bcRep: Advanced Analysis of B Cell Receptor Repertoire Data

Julia Bischof *Julia.Bischof@uksh.de*

2015-10-28

Contents

Introduction

Th bcRep package helps to analyze IMGT/HighV-QUEST output, in more detail. It functions well with B cells, but can also be used for T cell data, in some cases. Using this package you can read IMGT/HighV-QUEST output files and study sequences and clones. In special their functionality, junction frames, gene usage and mutations. Functions to analyze clones out of the IMGT/HighV-QUEST output, but also to compare sequences and clones, are provided.

Package features

- Handling IMGT/HighV-QUEST output: combine several output folders; read IMGT tables
- Data manipulation: filtering sequences or clones for functionality or junction frames
- Descriptive statistics: functionality and junction frame usage; CDR3 length distribution
- Clonotype analysis: cluster sequences to clones
- Gene usage analysis
- Basic mutation analysis
- Amino acid distribution and diversity: amino acid distribution, richnes, Shannon index, inverse Shannon index, Gini index
- Comparison of different samples: gene usage, amino acid distribution, diversity, clones
- several visualization methods

General information and package data

Before installing bcRep, following packages need to be installed: `vegan, gplots, ineq, parallel, doParallel, foreach`.

Parallel processing

Parallel processing is possible for functions `clones()`, `clones.shared()`, `sequences.geneComb()`, `compare.aaDistribution()`, `compare.geneUsage()` and `compare.trueDiversity`. Using only one core is the default parameter.

Datasets

There are several datasets provided in the package. Most of them are examples of IMGT/HighV-QUEST output; two additional files represent clonotype files:

```
library(bcRep)
data(summarytab) # An extract from IMGT/HighV-QUEST output file 1_Summary(...).txt
data(ntseqtab) # An extract from IMGT/HighV-QUEST output file 3_Nt-sequences(...).txt
data(aaseqtab) # An extract from IMGT/HighV-QUEST output file 5_AA-sequences(...).txt
data(mutationtab) # An extract from IMGT/HighV-QUEST output file
                  # 7_V-REGION-mutation-and-AA-change-table(...).txt

data(clones.ind) # Clonotypes of one individual
data(clones.allind) # Clonotypes of eight individuals

# Get first 6 lines of each file
head(summarytab)
head(ntseqtab)
```

```
head(aaseqtab)
head(mutationtab)
head(clones.ind)
head(clones.allind)
```

Processing of IMGT/HighV-QUEST data

IMGT/HighV-QUEST can process datasets with up to 500.000 sequences. If you like to study bigger datasets, you have to split the input FASTA-file into smaller datasets and upload them individually to IMGT. Afterwards you can use function `combineIMGT()` to combine several folders.

```
# Example: combine folders IMGT1a, IMGT1b and IMGT1c to a new datset NewProject
combineIMGT(folders = c("pathTo/IMGT1a", "pathTo/IMGT1b", "pathTo/IMGT1c"),
            name = "NewProject")
```

To read IMGT/HighV-QUEST output files, use `readIMGT("PathTo/file.txt",filterNoResults=TRUE)`. You can choose between including or excluding sequences without any information (that are lines with a sequence ID, but "No results"). Spaces and "-" in headers will be replaced by "_". If there is a special table required as input for a function, this is mentioned in the corresponding help file.

Amino acid distribution

Amino acid distribution can be analyzed using `aaDistribution()` and visualized with `plotAADistribution()`.

This function returns a list containing proportions of all amino acids (including stop codons "*") for each analyzed sequence length. Optionally, also the number used for analysis can be given.

Figures can be saved as PDF to the working directory.

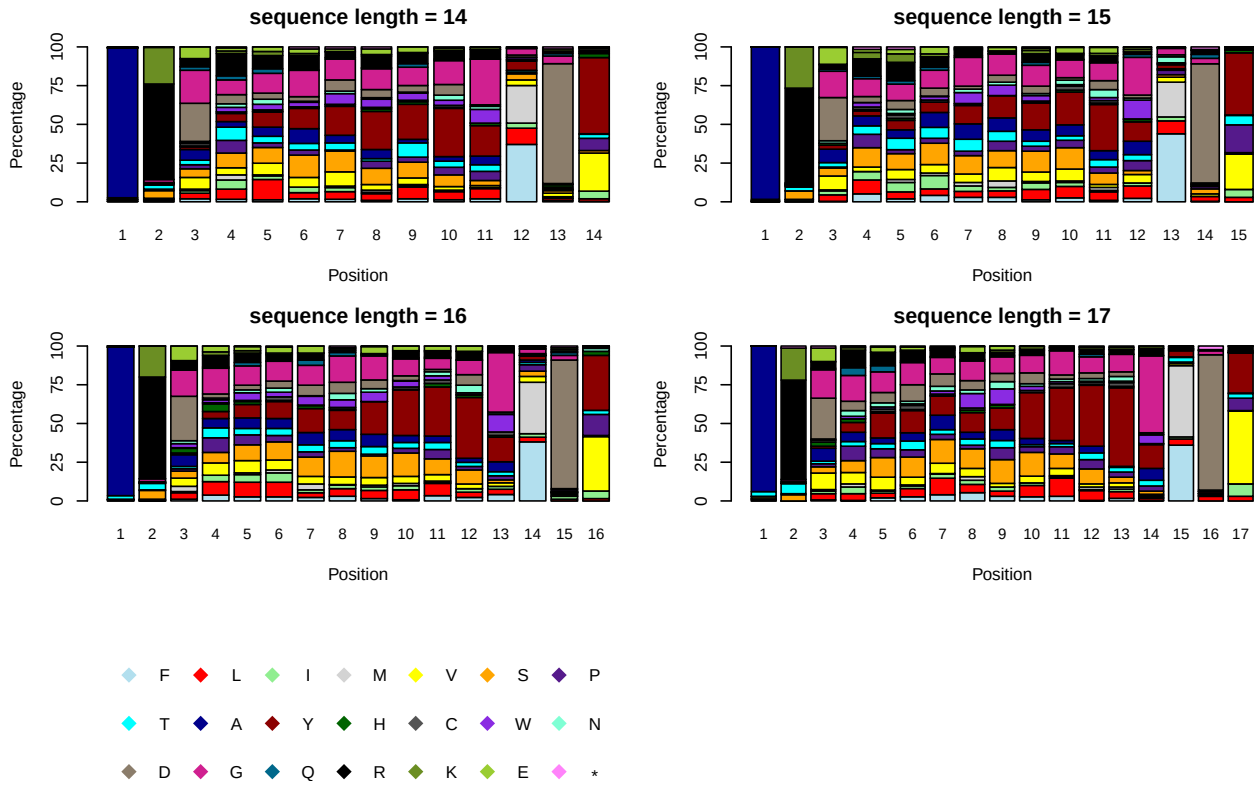
```
# Example:
data(aaseqtab)
aadistr<-aaDistribution(sequences = aaseqtab$CDR3_IMGT, numberSeq = TRUE)

# First 4 columns of Amino acid distribution table:
        # for a sequence length of 13 AA (* = stop codon)
        # (aadistr$Amino_acid_distribution$`sequence length = 13`)
```

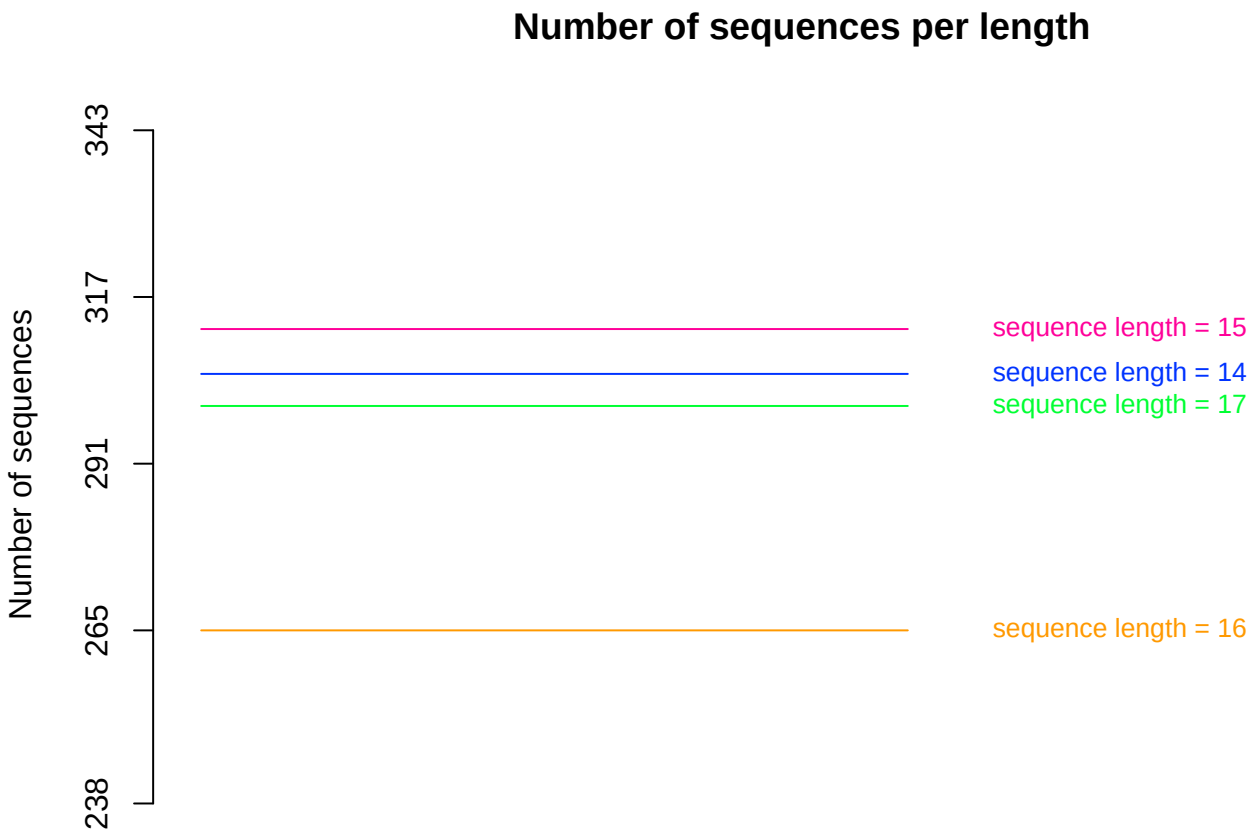
	Position1	Position2	Position3	Position4
F	0	0	0.007018	0.01053
L	0	0.003509	0.04561	0.08772
I	0	0.02105	0.02456	0.03158
M	0	0	0.01404	0.01754
V	0.003509	0.007018	0.09123	0.04561
S	0	0.07719	0.08421	0.07368
P	0.003509	0	0.02105	0.07018
T	0.02105	0.05614	0	0.03158
A	0.9614	0.003509	0.05965	0.05614

	Position1	Position2	Position3	Position4
Y	0	0	0.01754	0.05263
H	0	0.01404	0.01053	0.007018
C	0	0	0.02105	0.007018
W	0	0	0.007018	0.02807
N	0	0.01404	0.007018	0.04211
D	0	0	0.2281	0.06316
G	0.003509	0.01053	0.1825	0.207
Q	0	0	0.01053	0.01404
R	0.007018	0.607	0.06316	0.0807
K	0	0.186	0.007018	0.03509
E	0	0	0.09825	0.0386
*****	0	0	0	0

```
# Plot example for sequence lengths of 14-17 amino acids:
aadistr.part<-list(aadistr$Amino_acid_distribution[13:16],
                   data.frame(aadistr$Number_of_sequences_per_length[13:16,]))
names(aadistr.part)<-names(aadistr)
plotAADistribution(aaDistribution.tab=aadistr.part, plotAADistr=TRUE,
                   plotSeqN=TRUE, PDF=NULL)
```

Amino acid distribution





Diversity

Diversity of sequences and clones can be analyzed using `trueDiversity()` or `clones.giniIndex()`.

True diversity

Using `trueDiversity()` richness or diversity of sequences with the same length can be analyzed. Basically diversity of amino acids per position is calculated. True diversity can be measured for orders q = 0, 1 or 2. `plotTrueDiversity()` can be used for visualization.

Order 0: Richness (in this case it represents number of different amino acids per position).

Order 1: Exponential function of Shannon entropy using the natural logarithm as the base (weights all amino acids by their frequency).

Order 2: Inverse Simpson entropy (weights all amino acids by their frequency, but weights are given more to abundant amino acids).

These indices are very similar (Hill, 1973). For example the exponential function of Shannon index is linearly related to inverse Simpson.

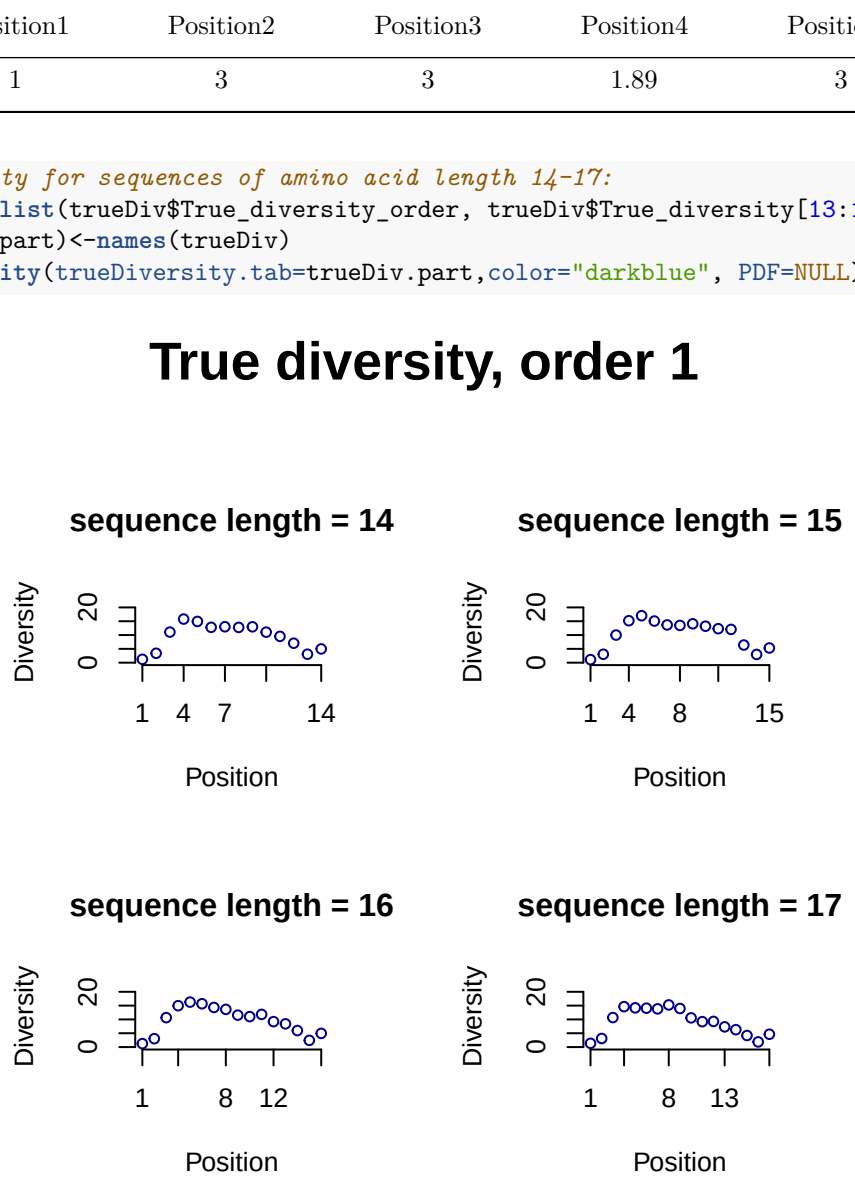
Amino acid sequences can be found in IMGT output table `5_AA-sequences(...).txt`.

```
# Example:
data(aaseqtab)
trueDiv<-trueDiversity(sequences = aaseqtab$CDR3_IMGT, order = 1)
        # using exponent of Shannon entropy
```

```
# True diversity of order 1 for amino acid length of 5 AA
        # (trueDiv$True_diversity$'sequence length = 5')
```

Position1	Position2	Position3	Position4	Position5
1	3	3	1.89	3

```
# True diversity for sequences of amino acid length 14-17:
trueDiv.part<-list(trueDiv$True_diversity_order, trueDiv$True_diversity[13:16])
names(trueDiv.part)<-names(trueDiv)
plotTrueDiversity(trueDiversity.tab=trueDiv.part,color="darkblue", PDF=NULL)
```



Gini index

`clones.giniIndex()` calculates the Gini index of clones. Input is a vector containing clone sizes (copy number). The Gini index measures the inequality of clone size distribution. It's between 0 and 1. An index of 0 represents a polyclonal distribution, where all clones have same size. An index of 1 represents a perfect monoclonal distribution.

If a PDF project name is given, the Lorenz curve will be returned, as well. Lorenz curve can be interpreted as p*100 percent have L(p)*100 percent of clone size.

```
# Example:
data(clones.ind)
clones.giniIndex(clone.size=clones.ind$total_number_of_sequences, PDF = NULL)
# [1] 0.7473557
```

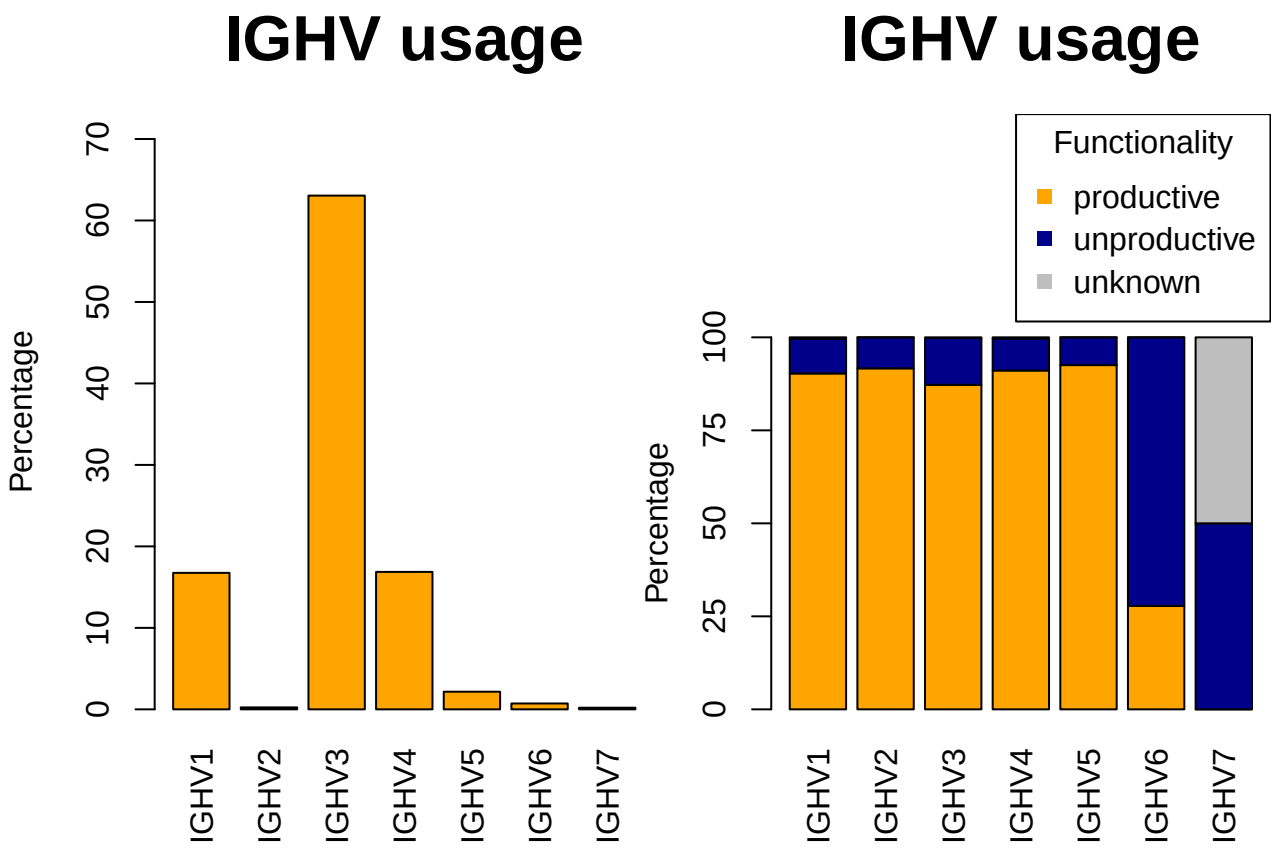
Gene usage

Gene usage can be analyzed using functions `geneUsage()` and `sequences.geneComb()`.

It can be differentiated between subgroup (f.e. IGHV1), gene (f.e. IGHV1-1) or allele (f.e. IGHV1-1*2), as well as between relative or absolute values.

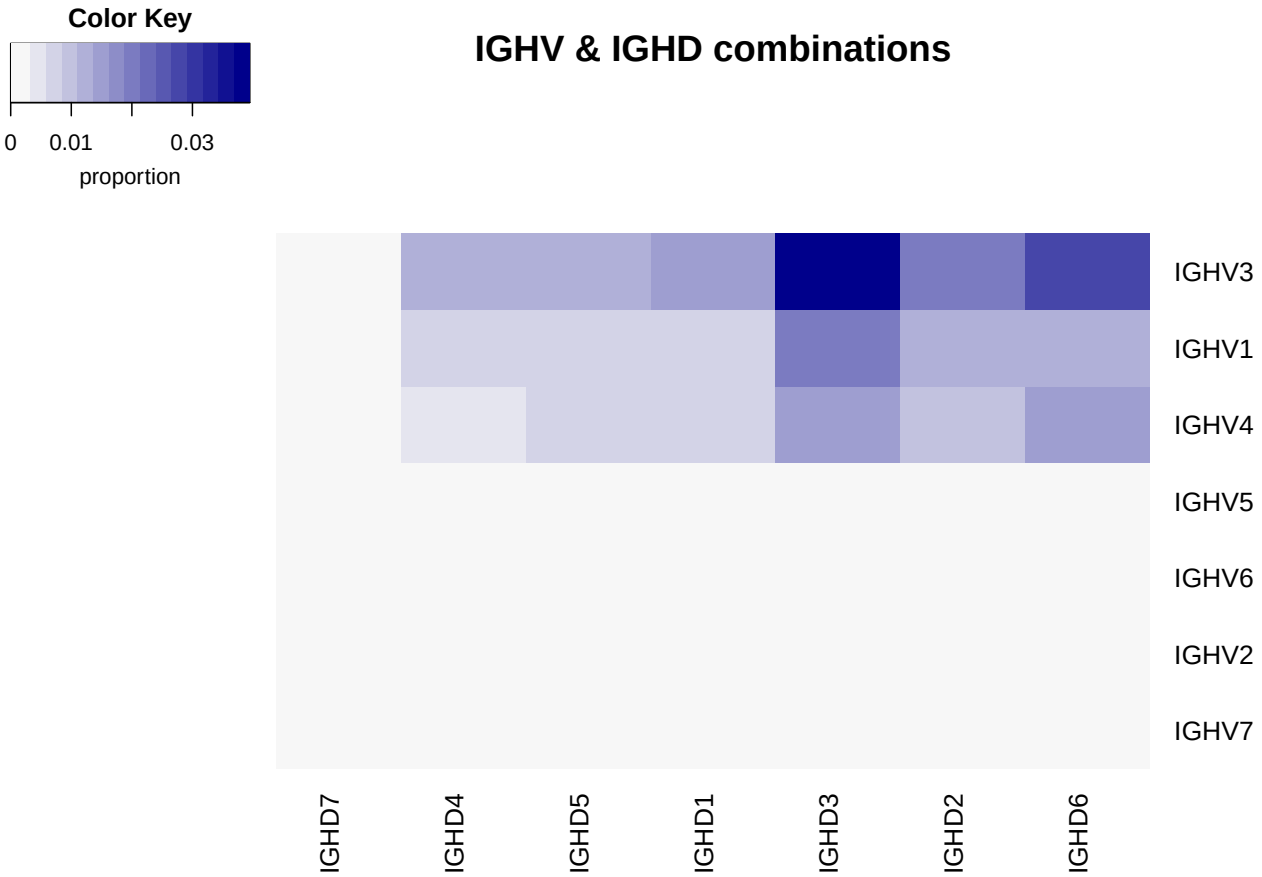
When using function `geneUsage()`, functionality and junction frame usage, dependent on gene usage can be studied. Important ist, that input vectors have same order. Single genes per line, as well as several genes per line, can be processed. IMGT nomenclature has to be used (f.e. "Homsap IGHV4-34*01")!

```
# Example:
data(summarytab)
Vgene<-geneUsage(genes = summarytab$V_GENE_and_allele, level = "subgroup",
                 functionality = summarytab$Functionality)
plotGeneUsage(geneUsage.tab = Vgene, plotFunctionality = TRUE, PDF = NULL,
              title = "IGHV usage")
```



Using function `sequences.geneComb()`, gene/gene ratios of two gene families will be analyzed. IMGT nomenclature has to be used (f.e. "Homsap IGHV4-34*01"), again. Results can be plotted with gene/gene combination, where one or both genes are unknown, or without.

```r
# Example:
data(summarytab)
VDcomb.tab<-sequences.geneComb(family1 = summarytab$V_GENE_and_allele,
                family2 = summarytab$D_GENE_and_allele,
                level = "subgroup", abundance = "relative")
plotGeneComb(geneComb.tab = VDcomb.tab, withNA = FALSE, PDF = NULL)
```



Functions for a set of sequences

There are some functions, which filter or summarize IMGT/HighV-QUEST output data:

Filter sequences

Sequence files can be filtered for functionality or junction frame usage:

- sequences.getProductives(): filters for productive sequences
- sequences.getUnproductives(): filters for unproductive sequences
- sequences.getAnyFunctionality(): excludes sequences without any functionality information
- sequences.getInFrames(): filters for in-frame sequences
- sequences.getOutOfFrames(): filters for out-of-frame sequences

- sequences.getAnyJunctionFrame(): excludes sequences without any junction frame information

The function looks for a column containing functionality or junction frame information and filters the whole table. All columns will remain, rows will be filtered. Junction frame information is only available in the IMGT summary table (`1_Summary(...).txt`). Functionality information is available in every table.

```
# Example: filter for productive sequences
data(summarytab)
ProductiveSequences<-sequences.getProductives(summarytab)
# dimension of the summary table [rows, columns]:
dim(summarytab)
## [1] 3000   29
# dimension of the summary table, filtered for productive sequences [rows, columns]:
dim(ProductiveSequences)
## [1] 2645   29
```

Functionality and junction frames

Some basic statistics about functionality and junction frames can be given by `sequences.functionality()` and `sequences.junctionFrame()`. Both functions return absolute or relative values for each kind of sequences.

```
# Example:
data(summarytab)
sequences.functionality(data = summarytab$Functionality, relativeValues=TRUE)

# Proportion of productive and unproductive sequences:
```

productive	unproductive	unknown
0.8817	0.1127	0.005667

Mutations

Some basic mutation analysis can be done with `sequences.mutation()`. Input is the IMGT output table `7_V-REGION-mutation-and-AA-change-table(...).txt` and optionally `1_Summary(...).txt`. Mutation analysis can be done for V-region, FR1-3 and CDR1-2 sequences. Functionality, as well as junction frame analysis can be included. Further the R/S ratio (ratio of replacement and silent mutations) can be returned.

```
data(mutationtab)
data(summarytab)
V.mutation<-sequences.mutation(mutationtab = mutationtab, summarytab = summarytab,
                               sequence = "V", junctionFr = TRUE, rsRatio=TRUE)

# V.mutation$Number_of_mutations, first 6 lines:
```

V_REGION_identity_nt	number_mutations	number_replacement
244/245 nt	1	1

V_REGION_identity_nt	number_mutations	number_replacement
244/244 nt	0	0
243/244 nt	1	0
240/247 nt	7	4
220/248 nt	29	21
241/244 nt	3	1

Table 4: Table continues below

number_silent	RS_ratio
0	0
0	0
1	0
3	1.333
8	2.625
2	0.5

```
# V.mutation$Junction frame:
```

	proportion
mutation_in_in-frame_sequences	0.6677
mutations_in_out-of-frame_sequences	0.005
mutations_in_sequences_with_unknown_JUNCTION_frame	0.066
no_mutation_in_in-frame_sequences	0.2417
no_mutations_in_out-of-frame_sequences	0.001333
no_mutations_in_sequences_with_unknown_JUNCTION_frame	0.01833

Functions for a set of clones

BcRep provides some functions to study clone features. Some columns of the clonotype file can simply be plotted using `boxplot()` or `barplot()`.

Defining clones and shared clones

Clones can be defined from a set of sequences, using `clones()`. Criteria for sequences, belonging to the same clones are:

- same CDR3 (amino acid) length and a identity of a given treshold (it's possible, to look for same CDR3 sequences or for a CDR3 identity of f.e. 85%)

- same V gene
- same J gene (optional)

Output of the clone table can be individually specified (see help), but following columns are mandatory:

- shared CDR3 amino acid sequence(s)
- CDR3 amino acid sequence length
- Number of unique CDR3 sequences (each CDR3 sequence is mentioned once)
- Number of all CDR3 sequences (CDR3 sequences are listed with duplicates)
- Sequence count per CDR3
- V gene (as used for analysis)
- V gene and allele (IMGT nomenclature)
- J gene (as used for analysis)
- J gene and allele (IMGT nomenclature)

Optionally columns like D gene, CDR3 amino acid sequences (f.e. for diversity analysis), CDR3 nucleotide sequences, functionality, junction frames and so on, can be specified.

```
# Example:
data(aaseqtab)
data(summarytab)
clones.tab<-clones(aaseqtab = aaseqtab, summarytab = summarytab, ntseqtab = NULL,
                   identity = 0.85, useJ = TRUE, dispD = FALSE, dispSeqID = FALSE,
                   dispCDR3aa = FALSE, dispCDR3nt = FALSE,
                   dispJunctionFr.ratio = FALSE, dispJunctionFr.list = FALSE,
                   dispFunctionality.ratio = FALSE, dispFunctionality.list = FALSE,
                   dispTotalSeq = FALSE, nrCores=1)

# Example of a clonotype file (not from the command above)
data(clones.ind)
```

Filter clones for their size

Using `clones.filterSize()` you can filter a set of clonotypes by their size. There are 3 methods for filtering:

Giving a treshold for the parameter

- `number`, filters the table for a given number of sequences; f.e. 20 smallest clones
- `propOfClones`, filters the table for a proportion of the total number of clones, f.e. the 10% biggest clones
- `propOfSequences`, filters the table for a proportion of all sequences, beloging to all clones; f.e. clones, that include 1% of all sequences

Further it can be filtered for both ends (biggest and smallest clones; `two.tailed`), or only for biggest (`upper.tail`) or smallest (`lower.tail`) clones. Biggest clones means clones with biggest sequence number/size/copy number.

```
# Example 1: Getting the 3 smallest clones (clones with 85% CDR3 identity)
clones.filtered1<-clones.filterSize(clones.tab=clones.ind,
                                    column="total_number_of_sequences",
                                    number=3, method="lower.tail")
```

```
    ## Output of clones.filtered1[,c("total_number_of_sequences",
                            ## "unique_CDR3_sequences_AA", "CDR3_length_AA", "V_gene")]:
```

total_number_of_sequences	unique_CDR3_sequences_AA
2	GALITMVRGVISWRFDP, GALIAMVRGVISWRFDP
2	ARGL*QRLVL, VRGL*QRLVL
2	AKDRYGDYALY, ARDRYGDYALY

Table 7: Table continues below

CDR3_length_AA	V_gene
17	IGHV4-4
10	IGHV4-4
11	IGHV4-4

```
# Example 2: Getting 10% biggest and smallest clones
    ## (clones with 85% CDR3 identity; column 4 = "total_number_of_sequences")
clones.filtered2<-clones.filterSize(clones.tab=clones.ind, column=4, propOfClones=0.1,
                                    method="two.tailed")
names(clones.filtered2) # a list with biggest and smallest 10% clones
## [1] "upper.tail" "lower.tail"
dim(clones.filtered2$upper.tail) # dimension of table with biggest clones [rows, columns]
## [1] 100  14
dim(clones.filtered2$lower.tail) # dimension of table with smallest clones [rows, columns]
## [1] 100  14

# Example 3: Getting clones, that include 0.5% of all sequences
    ## (clones with 85% CDR3 identity; column 4 = "total_number_of_sequences"")
clones.filtered3<-clones.filterSize(clones.tab=clones.ind, column=4,
                                    propOfSequences=0.005, method="two.tailed")
names(clones.filtered3) # a list with biggest and smallest 10% clones
## [1] "upper.tail" "lower.tail"
dim(clones.ind) ## dimension of clones.ind table [rows, columns]
## [1] 1000   14
## --> number of rows is equal to number of rows of biggest and smallest clones
dim(clones.filtered3$upper.tail) # dimension of table with biggest clones [rows, columns]
## [1] 48 14
dim(clones.filtered3$lower.tail) # dimension of table with smallest clones [rows, columns]
## [1] 952  14
```

Filter clones for functionality or junction frame usage

`clones.filterFunctionality()` and `clones.filterJunctionFrame` filter set of clones for functionality or junction frame usage.

- Filtering for functionality: you can filter for clones including only productive or only unproductive sequences

```
# Example
data(clones.ind)
productiveClones<-clones.filterFunctionality(clones.tab = clones.ind,
                                             filter = "productive")
# dimension of clones.ind [rows, columns]:
dim(clones.ind)
## [1] 1000   14
# dimension of clones.ind, filtered for productive sequences [rows, columns]:
dim(productiveClones)
## [1] 1000   14
```

- Filtering for junction frame usage: you can filter for clones including only in-frame or only out-of-frame sequences

```
# Example
data(clones.ind)
inFrameClones<-clones.filterJunctionFrame(clones.tab = clones.ind, filter = "in-frame")
# dimension of clones.ind [rows, columns]:
dim(clones.ind)
## [1] 1000   14
# dimension of clones.ind, filtered for in-frame sequences [rows, columns]:
dim(inFrameClones)
## [1] 873  14
```

Clone features

There are some function provided to analyse clone features, like CDR3 length distribution or clone copy number.

CDR3 length distribution of clones Using `clones.CDR3Length()` CDR3 length distribution of clones can be measured. The corresponding visualization method is `plotClonesCDR3Length()`. Input for both functions are vectors containing nucleotide or amino acid sequences. Further functionality and junction frame usage, depending on CDR3 length can be studied. Absolute or relative values can be returned.

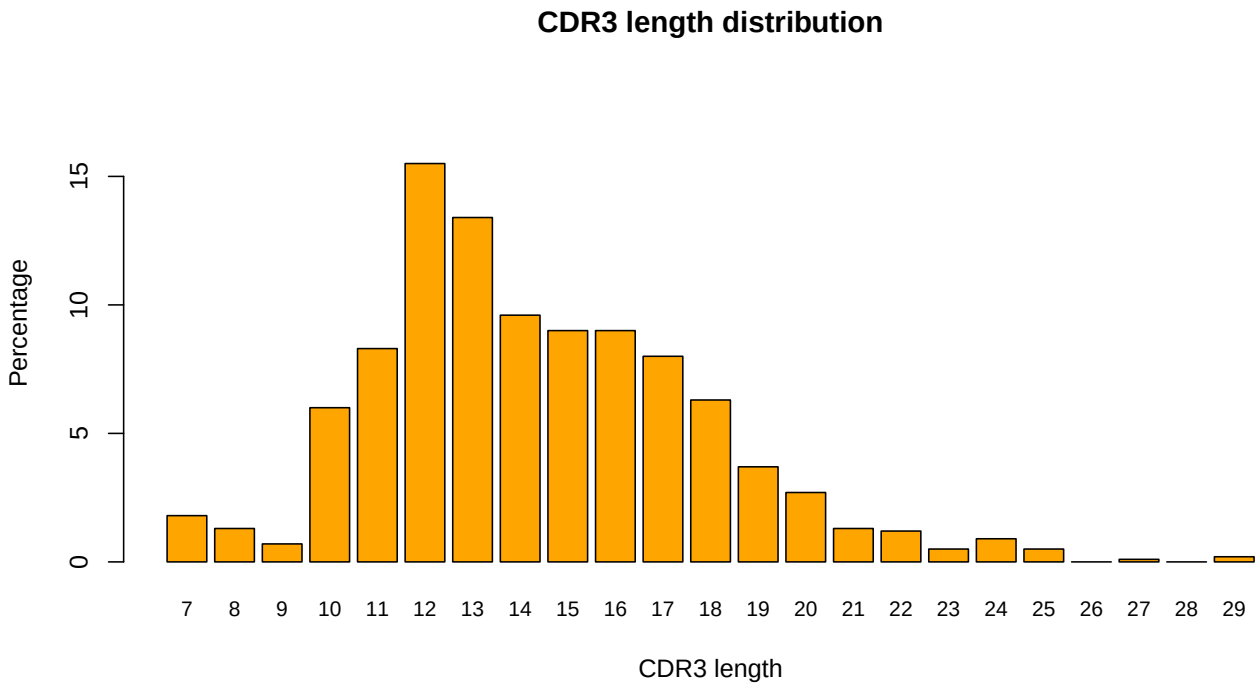
```
# Example:
data(clones.ind)
CDR3length<-clones.CDR3Length(CDR3Length = clones.ind$CDR3_length_AA,
                  functionality = clones.ind$Functionality_all_sequences)

# output of some CDR3 length proportions (CDR3length$CDR3_length[1:3]):
```

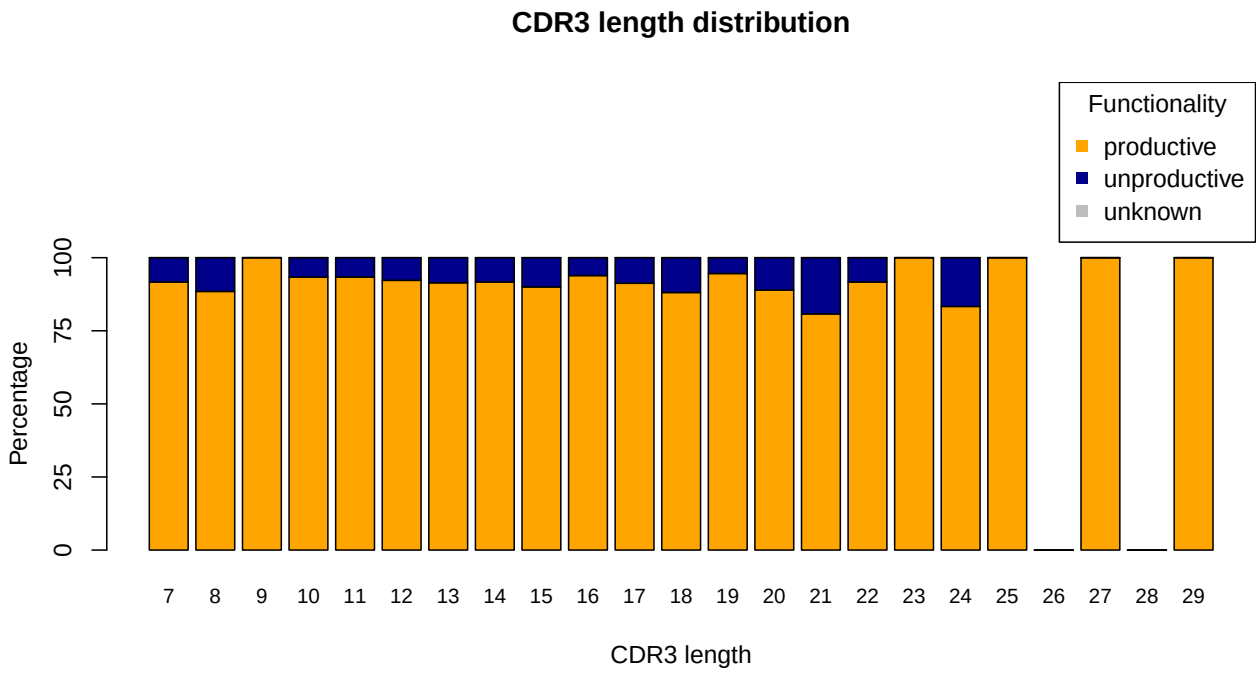
CDR3_length_7	CDR3_length_8	CDR3_length_9
0.018	0.013	0.007

```
# output of functionality ratios for some CDR3 lengths
    # (CDR3length$CDR3_length_vs_functionality[,1:3)])
```

	CDR3_length_7	CDR3_length_8	CDR3_length_9
productive	0.8889	0.8462	1
unproductive	0.1111	0.1538	0
unknown	0	0	0

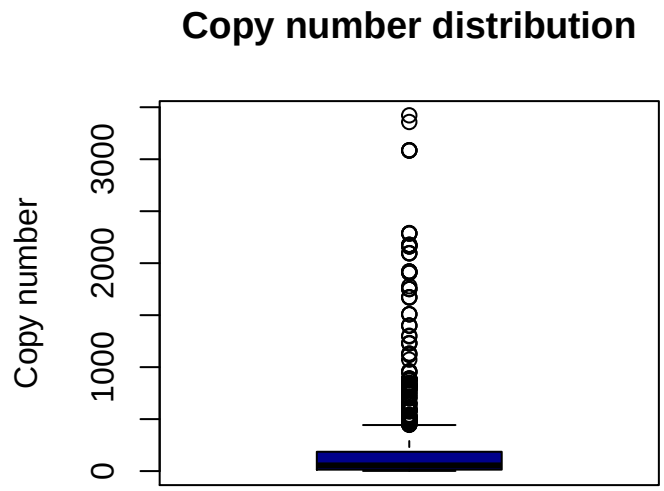
```
plotClonesCDR3Length(CDR3Length = clones.ind$CDR3_length_AA,
                     functionality = clones.ind$Functionality_all_sequences,
                     title = "CDR3 length distribution", PDF = NULL)
```



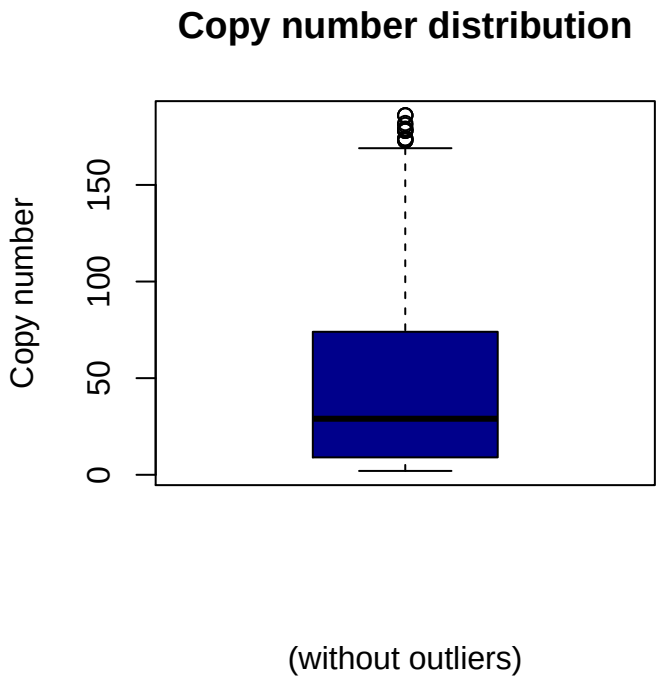


Clone copy number distribution `plotClonesCopyNumber()` can be used to plot clone sizes as an boxplot.

```
# Example: Clone copy number distribution with and without outliers
plotClonesCopyNumber(copyNumber = clones.ind$total_number_of_sequences,
                     withOutliers=TRUE, color = "darkblue",
                     title = "Copy number distribution", PDF = NULL)
```



```
plotClonesCopyNumber(copyNumber = clones.ind$total_number_of_sequences,
                     withOutliers=FALSE, color = "darkblue",
                     title = "Copy number distribution", PDF = NULL)
```



Further functions like `geneUsage()`or `trueDiversity()` can be used to further analysis of clone sets.

Comparison of different samples

If more than one sample should be analyzed, bcRep provides some function, to compare different samples. Gene usage, amino acid distribution, diversity and shared clones can be compared and visualized.

Comparison of gene usage

Gene usage can be compared using `compare.geneUsage()`. Input is a list including sequence vectors of each individual. If no names for the samples are specified, samples will be called Sample1, Sample2, ... in the input order. Analysis of subgroup (f.e. IGHV1), gene (f.e. IGHV1-1) or allele (f.e. IGHV1-1*2), as well as of relative or absolute values is possible.

Results can be visualized using `plotCompareGeneUsage()` (optional be saved as PDF in working directory). A heatmap will be returned, with samples as rows and genes as columns. Proportions of genes are color coded.

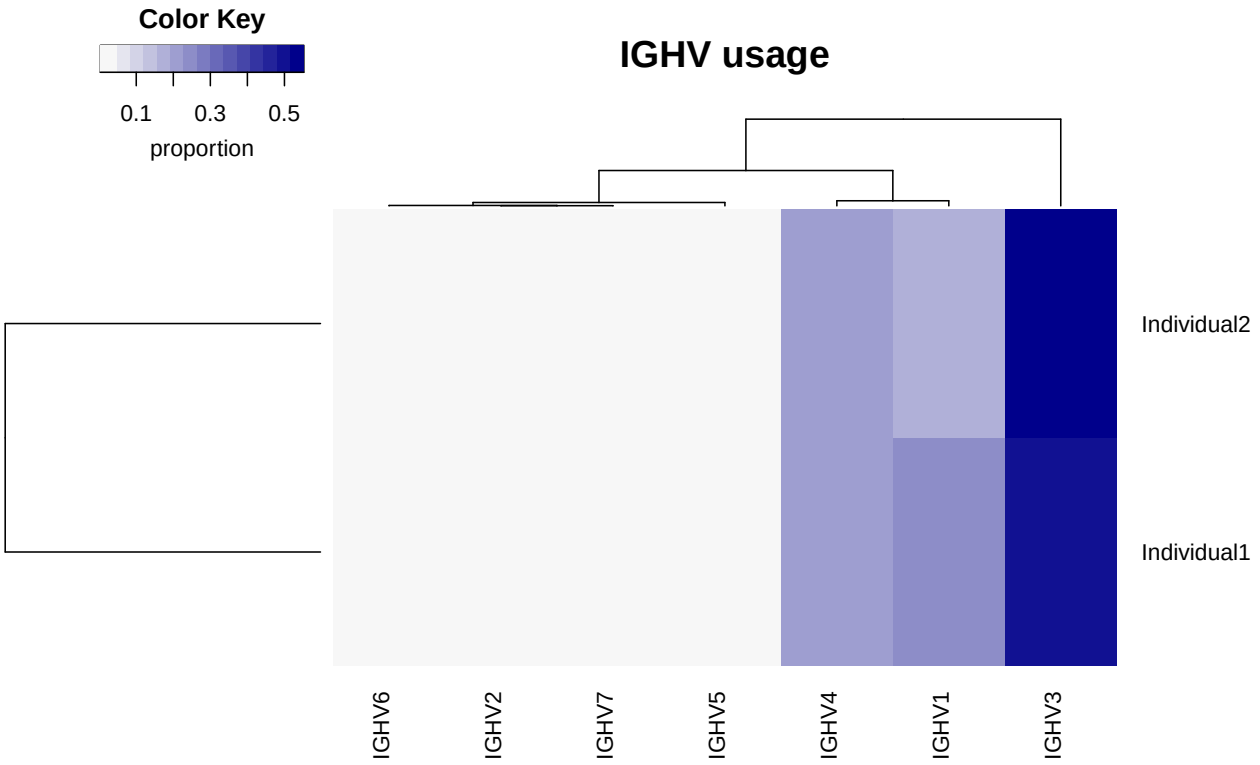
```
data(aaseqtab)
data(aaseqtab2)
V.comp<-compare.geneUsage(gene.list = list(aaseqtab$V_GENE_and_allele,
                                           aaseqtab2$V_GENE_and_allele),
                          level = "subgroup", abundance = "relative",
                          names = c("Individual1", "Individual2"),
                          nrCores = 1)
```

	IGHV1	IGHV2	IGHV3	IGHV4	IGHV5
Individual1	0.2477	0.004005	0.507	0.2096	0.02236
Individual2	0.1863	0.007667	0.5523	0.2077	0.03

Table 11: Table continues below

	IGHV6	IGHV7
Individual1	0.006008	0.003338
Individual2	0.007	0.009

```
plotCompareGeneUsage(comp.tab = V.comp, color = c("gray97", "darkblue"), PDF = NULL)
```



Comparison of amino acid distribution

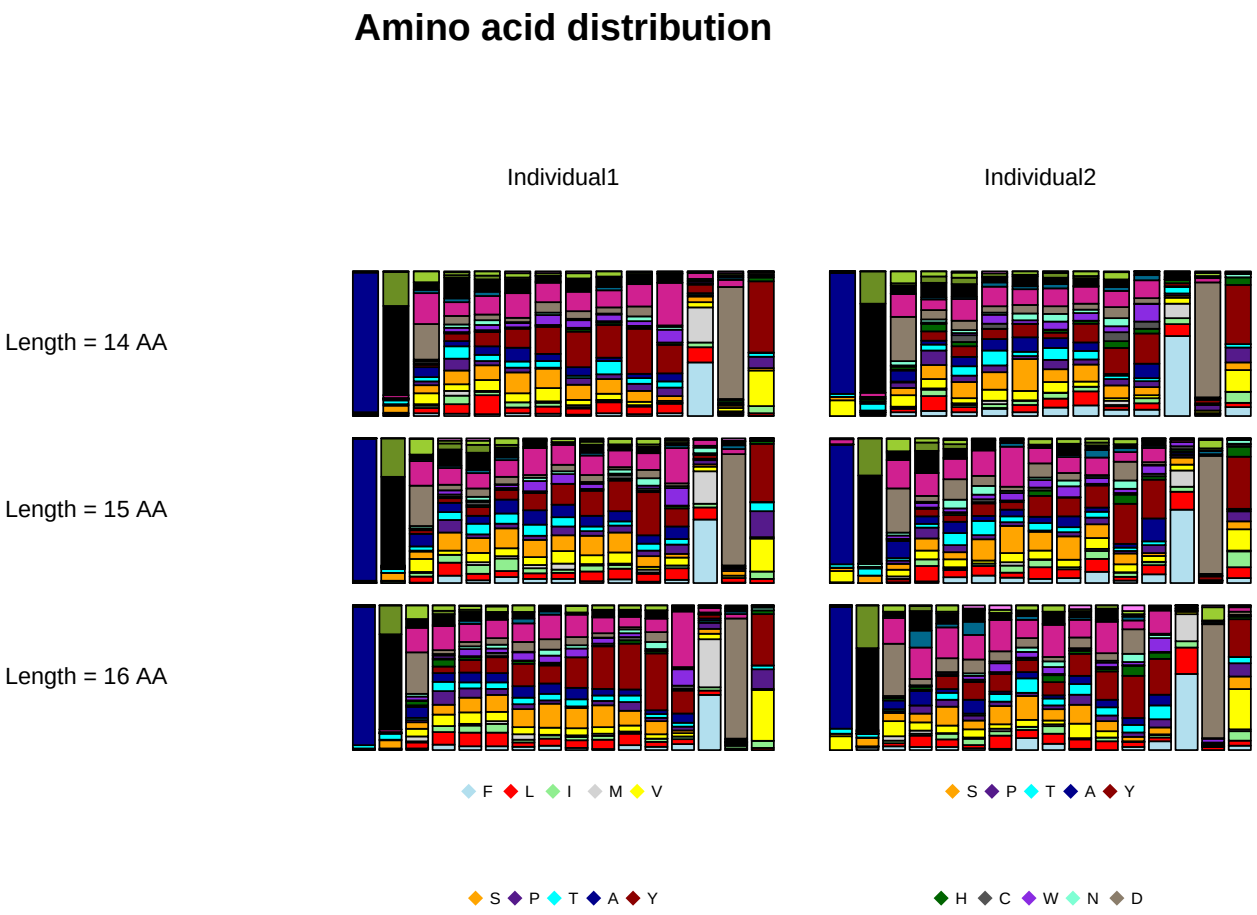
The amino acid distribution between two or more individuals can be compared using `compare.aaDistribution()`. Input is again a list including sequence vectors for each individual. Sequences of the same length will ne analyzed together. Optional the number of sequences used for analysis can be returned, as well. If no names for the samples are specified, samples will be called Sample1, Sample2, ... in the input order.

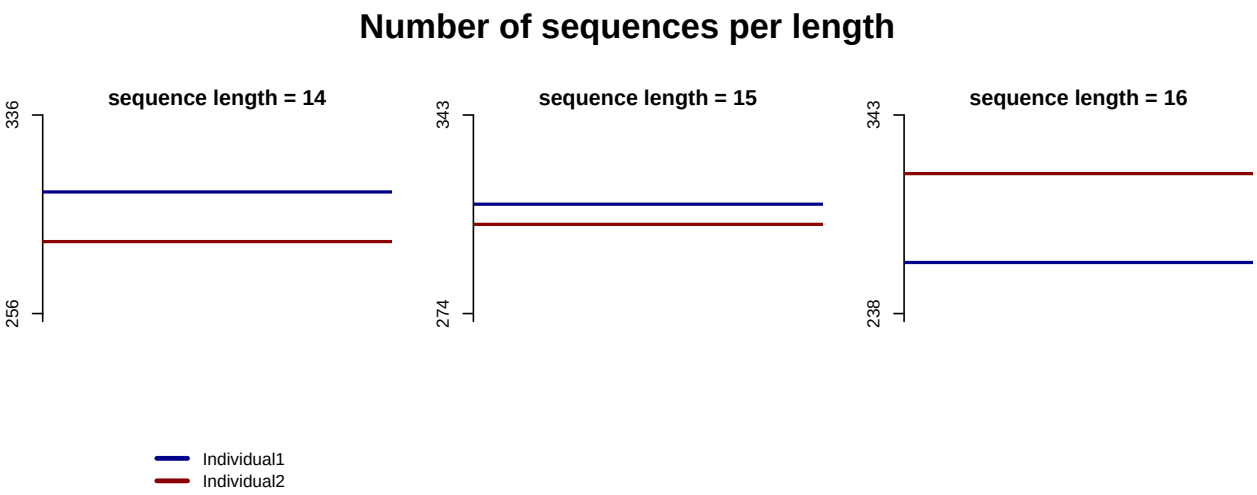
The output is a list, containing amino acid distributions for each sequence length and each individual.

`plotCompareAADistribution()` returns a plot (optional be saved as PDF in working directory), where

columns represent the samples, and rows represent different sequence length. In each field, one bar represents one position of the sequence.

```
data(aaseqtab)
data(aaseqtab2)
AAdistr.comp<-compare.aaDistribution(sequence.list = list(aaseqtab$CDR3_IMGT,
                                                          aaseqtab2$CDR3_IMGT),
                                     names = c("Individual1", "Individual2"),
                                     numberSeq = TRUE, nrCores = 1)

# Comparison of sequence length of 14-16 amino acids:
## Individual 1: grep("14|15|16",names(AAdistr.comp$Amino_acid_distribution$Individual1))
        ## --> 13:15
## Individual 2: grep("14|15|16",names(AAdistr.comp$Amino_acid_distribution$Individual2))
        ## --> 12:14
AAdistr.comp.part<-list(list(AAdistr.comp$Amino_acid_distribution$Individual1[13:15],
                             AAdistr.comp$Amino_acid_distribution$Individual2[12:14]),
                        list(AAdistr.comp$Number_of_sequences_per_length$Individual1[13:15],
                             AAdistr.comp$Number_of_sequences_per_length$Individual2[12:14]))
names(AAdistr.comp.part)<-names(AAdistr.comp)
names(AAdistr.comp.part$Amino_acid_distribution)<-
  names(AAdistr.comp$Amino_acid_distribution)
names(AAdistr.comp.part$Number_of_sequences_per_length)<-
  names(AAdistr.comp$Number_of_sequences_per_length)

plotCompareAADistribution(comp.tab = AAdistr.comp.part, plotSeqN = TRUE,
                          colors=c("darkblue","darkred"), PDF = NULL)
```





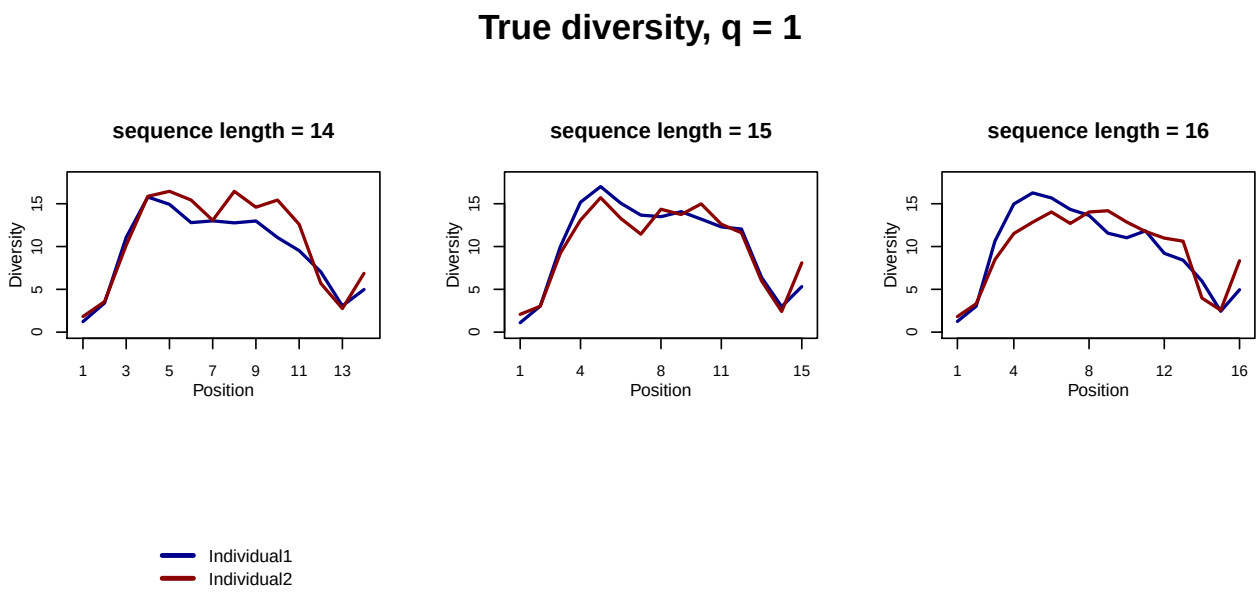
Comparison of richness and diversity

Richness and diversity can be compared using `compare.trueDiversity()`. Input can be a list of sequences or the output of `compare.aaDistribution()`. Diversity can be analyzed for order 0 (richness), 1 (exponent of Shannon entropy) or 2 (inverse Simpson) (see diversity analysis above). If no names for the samples are specified, samples will be called Sample1, Sample2, ... in the input order.

The output is a list, containing richness or diversity indices for each sequence length and each individual.

Results can be visualized using `plotCompareTrueDiversity()` (optional be saved as PDF in working directory). A plot will be returned, with one figure for each sequence length. Each figure contains the sequence position on the x-axis and the richness/diversity index on the y-axis. If no colors are specified, `rainbow()` will be used.

```
data(aaseqtab)
data(aaseqtab2)
trueDiv.comp<-compare.trueDiversity(sequence.list = list(aaseqtab$CDR3_IMGT,
                                                         aaseqtab2$CDR3_IMGT),
                                    names = c("Individual1", "Individual2"),
                                    order = 1, nrCores = 1)

# Comparison of sequence length of 14-16 amino acids:
grepindex1<-grep("14|15|16",names(trueDiv.comp$Individual1))
grepindex2<-grep("14|15|16",names(trueDiv.comp$Individual2))
trueDiv.comp.part<-list(trueDiv.comp$True_diversity_order,
                        trueDiv.comp$Individual1[grepindex1],
                        trueDiv.comp$Individual2[grepindex2])
names(trueDiv.comp.part)<-names(trueDiv.comp)
plotCompareTrueDiversity(comp.tab = trueDiv.comp.part, colors=c("darkblue","darkred"),
                         PDF = NULL)
```



Looking for clones, that are shared between several samples

To compare clones of different samples, the function `clones.shared()` can be used. The criteria are the same than in function `clones()`:

- same CDR3 (amino acid) length and a identity of a given treshold (so it's possible, to look for same CDR3 sequences or for a CDR3 identity of 85%)
- same V gene
- same J gene (optional)

The input for the function has to be prepared as following: combine all individual clone files to one using `rbind()` and add a column in front of all other columns with sample ID's. An example, how the table should look like, you can find in `data(clones.allind)`.

```
# Example:
data(clones.allind) # includes 300 clones for 8 individuals (C6-9, P1-3/10)
dim(clones.allind) # dimensions of clones.allind [rows, columns]
## [1] 2400   15

  # Some lines and columns from clones.allind:
    # data.frame(rbind(clones.allind[1:3,1:3],
                # clones.allind[(nrow(clones.allind)-2):nrow(clones.allind),1:3]),
                # row.names=NULL)
```

individuals	unique_CDR3_sequences_AA	CDR3_length_AA
C6	ARGSGAY, ARGSGEY	7
C6	ARDFADY, ARYFADY	7
C6	ARDRGLDY, ARDRQLDY, ARDRSLDY	8
P3	AKSARPFDY, AKSARPVDY	9
P3	ARETMVYFDY, ARETMVYLDY	10
P3	ASLADDESVY, ASLTDDESVY	10

Output of the table is the same than for `clones()`. Individual data is seperated by ";".

```
# Example:
data(clones.allind)
sharedclones<-clones.shared(clones.tab = clones.allind, identity = 0.85, useJ = TRUE)

  # Some columns of sharedclones
```

number_individuals	individuals	CDR3_length_AA
2	C9; P1	8

Table 14: Table continues below

shared_CDR3	number_shared_CDR3
ARGLPFDY; ARGLSFDY	2

Table 15: Table continues below

CDR3_sequences_per_individual	sequence_count_per_CDR3	V_gene
ARGLSFVY, ARGLSFDY; ARGLPFDY, ARGLPIDY	1, 16; 367, 1	IGHV4-34

The table including shared clones can be summarized using `clones.shared.summary()`. This function returns a data frame, which contains the number of individual clones (optional, only if clone table is given) and the number of shared clones. The number of individual clones is equivalent to the total number of clones per individual minus the number of shared clones.

```
# Example:
data(clones.allind) # includes 300 clones for 8 individuals (C6-9, P1-3/10)
sharedclones<-clones.shared(clones.tab = clones.allind, identity = 0.85, useJ = TRUE)
sharedclones.summary<-clones.shared.summary(shared.tab = sharedclones,
                                            clones.tab = clones.allind)
```

group	number_clones
only in C6	300
only in C7	300
only in C8	300
only in C9	299
only in P10	300
only in P1	299
only in P2	300
only in P3	300
C9; P1	1