS Ventura 13.7.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] gridExtra_2.3    plotly_4.10.4    ggplot2_3.5.2    png_0.1-8
## [5] abseqR_1.26.0    BiocStyle_2.36.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.6         circlize_0.4.16      shape_1.4.6.1
##  [4] xfun_0.52            bslib_0.9.0          htmlwidgets_1.6.4
##  [7] GlobalOptions_0.1.2  lattice_0.22-7       crosstalk_1.2.1
## [10] vctrs_0.6.5          tools_4.5.0          generics_0.1.3
## [13] parallel_4.5.0       tibble_3.2.1         cluster_2.1.8.1
## [16] pkgconfig_2.0.3      Matrix_1.7-3         data.table_1.17.0
## [19] RColorBrewer_1.1-3   VennDiagram_1.7.3    lifecycle_1.0.4
## [22] farver_2.1.2         compiler_4.5.0       stringr_1.5.1
## [25] textshaping_1.0.0    tinytex_0.57         munsell_0.5.1
## [28] codetools_0.2-20     permute_0.9-7        htmltools_0.5.8.1
## [31] sass_0.4.10          lazyeval_0.2.2       yaml_2.3.10
## [34] tidyr_1.3.1          pillar_1.10.2        jquerylib_0.1.4
## [37] MASS_7.3-65          BiocParallel_1.42.0  cachem_1.1.0
## [40] vegan_2.6-10         flexdashboard_0.6.2  ggcorrplot_0.1.4.1
## [43] nlme_3.1-168         tidyselect_1.2.1     digest_0.6.37
## [46] stringi_1.8.7        purrr_1.0.4          dplyr_1.1.4
## [49] reshape2_1.4.4       bookdown_0.43        labeling_0.4.3
## [52] splines_4.5.0        fastmap_1.2.0        colorspace_2.1-1
## [55] cli_3.6.4            magrittr_2.0.3       withr_3.0.2
## [58] scales_1.3.0         httr_1.4.7           rmarkdown_2.29
## [61] lambda.r_1.2.4       futile.logger_1.4.3  ragg_1.4.0
## [64] evaluate_1.0.3       knitr_1.50           viridisLite_0.4.2
## [67] mgcv_1.9-3           rlang_1.1.6          ggdendro_0.2.0
## [70] futile.options_1.0.1 Rcpp_1.0.14          glue_1.8.0
## [73] BiocManager_1.30.25  formatR_1.14         jsonlite_2.0.0
## [76] R6_2.6.1             plyr_1.8.9           systemfonts_1.2.2
```

7 References