Package ‘tigger’

April 1, 2015

Title R tools for inferring new IGHV alleles from Rep-Seq data

Version 2.1

Author Daniel Gadala-Maria and Jason Anthony Vander Heiden

Maintainer Daniel Gadala-Maria <`daniel.gadala-maria@yale.edu`>

Description Infers the IGHV genotype of an individual from Rep-Seq data, including any novel alleles, and uses this information to correct existing IGHV allele calls from among the sample sequences. Described in:
Gadala-Maria et al. (2015)
Automated analysis of high-throughput B cell sequencing data reveals a high frequency of novel immunoglobulin V gene segment alleles. PNAS. 112(8):E862-70

License CC BY-NC-SA 3.0

LazyData true

Depends R (>= 3.0.0),
dplyr

Suggests knitr

VignetteBuilder knitr

R topics documented:

`assignAlleleGroups` *Find indicies of allele calls*

Description

`assignAlleleGroups` determines the locations of unique alleles within a mixed group.

Usage

```
assignAlleleGroups(allele_calls, allele_min = 1e-04, binomial_cutoff = TRUE,
  alpha = 0.05)
```

Arguments

`allele_calls`	`character` vector respresenting Ig allele calls. Calls may consist of multiple comma-separated alleles.
`allele_min`	`numeric` indicating the minimum fraction of `allele_calls` that must contain an allele for it to be retained. Integers of 1 or greater are interprted as a minumum sequence count.
`binomial_cutoff`	`logical` indicating if an `allele_min` cutoff < 1 should be applied in a binomial manner.
`alpha`	`numeric` indicating the alpha cutoff used when applying a binomial cutoff of `allele_min`.

Value

`list` of indicies in `allele_calls` where each unique input allele can be found.

Examples

```
# Create a sample vector of allele calls
allele_names = c("IGHV1-69D*01","IGHV1-69*01","IGHV1-2*01","IGHV1-69-2*01",
"IGHV2-5*01","IGHV1-NL1*01","IGHV1-2*02", "IGHV1-69*02")
allele_counts = c(24, 15, 26, 36, 15, 43, 2, 42)
alleles = rep(allele_names, allele_counts)

# Find how many of each allele there are
assignAlleleGroups(alleles)
```

`compareGenotypes` *Compare two genotypes*

Description

`compareGenotypes` takes two genotypes, binds them together by gene, and adds columns indicating the alleles only in the first, the alleles only in the second, and the alleles shared between the two.

Usage

```
compareGenotypes(genotype1, genotype2)
```

Arguments

`genotype1`	a genotype of the type returned by `inferGenotype`
`genotype2`	a genotype of the type returned by `inferGenotype`

Value

A data frame indicating which alleles are unique to each genotype or shared between then two

See Also

`inferGenotype`

Examples

```
# Load example data
data(sample_db)

# Determine a genotype
geno = geno2 = inferGenotype(sample_db[,"V_CALL"])
# Shuffle the gene names to make a different "genotype"
geno2$gene = sample(geno2$gene)

# Compare the two genotypes
compareGenotypes(geno, geno2)
```

`compareSepString` *Compare two strings of separated values*

Description

`compareSepString` takes two strings, usually comma-separated, and returns their intersection or difference in the form of a string using the same separator.

Usage

```
compareSepString(string1, string2, value = "both", sep = ",")
```

Arguments

`string1`	a string of separated values, usually by a comma
`string2`	a second string of separated values, usually by a comma
`value`	what values to return. If "both" the intersection of the values will be returned. "only1" and "only2" will return, respectively, the values only in the first string or only in the second string.
`sep`	the separator that should be used to divide up the strings before comparing the values they hold

Value

A string of values representing the intersection or difference of of the input strings, separated in the same manner as the input

Examples

```
compareSepString("1,2,5,6", "1,5,7", value="both")
compareSepString("1,2,5,6", "1,5,7", value="only1")
compareSepString("1,2,5,6", "1,5,7", value="only2")
```

`createGermlines` *Create sequences with each combination of polymorphisms*

Description

`createGermlines` inserts nucleotides in the desired locations of a provided sequence, for each combination of possible insertions.

Usage

```
createGermlines(germline, positions, nucleotides)
```

Arguments

`positions`	a vector of positions which to be changed
`nucleotides`	a vector of nucletides to which to change the positions
`sequence`	the starting nucletide sequence

Details

The nucletoide sequence provided should be named, and will serve as the basis of the resulting names. For example, a sequence named IGHV1-2*02 with position 163 mutated to a C (from a T) will be named 1-2*01_T163C.

Value

Each combination of sequences with the desired nucleotides in provided locations, with names indicating the insertion(s) in the form _[germline_nucleotide][position][inserted_nucleotide].

See Also

insertPolymorphisms

Examples

```
# Insert each combination of letters at the listed locations
# Note that 3! is 6
createGermlines("hugged", c(1,2,6), c("t","i","r"))
```

---

detectNovelV *Find novel alleles from repertoire sequencing data*

Description

`detectNovelV` analyzes mutation patterns in sequences thought to align to each germline allele in order to determine which positions might be polymorphic.

Usage

```
detectNovelV(v_sequences, j_genes, junc_lengths, allele_groups, germline_db,
  y_intercept = 1/8, nt_min = 1, nt_max = 312, mut_min = 1,
  mut_max = 10, j_max = 0.1, min_seqs = 50, min_frac = 1/8,
  verbose = FALSE, quiet = T)
```

Arguments

`v_sequences`	a vector of IMGT-gapped sample V sequences
`j_genes`	a vector of J gene names utilized by the samples
`junc_lengths`	a vector of the junction lengths of the sample
`allele_groups`	a list whose names match the alle names in `germline_db` and the contents of which are the indicies of `v_sequences` that are assigned to those alleles. See `assignAlleleGroups`.
`germline_db`	a vector of named nucleotide germline sequences matching the calls detailed in `allele_groups`
`y_intercept`	the y-intercept above which positions should be considered potentially polymorphic, as utilized by `findIntercepts`
`nt_min`	the first nucleotide position to be considered, as utilized by `trimMutMatrix`
`nt_max`	the last nucleotide position to be considered, as utilized by `trimMutMatrix`
`mut_min`	the minimum number of sequence-wide mutations for sequences that will be used in analysis, as utilized by `trimMutMatrix`
`mut_max`	the maximum number of sequence-wide mutations for sequences that will be used in analysis, as utilized by `trimMutMatrix`
`j_max`	the maximum fraction of sequences perfectly aligning to a potential novel allele that are allowed to utilize to a particular combination of junction length and J gene
`min_seqs`	the minimum number of total sequences (within the desired mutational range and nucleotide range) required for the samples to be considered, as utilized by `trimMutMatrix`

`min_frac`	the minimum fraction of sequences that must have usable nucleotides in a given position for that position to considered, as utilized by `trimMutMatrix`
`verbose`	if `TRUE`, a message will be printed when the samples do not meet the required parameters

Details

`detectNovelV` applies `findNovelAlleles` to each allele call found in the names of `allele_groups`. Mutations are determined through comparison to the provided germline and the mutation frequency at each *position* is determined as a function of *sequence-wide* mutation counts. Polymorphic positions are expected to exhibit a high mutation frequency despite sequence-wide mutation count. False positive of potential novel alleles resulting from clonally-related sequences are guarded against by ensuring that sequences perfectly matching the potential novel allele utilize a wide range of combinations of J gene and junction length.

Value

for each potential novel allele, a list of length five is returned containing (1) the (named) germline sequence, (2) the y-intercepts of the position(s) which passed the y-intercept threshhold (with names indicating the positions themselves), (3) a matrix containing the fraction of sequences mutated at each nucleotide position (columns) as a function of sequence-wide mutation count (rows), (4) table(s) indicating the nucletoide usage at each polymorphic position as a function of mutation count, and (5) a table detailing the number of unmutated versions of the novel allele found to use each combination of J gene (columns) and junction length (rows).

See Also

`findNovelAlleles`, `findIntercepts`, `summarizeMutations`, `trimMutMatrix`

Examples

```
# Load example data and germlines
data(sample_db)
data(germline_ighv)

# Single out calls utilizing particular germline sequences and extract info
germs = germline_ighv[c(1,41)]
matches = lapply(names(germs),grep, sample_db$V_CALL, fixed=TRUE)
names(matches) = names(germs)
samples = sample_db$SEQUENCE_IMGT
j_genes = getGene(sample_db$J_CALL)
junc_lengths = sample_db$JUNCTION_LENGTH

# Find novel alleles and return relevant data
novel = detectNovelV(samples, j_genes, junc_lengths, matches, germline_ighv)
# Print the nucleotide sequence of the first (if any) novel alleles found
novel[[1]][[1]]
```

`findIntercepts` *Find which y-intercepts are above a threshhold*

Description

`findIntercepts` takes a matrix as returned by `trimMutMatrix`, where mutation frequencies at given positions (rows) are calculated as a function of sequence-wide mutation count (columns), and determines which y-intercepts (positionalal mutation frequency when sequence-wide mutation counts equals zero) are predicted to be above a certain threshhold.

Usage

```
findIntercepts(mut_fracs, y_intercept = 1/8, alpha = 0.05)
```

Arguments

`mut_fracs`	a matrix as returned by `trimMutMatrix`, where rows are named with nucleotide position and columns are named with sequence-wide mutation count
`y_intercept`	the y-intercept above which positions should be returned
`alpha`	the alpha cutoff that should be used in calculating the confidence interval for the y-intercept

Details

The y-intercept is tested by using a t-test to determine the range of values likely to contain the intercept. If the lower-bound of this confidence interval is greater than the y-intercept cutoff, the position will be returned. This is the key step in `detectNovelV`.

Value

A matrix of mutation frequencies at given positions (rows) as a function of sequence-wide mutation count (columns) for the desired ranges of each. If there is a problem with the number of sequences, etc., `NULL` will be returned.

See Also

`trimMutMatrix`, `detectNovelV`

Examples

```
# Invent some mutation matrix
mut_fracs = matrix(c(sapply(1:10, function(x) max(rnorm(1, x),0)/20),
                     sapply(1:10, function(x) max(rnorm(1, x),0)/20),
                     sapply(1:10, function(x) max(rnorm(1, x),0)/20),
                     sapply(rep(5,10), function(x) max(rnorm(1, x),0)/10)),
                   nrow=4, byrow=T)
colnames(mut_fracs) = 1:10; rownames(mut_fracs) = 1:4
# Test to see if any have a y-intercept above 1/8 (position 4 should)
findIntercepts(mut_fracs)
```

`findNovelAlleles` *Find novel alleles in sequences thought to utilize one particular allele*

Description

`findNovelAlleles` analyzes mutation patterns in sequences thought to align to a particular germline allele in order to determine which positions might be polymorphic.

Usage

```
findNovelAlleles(samples, germlines, j_genes, junc_lengths, y_intercept = 1/8,
  nt_min = 1, nt_max = 312, mut_min = 1, mut_max = 10, j_max = 0.1,
  min_seqs = 50, min_frac = 3/4, verbose = FALSE)
```

Arguments

`samples`	a vector of IMGT-gappedsample V sequences thought to be utilizing the same germline V allele
`j_genes`	a vector of J gene names utilized by the samples
`junc_lengths`	a vector of the junction lengths of the sample
`y_intercept`	the y-intercept above which positions should be considered potentially polymorphic, as utilized by `findIntercepts`
`nt_min`	the first nucleotide position to be considered, as utilized by `trimMutMatrix`
`nt_max`	the last nucleotide position to be considered, as utilized by `trimMutMatrix`
`mut_min`	the minimum number of sequence-wide mutations for sequences that will be used in analysis, as utilized by `trimMutMatrix`
`mut_max`	the maximum number of sequence-wide mutations for sequences that will be used in analysis, as utilized by `trimMutMatrix`
`j_max`	the maximum fraction of sequences perfectly aligning to a potential novel allele that are allowed to utilize to a particular combination of junction length and J gene
`min_seqs`	the minimum number of total sequences (within the desired mutational range and nucleotide range) required for the samples to be considered, as utilized by `trimMutMatrix`
`min_frac`	the minimum fraction of sequences that must have usable nucleotides in a given position for that position to considered, as utilized by `trimMutMatrix`
`verbose`	if `TRUE`, a message will be printed when the samples do not meet the required parameters
`germline`	the germline V sequence utilized by the samples

Details

Mutations are determined through comparison to the provided germline and the mutation frequency at each *position* is determined as a function of *sequence-wide* mutation counts. Polymorphic positions are expected to exhibit a high mutation frequency despite sequence-wide mutation count. False positive of potential novel alleles resulting from clonally- related sequences are guarded against by ensuring that sequences perfectly matching the potential novel allele utilize a wide range of combinations of J gene and junction length.

Value

For each potential novel allele, a list of length five is returned containing (1) the (named) germline sequence, (2) the y-intercepts of the position(s) which passed the y-intercept threshhold (with names indicating the positions themselves), (3) a matrix containing the fraction of sequences mutated at each nucleotide position (columns) as a function of sequence-wide mutation count (rows), (4) table(s) indicating the nucleotoide usage at each polymorphic position as a function of mutation count, and (5) a table detailing the number of unmutated versions of the novel allele found to use each combination of J gene (columns) and junction length (rows).

See Also

`findIntercepts`, `summarizeMutations`, `trimMutMatrix`

Examples

```
# Load example data and germlines
data(sample_db)
data(germline_ighv)

# Single out calls utilizing a particular germline sequence and extract info
germ = germline_ighv[41]
matches = grep(names(germ), sample_db$V_CALL, fixed=TRUE)
samples = sample_db$SEQUENCE_IMGT[matches]
j_genes = getGene(sample_db$J_CALL[matches])
junc_lengths = sample_db$JUNCTION_LENGTH[matches]

# Find novel alleles and return relevant data
novel = findNovelAlleles(samples, germ, j_genes, junc_lengths)
# Print the nucleotide sequence of the first (if any) novel alleles found
novel[[1]][[1]]
```

`findNucleotideUsage` *Determine nucleotide usage at a given position*

Description

`findNucleotideUsage` determines the nucleotide distribution at a given IMGT-numbered position as a function of sequence-wide mutation counts.

Usage

```
findNucletoideUsage(position, samples, germline, mut_counts, mut_min = 1,
  mut_max = 10)
```

Arguments

`position`	an integer representing the IMGT-numbered position of interest
`samples`	a vector of sample nucleotide sequences
`germline`	a string with the germline nucleotide sequence
`mut_counts`	a vector containing the mutation count of each sample

`mut_min`	the minimum number of sequence-wide mutations for sequences that will be included in the returned matrix
`mut_max`	the maximum number of sequence-wide mutations for sequences that will be included in the returned matrix

Value

A table of nucleotide usage at a given position as a function of sequence-wide mutation counts, with the germline base on top and the most frequent mutated-to base on the bottom

Examples

```
# Load example data and germlines
data(sample_db)
data(germline_ighv)

# Single out calls utilizing a particular germline sequence
germ = germline_ighv[1]
matches = grep(names(germ), sample_db$V_CALL, fixed=TRUE)
samples = sample_db$SEQUENCE_IMGT[matches]

# Find mutation counts in those sequences
mut_counts = getMutCount(samples, rep(names(germ), length(samples)), germ)

# Find nucleotide usage at position 2 as a function of mutation count
findNucletoideUsage(2, samples, germ, mut_counts)
```

`findUnmutatedCalls` *Determine which calls represent an unmutated allele*

Description

`findUnmutatedCalls` determines which allele calls would represent a perfect match with the germline sequence, given a vector of allele calls and mutation counts. In the case of multiple alleles being assigned to a sequence, only the subset that would represent a perfect match is returned.

Usage

```
findUnmutatedCalls(allele_calls, mut_counts, only_unmutated = TRUE)
```

Arguments

`allele_calls`	a vector of strings respresenting Ig allele calls, where multiple calls are separated by a comma
`mut_counts`	a list containing distance to each germline allele call within `allele_calls`, as returned by `getMutCount`
`only_unmutated`	if `TRUE`, calls where no allele that would represent an unmutated sequence will be omitted from the output

Value

A vector of strings containing the members of `allele_calls` that represent unmutated sequences

Examples

```
# Load germline database
data(germline_ighv)

# Use createGermlines to insert a mutation into a germline sequence
sample_seqs = c(germline_ighv[2],
                createGermlines(germline_ighv[1], 103, "G"),
                germline_ighv[1],
                germline_ighv[2])

# Pretend that one sample sequence has received an ambiguous allele call
sample_alleles = c(paste(names(germline_ighv[1:2]), collapse=","),
                  names(germline_ighv[2]),
                  names(germline_ighv[1]),
                  names(germline_ighv[2]))

# Compare the sequence to a subset of the germlines
mut_counts = getMutCount(sample_seqs, sample_alleles, germline_ighv)

# Find which of the sample alleles are unmutated
findUnmutatedCalls(sample_alleles, mut_counts)
```

`genotypeFasta` *Return the nucleotide sequences of a genotype*

Description

`genotypeFasta` converts a genotype table into a vector of nucleotide sequences.

Usage

```
genotypeFasta(genotype, germline_db)
```

Arguments

`genotype`	a table of alleles denoting a genotype, as returned by `inferGenotype`
`germline_db`	a vector of named nucleotide germline sequences matching the alleles detailed in `genotype`

Value

A named vector of strings containing the germline nucleotide sequences of the alleles in the provided genotype

See Also

`inferGenotype`

Examples

```
# Load example data
data(germline_ighv)
data(sample_db)

# Infer and view a genotype from the sample
geno = inferGenotype(updateAlleleNames(sample_db[,"V_CALL"]))
geno

# Return the sequences that correspond to the genotype
genotypeFasta(geno, germline_ighv)
```

`getMutatedPositions` *Find the location of mutations in a sequence*

Description

`getMutatedPositions` takes two vectors of aligned sequences and compares pairs of sequences. It returns a list of the nucleotide positions of any differences.

Usage

```
getMutatedPositions(samples, germlines, ignored_regex = "[\\.N-]",
  match_instead = FALSE)
```

Arguments

`samples`	a vector of strings respresenting aligned sequences
`germlines`	a vector of strings respresenting aligned sequences to which `samples` will be compared. If only one string is submitted, it will be used for all `samples`.
`ignored_regex`	a regular expression indicating what characters should be ignored (such as gaps and N nucleotides).
`match_instead`	if `TRUE`, the function returns the positions that are the same instead of those that are different.

Value

A list of the nucleotide positions of any differences between the input vectors.

Examples

```
# Create strings to act as a sample sequences and a reference sequence
seqs = c("----GATA","GAGAGAGA","TANA")
ref = "GATAGATA"

# Find the differences between the two
getMutatedPositions(seqs, ref)
```

`getMutCount` *Determine the mutation counts from allele calls*

Description

`getMutCount` takes a set of nucleotide sequences and their allele calls and determines the distance between that seqeunce and any germline alleles contained within the call

Usage

```
getMutCount(samples, allele_calls, germline_db)
```

Arguments

`samples`	a vector of IMGT-gapped sample V sequences
`allele_calls`	a vector of strings respresenting Ig allele calls for the sequences in `samples`, where multiple calls are separated by a comma
`germline_db`	a vector of named nucleotide germline sequences matching the calls detailed in `allele_calls`

Value

A list equal in length to `samples`, containing the Hamming distance to each germline allele contained within each call within each element of `samples`

Examples

```
# Load germline database
data(germline_ighv)

# Use createGermlines to insert a mutation into a germline sequence
sample_seqs = c(germline_ighv[2],
                createGermlines(germline_ighv[1], 103, "G"),
                createGermlines(germline_ighv[1], 107, "C"))

# Pretend that one sample sequence has received an ambiguous allele call
sample_alleles = c(paste(names(germline_ighv[1:2]), collapse=","),
                  names(germline_ighv[2]),
                  names(germline_ighv[1]))

# Compare each sequence to its assigned germline(s) to determine the distance
getMutCount(sample_seqs, sample_alleles, germline_ighv)
```

---

getSegment *Get Ig segment allele, gene and family names*

Description

`getSegment` performs generic matching of delimited segment calls with a custom regular expression. `getAllele`, `getGene` and `getFamily` extract the allele, gene and family names, respectively, from a character vector of immunoglobulin (Ig) segment allele calls in IMGT format.

Usage

```
getSegment(segment_call, segment_regex, first = TRUE, collapse = TRUE,
  sep = ",")

getAllele(segment_call, first = TRUE, collapse = TRUE, sep = ",")

getGene(segment_call, first = TRUE, collapse = TRUE, sep = ",")

getFamily(segment_call, first = TRUE, collapse = TRUE, sep = ",")
```

Arguments

`segment_call`	character vector containing segment calls delimited by commas.
`segment_regex`	string defining the segment match regular expression.
`first`	if `TRUE` return only the first call in `segment_call`; if `FALSE` return all calls delimited by commas.
`collapse`	if `TRUE` check for duplicates and return only unique segment assignments; if `FALSE` return all assignments (faster). Has no effect if `first=TRUE`.
`sep`	character defining both the input and output segment call delimiter.

Value

A character vector containing allele, gene or family names

References

`http://imgt.org`

See Also

Uses `str_extract`.

Examples

```
kappa_call <- c("Homsap IGKV1-39*01 F,Homsap IGKV1D-39*01 F", "Homsap IGKJ5*01 F")

getAllele(kappa_call)
getAllele(kappa_call, first=FALSE)

getGene(kappa_call)
getGene(kappa_call, first=FALSE)
```

```
getFamily(kappa_call)
getFamily(kappa_call, first=FALSE)
getFamily(kappa_call, first=FALSE, collapse=TRUE)
```

`inferGenotype` *Infer a subject-specific genotype*

Description

`inferGenotype` infers an subject's genotype by finding the minimum number set of alleles that can explain the majority of each gene's calls. The most common allele of each gene is included in the genotype first, and the next most common allele is added until the desired fraction of alleles can be explained. In this way, mistaken allele calls (resulting from sequences which by chance have been mutated to look like another allele) can be removed.

Usage

```
inferGenotype(allele_calls, fraction_to_explain = 7/8, gene_cutoff = 0.001)
```

Arguments

`allele_calls`	a vector of strings respresenting Ig allele calls of unmutated sequences from a single subject
`fraction_to_explain`	the portion of each gene that must be explained by the alleles that will be included in the genotype
`gene_cutoff`	either a number of sequences or a fraction of the length of `allele_calls` denoting the minimum number of times a gene must be observed in `allele_calls` to be included in the genotype

Details

Allele calls representing cases where multiple alleles have been assigned to a single sample sequence are rare among unmutated sequences but may result if nucleotides for certain positions are not available. Calls containing multiple alleles are treated as belonging to all groups until one of those groups is included in the genotype.

Value

A table of alleles denoting the genotype of the subject

Note

This method works best with data derived from blood, where a large portion of sequences are expected to be unmutated. Ideally, there should be hundreds of allele calls per gene in the input.

Examples

```
# Load example data; we'll pretend allele calls are unmutated
data(sample_db)

# Infer the V genotype
inferGenotype(sample_db[,"V_CALL"])

# Inger the J genotype
inferGenotype(sample_db[,"J_CALL"])
```

`insertPolymorphisms` *Insert polymorphisms into a nucleotide sequence*

Description

`insertPolymorphisms` replaces nucleotides in the desired locations of a provided sequence.

Usage

```
insertPolymorphisms(sequence, positions, nucleotides)
```

Arguments

`sequence`	the starting nucletide sequence
`positions`	a vector of positions which to be changed
`nucleotides`	a vector of nucletides to which to change the positions

Value

a sequence with the desired nucleotides in provided locations

Examples

```
insertPolymorphisms("hugged", c(1,2,6), c("t","i","r"))
```

`novelSummary` *Return a summary of any novel alleles discovered*

Description

`novelSummary` summaries the output of `runTigger`, stating which novel alleles were included in the genotype. It returns the nucleotide sequences of the novel alleles.

Usage

```
novelSummary(tigger_result, seqs_to_return = c("in genotype", "all")[1])
```

Arguments

`tigger_result`	the output of `runTigger`
`seqs_to_return`	either `"in genotype"` or `"all"`, indicating whether only those potential novel alleles alleles in the genotype should be returned or if all should be returned

Value

a named list of novel allele sequences, as well as text output indicating what number were detected versus included in the genotype

Examples

```
## Not run:
## Load example data and run all aspects of TIgGER (takes a few minutes)
data(sample_db)
data(germline_ighv)
results = runTigger(sample_db, germline_ighv)

## Summarize the detected novel alleles, add them to vector of all alleles
novel_sequences = novelSummary(results, seqs_to_return = "in genotype")
germline_ighv = c(germline_ighv, novel_sequences)
## Plot positional mutation frequency versus sequence-wide mutation count
plotNovelLines(results$novel)
## Plot nucleotide usage at polymorphic positions
plotNovelBars(results$novel)
## Plot J and junction usage for sequences perfectly matching novel alleles
plotJunctionBars(results$novel)

## View the inferred genotype
print(results$genotype)
## Get the nucleotide sequences of all genotype alleles
genotype_sequences = genotypeFasta(results$genotype, germline_ighv)

## Extract the corrected V allele calls and appened them to the data frame
V_CALL_GENOTYPED = results$new_calls
sample_db = cbind(sample_db, V_CALL_GENOTYPED)

## End(Not run)
```

`plotJunctionBars` *Visualization of J gene usage and junction length*

Description

`plotJunctionBars` shows the frequency of each combination of J gene junction length found among sequences representing unmutated versions of potential novel alleles.

Usage

```
plotJunctionBars(novel)
```

Arguments

`novel`	a list of the type returned by `detectNovelV`

Value

plot(s) of the frequency of each combination of J gene and junction length among sequences using potential novel alleles

See Also

`detectNovelV`, `runTigger`

Examples

```
## Not run:
## Load example data and run all aspects of TIgGER (takes a few minutes)
data(sample_db)
data(germline_ighv)
results = runTigger(sample_db, germline_ighv)

## Summarize the detected novel alleles, add them to vector of all alleles
novel_sequences = novelSummary(results, seqs_to_return = "in genotype")
germline_ighv = c(germline_ighv, novel_sequences)
## Plot positional mutation frequency versus sequence-wide mutation count
plotNovelLines(results$novel)
## Plot nucleotide usage at polymorphic positions
plotNovelBars(results$novel)
## Plot J and junction usage for sequences perfectly matching novel alleles
plotJunctionBars(results$novel)

## View the inferred genotype
print(results$genotype)
## Get the nucleotide sequences of all genotype alleles
genotype_sequences = genotypeFasta(results$genotype, germline_ighv)

## Extract the corrected V allele calls and appened them to the data frame
V_CALL_GENOTYPED = results$new_calls
sample_db = cbind(sample_db, V_CALL_GENOTYPED)

## End(Not run)
```

`plotNovelBars` *Visualization of nucleotide usage*

Description

`plotNovelBars` shows the nucleotide usage at polymorphic positions as a function of sequence-wide mutation count.

Usage

```
plotNovelBars(novel)
```

Arguments

`novel`	a list of the type returned by `detectNovelV`

Value

plot(s) of nucleotide usage at polymorphic positions as a function of sequence-wide mutation count.

See Also

`detectNovelV`, `runTigger`

Examples

```
## Not run:
## Load example data and run all aspects of TIgGER (takes a few minutes)
data(sample_db)
data(germline_ighv)
results = runTigger(sample_db, germline_ighv)

## Summarize the detected novel alleles, add them to vector of all alleles
novel_sequences = novelSummary(results, seqs_to_return = "in genotype")
germline_ighv = c(germline_ighv, novel_sequences)
## Plot positional mutation frequency versus sequence-wide mutation count
plotNovelLines(results$novel)
## Plot nucleotide usage at polymorphic positions
plotNovelBars(results$novel)
## Plot J and junction usage for sequences perfectly matching novel alleles
plotJunctionBars(results$novel)

## View the inferred genotype
print(results$genotype)
## Get the nucleotide sequences of all genotype alleles
genotype_sequences = genotypeFasta(results$genotype, germline_ighv)

## Extract the corrected V allele calls and appened them to the data frame
V_CALL_GENOTYPED = results$new_calls
sample_db = cbind(sample_db, V_CALL_GENOTYPED)

## End(Not run)
```

`plotNovelLines` *Visualization of positional mutation frequencies*

Description

`plotNovelLines` plots the mutation frequency of nucleotide positions as a function of sequence-wide mutation count. Potentially polymorphic positions are highlighted in red.

Usage

```
plotNovelLines(novel)
```

Arguments

`novel`	a list of the type returned by `detectNovelV`

Value

plot(s) the mutation frequency of nucleotide positions as a function of sequence-wide mutation count.

See Also

`detectNovelV`, `runTigger`

Examples

```
## Not run:
## Load example data and run all aspects of TIgGER (takes a few minutes)
data(sample_db)
data(germline_ighv)
results = runTigger(sample_db, germline_ighv)

## Summarize the detected novel alleles, add them to vector of all alleles
novel_sequences = novelSummary(results, seqs_to_return = "in genotype")
germline_ighv = c(germline_ighv, novel_sequences)
## Plot positional mutation frequency versus sequence-wide mutation count
plotNovelLines(results$novel)
## Plot nucleotide usage at polymorphic positions
plotNovelBars(results$novel)
## Plot J and junction usage for sequences perfectly matching novel alleles
plotJunctionBars(results$novel)

## View the inferred genotype
print(results$genotype)
## Get the nucleotide sequences of all genotype alleles
genotype_sequences = genotypeFasta(results$genotype, germline_ighv)

## Extract the corrected V allele calls and appened them to the data frame
V_CALL_GENOTYPED = results$new_calls
sample_db = cbind(sample_db, V_CALL_GENOTYPED)

## End(Not run)
```

`readGermlineDb` *Read a germline database*

Description

`readGermlineDb` reads a fasta-formatted file of immunoglobulin (Ig) sequences and returns a named vector of those sequences.

Usage

```
readGermlineDb(fasta_file, strip_down_name = TRUE, force_caps = TRUE)
```

Arguments

`fasta_file`	fasta-formatted file of immunoglobuling sequences
`strip_down_name`	if `TRUE`, will extract only the allele name from the strings fasta file's sequence names
`force_caps`	if `TRUE`, will force nucleotides to uppercase

Value

a named vector of strings respresenting Ig alleles

Examples

```
## Not run:
## Read an imaginary file called "foo.fasta"
foo = readGermlineDb("foo.fasta")

## End(Not run)
```

`reassignAlleles` *Correct allele calls based on a personalized genotype*

Description

`reassignAlleles` uses a subject-specific genotype to correct correct preliminary allele assignments of a set of sequences derived from a single subject.

Usage

```
reassignAlleles(v_calls, v_sequences, genotype_db)
```

Arguments

`v_calls`	a vector of strings respresenting Ig allele calls for the sequences in `v_sequences`, where multiple calls are separated by a comma
`v_sequences`	a vector of IMGT-gapped sample V sequences from a single subject
`genotype_db`	a vector of named nucleotide germline sequences matching the calls detailed in `allele_calls` and personalized to the subject

Value

a list equal in length to `v_calls`, best allele call from among the sequences listed in `genotype_db`

Examples

```
## Not run:
## Load example data and run all aspects of TIgGER (takes a few minutes)
data(sample_db)
data(germline_ighv)
results = runTigger(sample_db, germline_ighv)

## Derive the subject-specific Ig sequences
novel_sequences = novelSummary(results, seqs_to_return = "in genotype")
germline_ighv = c(germline_ighv, novel_sequences)
genotype_db = genotypeFasta(sample_output$genotype, germline_ighv)

## Extract the appropriate portions of example data
v_seqs = sapply(sample_db$SEQUENCE_IMGT, substr, 1, 312)

## Derive the vector of corrected calls
corrected_calls = reassignAlleles(sample_db$V_CALL, v_seqs, genotype_db)

## End(Not run)
```

`runTigger` *Infer genotype (including novel alleles) and correct V calls*

Description

`runTigger` takes a table of sample sequences from a single subject and a vector of database germline sequences. It then performs the following: (1) Infers the presence of novel IGHV alleles not in the germline database. (2) Infers the individuals V genotype. (3) Corrects the IGHV allele calls of the samples based on the IGHV genotype. The sample sequences should be a data frame where each row is a sequence and each column contains data about that sequence. The database germlines should be a vector of sequences with names matching those in the table of sample sequences.

Usage

```
runTigger(sample_db, germline_db, find_novel = TRUE, find_genotype = TRUE,
  correct_calls = TRUE, allele_min = 1e-04, y_intercept = 1/8,
  nt_min = 1, nt_max = 312, mut_min = 1, mut_max = 10, j_max = 0.1,
  min_seqs = 50, min_frac = 3/4, fraction_to_explain = 7/8,
  gene_cutoff = 0.001, seq_gap = "SEQUENCE_IMGT", v_call_col = "V_CALL",
  j_call_col = "J_CALL", junc_length_col = "JUNCTION_LENGTH",
  quiet = FALSE)
```

Arguments

`sample_db`	a data frame with the colums described in Details below.
`germline_db`	a vector of named nucleotide germline sequences matching the calls in `sample_db`
`find_novel`	logical. Should novel alleles be searched for?
`find_genotype`	logical. Should the genotype be inferred?
`correct_calls`	logical. Should the allele calls be corrected?

`allele_min`	a number < 1 representing the minimum fraction of sequences required for an allele to not be excluded from analysis. or a number >= 1 representing the minimum count for sequences. See `assignAlleleGroups`.
`y_intercept`	the y-intercept above which positions should be considered potentially polymorphic. See `detectNovelV`.
`nt_min`	the first nucleotide position to be considered in intercept calculations. See `detectNovelV`.
`nt_max`	the last nucleotide position to be considered in intercept calculations. See `detectNovelV`.
`mut_min`	the minimum number of mutations carried by sequences used in in intercept calculations. See `detectNovelV`.
`mut_max`	the maximum number of mutations carried by sequences used in in intercept calculations. See `detectNovelV`.
`j_max`	the maximum fraction of sequences perfectly aligning to a potential novel allele that are allowed to utilize to a particular combination of junction length and J gene. See `detectNovelV`.
`min_seqs`	the minimum number of total sequences (within the desired mutational range and nucleotide range) required for the samples to be analyzed for polymorphisms. See `detectNovelV`.
`min_frac`	the maxmium number of total sequences (within the desired mutational range and nucleotide range) required for the samples to be analyzed for polymorphisms. See `detectNovelV`.
`fraction_to_explain`	the portion of each gene that must be explained by the alleles that will be included in the genotype. See `inferGenotype`.
`gene_cutoff`	the minimum fraction of the unmutated sequences that must be attributed to a gene in order for it to be included in the genotype. See `inferGenotype`.
`seq_gap`	the name of the column in `sample_db` that includes the IMGT-gapped sequence
`v_call_col`	the name of the column in `sample_db` that includes the intial V call in the column indicated by `seq_gap`
`j_call_col`	the name of the column in `sample_db` that includes the initial J call
`junc_length_col`	the name of the column in `sample_db` that includes the junction length
`quiet`	logical indicating if additional diagonostic output will be suppressed
`v_length_col`	the name of the column in `sample_db` that includes the length of the V sequence contained within `seq_gap`

Details

The required columns that must be contained within `sample_db` are detailed below:

- `SEQUENCE_IMGT`: V(D)J sequence in the IMGT gapped format
- `V_CALL`: (Comma separated) name(s) of the nearest V allele(s)
- `J_CALL`: (Comma separated) name(s) of the nearest J allele(s)
- `JUNCTION_LENGTH`: Length of the junction region of the V(D)J sample

Value

a list containing data on new alleles, the inferred genotype, and the corrected IGHV calls.

References

Gadala-Maria D, Yaari G, Uduman M, Kleinstein SH (2015) Automated analysis of high-throughput B cell sequencing data reveals a high frequency of novel immunoglobulin V gene segment alleles. PNAS. 112(8):E862-70

See Also

`detectNovelV`, `inferGenotype`, `reassignAlleles`

Examples

```
## Not run:
## Load example data and run all aspects of TIgGER (takes a few minutes)
data(sample_db)
data(germline_ighv)
results = runTigger(sample_db, germline_ighv)

## Summarize the detected novel alleles, add them to vector of all alleles
novel_sequences = novelSummary(results, seqs_to_return = "in genotype")
germline_ighv = c(germline_ighv, novel_sequences)
## Plot positional mutation frequency versus sequence-wide mutation count
plotNovelLines(results$novel)
## Plot nucleotide usage at polymorphic positions
plotNovelBars(results$novel)
## Plot J and junction usage for sequences perfectly matching novel alleles
plotJunctionBars(results$novel)

## View the inferred genotype
print(results$genotype)
## Get the nucleotide sequences of all genotype alleles
genotype_sequences = genotypeFasta(results$genotype, germline_ighv)

## Extract the corrected V allele calls and appened them to the data frame
V_CALL_GENOTYPED = results$new_calls
sample_db = cbind(sample_db, V_CALL_GENOTYPED)

## End(Not run)
```

`sortAlleles` *Sort allele names*

Description

`sortAlleles` returns a sorted vector of strings respresenting Ig allele names. Names are first sorted by gene family, then by gene, then by allele. Duplicated genes have their alleles are sorted as if they were part of their non-duplicated counterparts (e.g. IGHV1-69D*01 comes after IGHV1-69*01 but before IGHV1-69*02), and non-localized genes (e.g. IGHV1-NL1*01) come last within their gene family.

Usage

```
sortAlleles(allele_calls)
```

Arguments

`allele_calls`	a vector of strings respresenting Ig allele names

Value

A sorted vector of strings respresenting Ig allele names

Examples

```
# Create a list of allele names
alleles = c("IGHV1-69D*01","IGHV1-69*01","IGHV1-2*01","IGHV1-69-2*01",
"IGHV2-5*01","IGHV1-NL1*01","IGHV1-2*02", "IGHV1-69*02")

# Sort the alleles
sortAlleles(alleles)
```

`summarizeMutations` *Find positional mutation counts vs sequence-wide mutation count*

Description

`summarizeMutations` takes the positions of that are similar and that are different between a set of sequences and a germline, and returns a pair of tables summarizing the positional mutation counts.

Usage

```
summarizeMutations(mut_list, match_list)
```

Arguments

`mut_list`	a list of the nucleotide positions of any differences between the two vectors of sequences, as generated by `getMutatedPositions`.
`match_list`	a list of the nucleotide positions of any similarities between the two vectors of sequences, as generated by `getMutatedPositions` where `match_instead = TRUE`.

Value

A list containing two matricies. The first details counts of sequences mutated at given positions (rows) mutated as a function of sequence-wide mutation count (columns). The second is details how many usable nucleotides (i.e., not gaps or Ns) were found for each combination of position and sequence-wide mutation count.

See Also

`getMutatedPositions`

Examples

```
# Create strings to act as a sample sequences and a reference sequence
seqs = c("----GATA","GAGAGAGA","TANA")
ref = "GATAGATA"

# Find the differences/similarities between the two
muts = getMutatedPositions(seqs, ref)
matches = getMutatedPositions(seqs, ref, match_instead =TRUE)

# Find positional mutation and nucleotide counts
summarizeMutations(muts, matches)
```

`tigger` *tigger*

Description

Here we provide a *T*ool for *I*mmuno*g*lobulin *G*enotype *E*lucidation via *R*ep-Seq (TIgGER). TIgGER inferrs the set of Ig alleles carried by an individual (including any novel alleles) and then uses this set of alleles to correct the initial assignments given to sample sequences by existing tools.

Details

Immunoglobulin Repertoire-Sequencing (Rep-Seq) data is currently the subject of much study. A key step in analyzing these data involves assigning the closest known V(D)J germline alleles to the (often somatically mutated) sample sequences using a tool such as IMGT/HighV-QUEST. However, if the sample utilizes alleles not in the germline database used for alignment, this step will fail. Additionally, this alignment has an associated error rate of ~5 The purpose of TIgGER is to address these issues.

References

Gadala-Maria et al. (2015) Automated analysis of high-throughput B cell sequencing data reveals a high frequency of novel immunoglobulin V gene segment alleles. PNAS. 112(8):E862-70

`trimMutMatrix` *Trim a mutation summary*

Description

`trimMutMatrix` takes a pair of lists as returned by `summarizeMutations` and returns a matrix of mutation frequencies at given positions (rows) as a function of sequence-wide mutation count (columns) for the desired ranges of each.

Usage

```
trimMutMatrix(mut_summary, mut_min = 1, mut_max = 10, nt_min = 1,
  nt_max = 312, min_seqs = 50, min_frac = 0.75, verbose = F)
```

Arguments

`mut_summary`	a pair of lists as returned by `summarizeMutations`
`mut_min`	the minimum number of sequence-wide mutations for sequences that will be included in the returned matrix
`mut_max`	the maximum number of sequence-wide mutations for sequences that will be included in the returned matrix
`nt_min`	the first nucleotide position to be included in the returned matrix
`nt_max`	the last nucleotide position to be included in the returned matrix
`min_seqs`	the minimum number of total sequences (within the desired mutational range and nucleotide range) required for the function to return a value
`min_frac`	the minimum fraction of sequences that must have usable nucleotides in a given position for that position to not be made `NA`
`verbose`	if `TRUE`, a message will be printed when the input causes a value of `NULL` to be returned

Value

A matrix of mutation frequencies at given positions (rows) as a function of sequence-wide mutation count (columns) for the desired ranges of each. If there is a problem with the number of sequences, etc., `NULL` will be returned.

See Also

`summarizeMutations`

Examples

```
# Create strings to act as a sample sequences and a reference sequence
seqs = c("----GATA","GAGAGAGA","GATAGGGA","TANA")
ref = "GATAGATA"

# Find the differences/similarities between the two
muts = getMutatedPositions(seqs, ref)
matches = getMutatedPositions(seqs, ref, match_instead =TRUE)

# Find positional mutation and nucleotide counts
mut_mat = summarizeMutations(muts, matches)

# Summarize the frequency for counts above one
trimMutMatrix(mut_mat, mut_max=2, nt_max=8, min_seqs=0, min_frac=0)
```

`updateAlleleNames` *Update IGHV allele names*

Description

`updateAlleleNames` takes a set of IGHV allele calls and replaces any outdated names (e.g. IGHV1-f) with the new IMGT names.

Usage

```
updateAlleleNames(allele_calls)
```

Arguments

`allele_calls`	vector of strings respresenting IGHV allele names.

Details

The updated allele names are based on IMGT release 201408-4.

Value

vector of strings respresenting updated IGHV allele names

Note

IGMT has removed IGHV2-5*10 and IGHV2-5*07 as it has determined they are actually alleles *02 and *04, respectively.

References

Xochelli et al. (2014) Immunoglobulin heavy variable (IGHV) genes and alleles: new entities, new names and implications for research and prognostication in chronic lymphocytic leukaemia. Immunogenetics. 67(1):61-6

Examples

```
# Create a vector that uses old gene/allele names.
alleles = c("IGHV1-c*01", "IGHV1-f*02", "IGHV2-5*07")

# Update the alleles to the new names
updateAlleleNames(alleles)
```

`writeFasta` *Write nucleotide sequences to a fasta file*

Description

`writeFasta` write a vector of nucleotide sequences to a file

Usage

```
writeFasta(named_sequences, file, char_per_line = 60)
```

Arguments

`named_sequences`	a vector of nucleotide sequences
`file`	a character string naming the file to write to
`char_per_line`	how many characters should be printed per line

Value

Saves a fasta file containing the sequences of interest

Examples

```
## Not run:
## Load example IGHV germlines and write them to a fasta file
data(germline_ighv)
writeFasta(germline_ighv, file="germline_ighv.fasta")

## End(Not run)
```

Index