Package 'LocaTT'

March 28, 2026

Title Geographically-Conscious Taxonomic Assignment for Metabarcoding

Version 1.2.0

Description A bioinformatics pipeline for performing taxonomic assignment of DNA metabarcoding sequence data while considering geographic location. A detailed tutorial is available at <https://urodelan.github.io/LocaTT_Tutorial/>. A manuscript describing these methods is in preparation.

License GPL (>= 3)

URL https://github.com/Urodelan/LocaTT

BugReports https://github.com/Urodelan/LocaTT/issues

Encoding UTF-8

RoxygenNote 7.3.3

Imports utils, stats, parallel

Suggests taxize

NeedsCompilation no

Author Kenen B Goodwin [aut, cre] (ORCID: <https://orcid.org/0000-0002-9219-7693>), Christopher Cousins [ctb], Taal Levi [ctb], Tiffany S Garcia [ths]

Maintainer Kenen B Goodwin <urodelan@gmail.com>

Repository CRAN

Date/Publication 2026-03-27 23:00:02 UTC

Contents

`adjust_taxonomies` *Adjust Taxonomies*

Description

Performs adjustments to a taxonomy system according to a taxonomy edits file.

Usage

```
adjust_taxonomies(
  path_to_input_file,
  path_to_output_file,
  path_to_taxonomy_edits
)
```

Arguments

path_to_input_file
: String specifying path to list of species (in CSV format) whose taxonomies are to be adjusted. The file should contain the following fields: 'Common_Name', 'Domain', 'Phylum', 'Class', 'Order', 'Family', 'Genus', 'Species'. There should be no NAs or blanks in the taxonomy fields, and the species field should contain the binomial name. Additional fields may be present in the input file, and fields can be in any order.

path_to_output_file
: String specifying path to output species list with adjusted taxonomies. The output file will be in CSV format.

path_to_taxonomy_edits
: String specifying path to taxonomy edits file in CSV format. The file must contain the following fields: 'Old_Taxonomy', 'New_Taxonomy', 'Notes'. Old taxonomies are replaced with new taxonomies in the order the records appear in the file. The taxonomic levels in the 'Old_Taxonomy' and 'New_Taxonomy' fields should be delimited by a semi-colon.

Value

No return value. Writes an output CSV file with adjusted taxonomies.

See Also

get_taxonomies.species_binomials for remotely fetching NCBI taxonomies from species binomials.

get_taxonomies.IUCN for formatting taxonomies from the IUCN Red List.

Examples

```
# Get path to input file.
path_to_input_file<-system.file("extdata",
                                "example_local_taxa_list.csv",
                                package="LocaTT",
                                mustWork=TRUE)

# Get path to taxonomy edits.
path_to_taxonomy_edits<-system.file("extdata",
                                    "example_taxonomy_edits.csv",
                                    package="LocaTT",
                                    mustWork=TRUE)

# Create temporary output file path.
path_to_output_file<-tempfile(fileext=".csv")

# Adjust taxonomies.
adjust_taxonomies(path_to_input_file=path_to_input_file,
                  path_to_output_file=path_to_output_file,
                  path_to_taxonomy_edits=path_to_taxonomy_edits)
```

binomial_test *Binomial Test*

Description

Performs binomial tests.

Usage

```
binomial_test(k, n, p, alternative = "greater")
```

Arguments

k	A numeric vector of the number of successes.
n	A numeric vector of the number of trials.
p	A numeric vector of the hypothesized probabilities of success.
alternative	A string specifying the alternative hypothesis. Must be "less" or "greater" (the default).

Details

Calls on the pbinom function in the stats package to perform vectorized binomial tests. Arguments are recycled as in pbinom. Only one-sided tests are supported, and only p-values are returned.

Value

A numeric vector of p-values from the binomial tests.

Examples

```
binomial_test(k=c(5,1,7,4),
              n=c(10,3,15,5),
              p=c(0.2,0.1,0.5,0.6),
              alternative="greater")
```

blast_command_found *Check BLAST Installation*

Description

Checks whether a BLAST program can be found.

Usage

```
blast_command_found(blast_command)
```

Arguments

blast_command	String specifying the path to a BLAST program.

Value

Logical. Returns TRUE if the BLAST program could be found.

Examples

```
blast_command_found(blast_command="blastn")
```

---

blast_version *Get BLAST Version*

Description

Gets the version of a BLAST program.

Usage

```
blast_version(blast_command = "blastn")
```

Arguments

blast_command	String specifying the path to a BLAST program. The default ('blastn') should return the version of the blastn program for standard BLAST installations. The user can provide a path to a BLAST program for non-standard BLAST installations.

Value

Returns a string of the version of the BLAST program.

Examples

```
blast_version()
```

`circle` *Draw Circle Polygon*

Description

Draws circle polygon.

Usage

```
circle(r, v = 1000, ...)
```

Arguments

`r`	Numeric scalar of circle radius.
`v`	Numeric scalar of vertex count (default = `1000`).
`...`	Additional arguments passed to `polgyon`.

Details

Draws a circle polygon of a given radius. The circle is drawn about the origin (*i.e.*, x = 0, y = 0). Intended for use with `template` to generate `detection` and `proportion` plots.

Value

No return value.

See Also

`sector` for plotting sector polygons.

Examples

```
template(l=1)
circle(r=1)
```

contains_wildcards *Check Whether DNA Sequences Contain Wildcard Characters*

Description

Checks whether DNA sequences contain wildcard characters.

Usage

```
contains_wildcards(sequences)
```

Arguments

sequences	A character vector of DNA sequences.

Value

A logical vector indicating whether each DNA sequence contains wildcard characters.

Examples

```
contains_wildcards(sequences=c("TKCTAGGTGW","CATAATTAGG","ATYGGCTATG"))
```

coordinates *Generate Circular Coordinates*

Description

Generates coordinates along a circular path.

Usage

```
coordinates(a, r)
```

Arguments

a	Numeric vector of angles (degrees).
r	Numeric scalar of the circle radius.

Details

Calculates xy coordinates along a circular path given a vector of angles and a specified radius. The center of the circle is at the origin (*i.e.*, x = 0, y = 0). This function is helpful for calculating the vertices of circle and sector polygons.

Value

A numeric matrix of xy coordinates.

See Also

`circle` for plotting circle polygons.

`sector` for plotting sector polygons.

Examples

```
coordinates(a=c(90,180,270,360),r=1)
```

`decode_quality_scores` *Decode DNA Sequence Quality Scores*

Description

Decodes Phred quality scores in Sanger format from symbols to numeric values.

Usage

```
decode_quality_scores(symbols)
```

Arguments

`symbols`	A string containing quality scores encoded as symbols in Sanger format.

Value

A numeric vector of Phred quality scores.

Examples

```
decode_quality_scores(symbols="989!.C;F@\"")
```

`detection` *Grouped Detection Plot*

Description

Generates detection plots for multiple groups.

Usage

```
detection(
  x,
  r = 1,
  b = 0.025,
  v = 1000,
  w = 1,
  f = 0.5,
  c = "lightskyblue",
  m = 3,
  ...
)
```

Arguments

`x`	A list of vectors named `"g"`, `"s"`, `"r"`, and `"d"`. The elements of vector `"g"` (character, numeric, or factor) specify the group. The elements of vector `"s"` (character or numeric) specify the sample. The elements of vector `"r"` (numeric) specify the number of replicates within sample `"s"`. The elements of vector `"d"` (numeric) specify the number of replicates within sample `"s"` with detections.
`r`	Numeric scalar. Radius of plot circle (default = `1`).
`b`	Numeric scalar. Plot radius buffer (proportion; default = `0.025`).
`v`	Numeric scalar. Vertex count of plot circle (default = `1000`).
`w`	Numeric scalar. Line width of outer circle (default = `1`).
`f`	Numeric scalar. Line width of sectors as a proportion of `w` (default = `0.5`).
`c`	Character string. Fill color of sub-sector detections (default = `"lightskyblue"`).
`m`	Numeric scalar. Maximum number of plot columns (default = `3`).
`...`	Additional arguments passed to `title`.

Details

Produces a pie-chart-like detection plot with grouping structure. Each circle represents a group. Each sector represents a sample, and each sub-sector represents a replicate. Filled replicates represent detections. Groups are sorted alphabetically (or inherit factor level ordering) and arranged from left to right and top to bottom. Samples are sorted alphabetically and arranged in a clockwise orientation (from angle zero). Samples are sorted independently for each group. This plot design is specialized for visualizing binary detection data.

Value

No return value.

References

A manuscript describing this plot design is in preparation.

See Also

singular.detection for singular detection plots.

proportion for grouped proportion plots.

Examples

```
set.seed(1234)
n.groups<-6
n.samples<-6
n.replicates<-3
data<-list(g=rep(x=LETTERS[1:n.groups],each=n.samples),
           s=rep(x=letters[1:n.samples],times=n.groups),
           r=rep(x=n.replicates,times=n.groups*n.samples),
           d=sample(x=0:n.replicates,size=n.groups*n.samples,
                    replace=TRUE))
detection(x=data)
```

---

expand_taxonomies *Expand Taxonomies*

Description

Extracts each taxonomic level from a vector of taxonomic strings.

Usage

```
expand_taxonomies(
  taxonomies,
  levels = c("Domain", "Phylum", "Class", "Order", "Family", "Genus", "Species"),
  full_names = TRUE,
  delimiter = ";",
  ignore
)
```

Arguments

taxonomies	A character vector of taxonomic strings.
levels	A character vector of taxonomic level names. The length of levels determines the number of taxonomic levels to extract from the taxonomies, and the order of elements in levels is assumed to match the order of taxonomic levels in the taxonomies. levels is also used as the field names of the returned data frame (see return value section). The default vector includes: "Domain", "Phylum", "Class", "Order", "Family", "Genus", and "Species".
full_names	Logical. If TRUE (the default), then full taxonomies are returned down to the extracted taxonomic level. If FALSE, then only the extracted taxonomic level is returned.
delimiter	A character string of the delimiter between taxonomic levels in the input taxonomies. The default is ";".
ignore	An optional character vector of taxonomic strings for which taxonomic expansion will be skipped. In the returned data frame (see return value section), the record for each skipped taxonomic string will be filled with NAs.

Value

Returns a data frame of extracted taxonomic levels. One record for each element of taxonomies, and one field for each element of levels. Field names are inherited from levels. If a taxonomic level is not present in a taxonomic string, then the respective cell in the returned data frame will contain NA.

See Also

get_taxonomic_level for extracting a taxonomic level from taxonomic strings.

get_consensus_taxonomy for generating a consensus taxonomy from taxonomic strings.

Examples

```
expand_taxonomies(taxonomies=
   c("Eukaryota;Chordata;Amphibia;Caudata;Ambystomatidae;Ambystoma;Ambystoma mavortium",
     "Eukaryota;Chordata;Amphibia;Anura;Bufonidae;Anaxyrus;Anaxyrus boreas",
     "Eukaryota;Chordata;Amphibia;Anura;Ranidae;Rana;Rana luteiventris"),
                    full_names=FALSE,
                    delimiter=";")
```

---

filter_sequences *Filter DNA Sequences by PCR Replicates*

Description

Filters DNA sequences by minimum read count within a PCR replicate, minimum proportion within a PCR replicate, and number of detections across PCR replicates.

Usage

```
filter_sequences(
  input_files,
  samples,
  PCR_replicates,
  output_file,
  minimum_reads.PCR_replicate = 1,
  minimum_reads.sequence = 1,
  minimum_proportion.sequence = 0.005,
  binomial_test.enabled = TRUE,
  binomial_test.p.adjust.method = "none",
  binomial_test.alpha_level = 0.05,
  minimum_PCR_replicates = 2,
  delimiter.read_counts = ": ",
  delimiter.PCR_replicates = ", "
)
```

Arguments

`input_files`	A character vector of file paths to input FASTA files. DNA sequences in the input FASTA files are assumed to be summarized by frequency of occurrence, with each FASTA header line beginning with "Frequency: " and followed by the sequence's read count. Output FASTA files from `truncate_and_merge_pairs` have this format and can be used directly with this function. Each input FASTA file is assumed to contain the DNA sequence reads for a single PCR replicate for a single sample.
`samples`	A character vector of sample identifiers, with one element for each element of `input_files`.
`PCR_replicates`	A character vector of PCR replicate identifiers, with one element for each element of `input_files`.
`output_file`	String specifying path to output file of filtered sequences in CSV format.
`minimum_reads.PCR_replicate`	Numeric. PCR replicates which contain fewer reads than this value are discarded and do not contribute detections to any sequence. The default is `1` (*i.e.*, no PCR replicates discarded).
`minimum_reads.sequence`	Numeric. For a sequence to be considered detected within a PCR replicate, the sequence's read count within the PCR replicate must match or exceed this value. The default is `1` (*i.e.*, no filtering by minimum read count within PCR replicates).
`minimum_proportion.sequence`	Numeric. For a sequence to be considered detected within a PCR replicate, the proportion of reads in the PCR replicate comprised by the sequence must exceed this value. If `binomial_test.enabled = TRUE`, then this argument is used as the null hypothesis for a one-sided binomial test, and a significance test is used to determine whether the minimum proportion requirement for detection

is satisfied instead. See the binomial_test.enabled argument below. The default is 0.005 (*i.e.*, 0.5%). To disable sequence filtering by minimum proportion within PCR replicates, set to 0.

binomial_test.enabled
: Logical. If TRUE (the default), then for a sequence to be considered detected within a PCR replicate, the proportion of reads in the PCR replicate comprised by the sequence must significantly exceed the value of the minimum_proportion.sequence argument at the provided alpha level (binomial_test.alpha_level argument) based on a one-sided binomial test (*i.e.*, binomial_test with alternative = "greater"). Optionally, p-values within a PCR replicate can be adjusted for multiple hypothesis testing by setting the binomial_test.p.adjust.method argument below. To disable significance testing, set to FALSE (minimum proportion filtering will still occur if minimum_proportion.sequence > 0, see above).

binomial_test.p.adjust.method
: String specifying the p-value adjustment method for multiple hypothesis testing. p-value adjustments are performed within each PCR replicate for each sample. Passed to the method argument of p.adjust in the stats package. Available methods are contained within the stats::p.adjust.methods vector. If "none" (the default), then p-value adjustments are not performed. Ignored if binomial_test.enabled = FALSE.

binomial_test.alpha_level
: Numeric. The alpha level used in deciding whether the proportion of reads in a PCR replicate comprised by a sequence significantly exceeds a minimum threshold required for detection. See the binomial_test.enabled argument. The default is 0.05. Ignored if binomial_test.enabled = FALSE.

minimum_PCR_replicates
: Numeric. The minimum number of PCR replicates in which a sequence must be detected in order to be considered present (*i.e.*, not erroneous) in a sample. The default is 2.

delimiter.read_counts
: String specifying the delimiter between PCR replicate identifiers and sequence read counts in the Read_count_by_PCR_replicate field of the output CSV file (see details section). The default is ": ".

delimiter.PCR_replicates
: String specifying the delimiter between PCR replicates in the Read_count_by_PCR_replicate field of the output CSV file (see details section). The default is ", ".

Details

For each set of input polymerase chain reaction (PCR) replicate FASTA files associated with a sample, writes out DNA sequences which are detected across a minimum number of PCR replicates (minimum_PCR_replicates argument). Detection within a PCR replicate is defined as a sequence having at least a minimum read count *and* exceeding a minimum proportion of reads (minimum_reads.sequence and minimum_proportion.sequence arguments, respectively). When binomial_test.enabled = TRUE, a sequence must significantly exceed the minimum proportion within a PCR replicate at the provided alpha level (binomial_test.alpha_level argument) based on a one-sided binomial test (*i.e.*, binomial_test with alternative = "greater"). Within a PCR replicate, p-values can be adjusted for multiple hypothesis testing by setting the binomial_test.p.adjust.method

argument (see stats::p.adjust.methods and p.adjust in the stats package). PCR replicates which contain fewer than a minimum number of reads are discarded (minimum_reads.PCR_replicate argument) and do not contribute detections to any sequence.

DNA sequences in the input FASTA files are assumed to be summarized by frequency of occurrence, with each FASTA header line beginning with "Frequency: " and followed by the sequence's read count. Output FASTA files from truncate_and_merge_pairs have this format and can be used directly with this function. Each input FASTA file is assumed to contain the DNA sequence reads for a single PCR replicate for a single sample.

For pipeline calibration purposes, a data frame containing unfiltered DNA sequences with their read counts, proportions, and p-values in each PCR replicate is invisibly returned (see return value section). While the primary output of this function is the written CSV file of filtered sequences (described below), the invisibly returned data frame of unfiltered sequences can be helpful when calibrating or troubleshooting filtering parameters. To aid in troubleshooting filtering parameters, the data frame is invisibly returned even if the error "Filtering removed all sequences" is received.

For the primary output, this function writes a CSV file of filtered DNA sequences with the following field definitions:

- Sample: The sample name.
- Sequence: The DNA sequence.
- Detections_across_PCR_replicates: The number of PCR replicates the sequence was detected in.
- Read_count_by_PCR_replicate: The sequence's read count in each PCR replicate the sequence was detected in.
- Sequence_read_count: The sequence's total read count across the PCR replicates the sequence was detected in. Calculated as the sum of the read counts in the Read_count_by_PCR_replicate field.
- Sample_read_count: The sample's total read count across all sequences detected in the PCR replicates. Calculated as the sum of the read counts in Sequence_read_count field associated with the sample.
- Proportion_of_sample: The proportion of sample reads comprised by the sequence. Calculated by dividing the Sequence_read_count field by the Sample_read_count field. Equivalent to the weighted average of the sequence's proportion in each PCR replicate, with weights given by the proportion of the sample's total reads contained in each PCR replicate.

Value

Invisibly returns a data frame containing unfiltered DNA sequences with their read counts, proportions, and p-values in each PCR replicate. While the primary output of this function is the written CSV file of filtered sequences described in the details section, the invisibly returned data frame of unfiltered sequences can be helpful when calibrating or troubleshooting filtering parameters. To aid in troubleshooting filtering parameters, the data frame is invisibly returned even if the error "Filtering removed all sequences" is received. Field definitions for the invisibly returned data frame of unfiltered sequences are:

- Sample: The sample name.

- PCR_replicate: The PCR replicate identifier.
- Sequence: The DNA sequence.
- Read_count.sequence: The sequence's read count within the PCR replicate.
- Read_count.PCR_replicate: The number of reads in the PCR replicate.
- Proportion_of_PCR_replicate.observed: The proportion of reads in the PCR replicate comprised by the sequence.
- Proportion_of_PCR_replicate.null (Field only present if `binomial_test.enabled = TRUE`): The null hypothesis for a one-sided binomial test (inherited from the `minimum_proportion.sequence` argument). See the p.value field below.
- p.value (Field only present if `binomial_test.enabled = TRUE`): The p-value from a one-sided binomial test of whether the proportion of reads in the PCR replicate comprised by the sequence exceeds the null hypothesis (*i.e.*, `binomial_test` with `alternative = "greater"`).
- p.value.adjusted (Field only present if `binomial_test.enabled = TRUE`): The p-value from the one-sided binomial test adjusted for multiple comparisons within each PCR replicate for each sample. See the p.value_adjustment_method field below.
- p.value_adjustment_method (Field only present if `binomial_test.enabled = TRUE`): The p-value adjustment method (inherited from the `binomial_test.p.adjust.method` argument).

References

A manuscript describing these methods is in preparation.

See Also

`binomial_test` for performing vectorized one-sided binomial tests.

`truncate_and_merge_pairs` for truncating and merging read pairs prior to sequence filtering.

`local_taxa_tool` for performing geographically-conscious taxonomic assignment of filtered sequences.

Examples

```
# Get example FASTA files.
input_files<-system.file("extdata",
                         paste0(rep(x=paste0("S0",1:3),
                                    each=3),
                                "P0",1:3,".fasta"),
                         package="LocaTT",
                         mustWork=TRUE)

# Create path for temporary output file.
output_file<-tempfile(fileext=".csv")

# Specify samples.
samples<-rep(x=paste0("S0",1:3),each=3)

# Specify replicates.
```

```
PCR_replicates<-rep(x=paste0("P0",1:3),times=3)

# Filter sequences.
filter_sequences(input_files=input_files,
                 samples=samples,
                 PCR_replicates=PCR_replicates,
                 output_file=output_file)
```

`format_reference_database` *Format Reference Databases*

Description

Formats reference databases from MIDORI or UNITE for use with the `local_taxa_tool` function.

Usage

```
format_reference_database(
  path_to_input_database,
  path_to_output_database,
  input_database_source = "MIDORI",
  path_to_taxonomy_edits = NA,
  path_to_sequence_edits = NA,
  path_to_taxa_subset_list = NA,
  makeblastdb_command = "makeblastdb",
  ...
)
```

Arguments

`path_to_input_database`
: String specifying path to input reference database in FASTA format.

`path_to_output_database`
: String specifying path to output BLAST database in FASTA format. File path cannot contain spaces.

`input_database_source`
: String specifying input reference database source (`'MIDORI'` or `'UNITE'`). The default is `'MIDORI'`.

`path_to_taxonomy_edits`
: String specifying path to taxonomy edits file in CSV format. The file must contain the following fields: 'Old_Taxonomy', 'New_Taxonomy', 'Notes'. Old taxonomies are replaced with new taxonomies in the order the records appear in the file. The taxonomic levels in the 'Old_Taxonomy' and 'New_Taxonomy' fields should be delimited by a semi-colon. If no taxonomy edits are desired, then set this variable to `NA` (the default).

`path_to_sequence_edits`
: String specifying path to sequence edits file in CSV format. The file must contain the following fields: 'Action', 'Common_Name', 'Domain', 'Phylum', 'Class', 'Order', 'Family', 'Genus', 'Species', 'Sequence', 'Notes'. The values in the 'Action' field must be either 'Add' or 'Remove', which will add or remove the respective sequence from the reference database. Values in the 'Common_Name' field are optional. Values should be supplied to all taxonomy fields. If using a reference database from MIDORI, then use NCBI domain names (*e.g.*, 'Eukaryota') in the 'Domain' field. If using a reference database from UNITE, then use kingdom names (*e.g.*, 'Fungi') in the 'Domain' field. The 'Species' field should contain species binomials. Sequence edits are performed after taxonomy edits, if applied. If no sequence edits are desired, then set this variable to `NA` (the default).

`path_to_taxa_subset_list`
: String specifying path to list of species (in CSV format) to subset the reference database to. This option is helpful if the user wants the reference database to include only the sequences of local species. The file should contain the following fields: 'Common_Name', 'Domain', 'Phylum', 'Class', 'Order', 'Family', 'Genus', 'Species'. There should be no `NA`s or blanks in the taxonomy fields. The species field should contain the binomial name without subspecies or other information below the species level. There should be no duplicate species (*i.e.*, multiple records with the same species binomial and taxonomy) in the species list. Subsetting the reference database to the sequences of certain species is performed after taxonomy and sequence edits are applied to the reference database, and species must match at all taxonomic levels in order to be retained in the reference database. If subsetting the reference database to the sequences of certain species is not desired, set this variable to `NA` (the default).

`makeblastdb_command`
: String specifying path to the makeblastdb program, which is a part of BLAST. The default (`'makeblastdb'`) should work for standard BLAST installations. The user can provide a path to the makeblastdb program for non-standard BLAST installations.

`...`
: Accepts former argument names for backwards compatibility.

Value

No return value. Writes formatted BLAST database files.

See Also

`local_taxa_tool` for performing geographically-conscious taxonomic assignment.

`adjust_taxonomies` for adjusting a taxonomy system.

Examples

```
# Get path to example reference sequences FASTA file.
path_to_input_file<-system.file("extdata",
                                "example_reference_sequences.fasta",
```

```
                                    package="LocaTT",
                                    mustWork=TRUE)

# Create a temporary file path for the output reference database FASTA file.
path_to_output_file<-tempfile(fileext=".fasta")

# Format reference database.
format_reference_database(path_to_input_database=path_to_input_file,
                          path_to_output_database=path_to_output_file)
```

`get_consensus_taxonomy` *Get Consensus Taxonomy from Taxonomic Strings*

Description

Gets the consensus taxonomy from a vector of taxonomic strings.

Usage

```
get_consensus_taxonomy(taxonomies, full_names = TRUE, delimiter = ";")
```

Arguments

`taxonomies`	A character vector of taxonomic strings.
`full_names`	Logical. If `TRUE` (the default), then the full consensus taxonomy is returned. If `FALSE`, then only the lowest taxonomic level of the consensus taxonomy is returned.
`delimiter`	A character string of the delimiter between taxonomic levels in the input taxonomies. The default is `";"`.

Value

A character string containing the taxonomy agreed upon by all input taxonomies. If the input taxonomies are not the same at any taxonomic level, then `NA` is returned.

See Also

`get_taxonomic_level` for extracting a taxonomic level from taxonomic strings.

`expand_taxonomies` for extracting each taxonomic level from a vector of taxonomic strings.

Examples

```
get_consensus_taxonomy(taxonomies=
   c("Eukaryota;Chordata;Amphibia;Caudata;Ambystomatidae;Ambystoma;Ambystoma_mavortium",
     "Eukaryota;Chordata;Amphibia;Anura;Bufonidae;Anaxyrus;Anaxyrus_boreas",
     "Eukaryota;Chordata;Amphibia;Anura;Ranidae;Rana;Rana_luteiventris"),
                      full_names=TRUE,
                      delimiter=";")
```

---

get_taxonomic_level *Get Specified Taxonomic Level from Taxonomic Strings*

Description

Gets the specified taxonomic level from a vector of taxonomic strings.

Usage

```
get_taxonomic_level(taxonomies, level, full_names = TRUE, delimiter = ";")
```

Arguments

`taxonomies`	A character vector of taxonomic strings.
`level`	A numeric value representing the taxonomic level to be extracted. A value of `1` retrieves the highest taxonomic level (*e.g.*, domain) from the input taxonomies, with each sequentially higher value retrieving sequentially lower taxonomic levels. `0` is a special value which retrieves the lowest taxonomic level available in the input taxonomies.
`full_names`	Logical. If `TRUE` (the default), then full taxonomies are returned down to the requested taxonomic level. If `FALSE`, then only the requested taxonomic level is returned.
`delimiter`	A character string of the delimiter between taxonomic levels in the input taxonomies. The default is `";"`.

Value

A character vector containing the requested taxonomic level for each element of the input taxonomies.

See Also

`expand_taxonomies` for extracting each taxonomic level from a vector of taxonomic strings.

`get_consensus_taxonomy` for generating a consensus taxonomy from taxonomic strings.

Examples

```
get_taxonomic_level(taxonomies=
   c("Eukaryota;Chordata;Amphibia;Caudata;Ambystomatidae;Ambystoma;Ambystoma_mavortium",
     "Eukaryota;Chordata;Amphibia;Anura;Bufonidae;Anaxyrus;Anaxyrus_boreas",
     "Eukaryota;Chordata;Amphibia;Anura;Ranidae;Rana;Rana_luteiventris"),
   level=5,
   full_names=TRUE,
   delimiter=";")
```

---

get_taxonomies.IUCN *Get Taxonomies from IUCN Red List Files*

Description

Formats taxonomies from IUCN Red List taxonomy.csv and common_names.csv files for use with the local_taxa_tool function.

Usage

```
get_taxonomies.IUCN(
  path_to_taxonomies,
  path_to_common_names,
  path_to_output_file,
  domain = "Eukaryota",
  path_to_taxonomy_edits = NA,
  ...
)
```

Arguments

path_to_taxonomies	String specifying path to input IUCN Red List taxonomy.csv file.
path_to_common_names	String specifying path to input IUCN Red List common_names.csv file.
path_to_output_file	String specifying path to output species list (in CSV format) with formatted taxonomies.
domain	String specifying the domain name to use for all species. The IUCN Red List files do not include domain information, so a domain name must be provided. If using a reference database from UNITE, provide a kingdom name here (*e.g.*, 'Fungi'). The default is 'Eukaryota'.
path_to_taxonomy_edits	String specifying path to taxonomy edits file in CSV format. The file must contain the following fields: 'Old_Taxonomy', 'New_Taxonomy', 'Notes'. Old taxonomies are replaced with new taxonomies in the order the records appear in the file. The taxonomic levels in the 'Old_Taxonomy' and 'New_Taxonomy' fields should be delimited by a semi-colon. If no taxonomy edits are desired, then set this variable to NA (the default).

... Accepts former argument names for backwards compatibility.

Value

No return value. Writes an output CSV file with formatted taxonomies.

See Also

get_taxonomies.species_binomials for remotely fetching NCBI taxonomies from species binomials.

adjust_taxonomies for adjusting a taxonomy system.

Examples

```
# Get path to example taxonomy CSV file.
path_to_taxonomies<-system.file("extdata",
                                "example_taxonomy.csv",
                                package="LocaTT",
                                mustWork=TRUE)

# Get path to example common names CSV file.
path_to_common_names<-system.file("extdata",
                                  "example_common_names.csv",
                                  package="LocaTT",
                                  mustWork=TRUE)

# Create a temporary file path for the output CSV file.
path_to_output_file<-tempfile(fileext=".csv")

# Format common names and taxonomies.
get_taxonomies.IUCN(path_to_taxonomies=path_to_taxonomies,
                    path_to_common_names=path_to_common_names,
                    path_to_output_file=path_to_output_file)
```

---

get_taxonomies.species_binomials

Get NCBI Taxonomies from Species Binomials

Description

Remotely fetches taxonomies from the NCBI taxonomy database for a list of species binomials. Installation of the taxize package is required to use this function.

Usage

```
get_taxonomies.species_binomials(
  path_to_species_binomials,
  path_to_output_file,
```

```
  path_to_taxonomy_edits = NA,
  print_queries = TRUE,
  ...
)
```

Arguments

path_to_species_binomials	String specifying path to input species list with common and scientific names. The file should be in CSV format and contain the following fields: 'Common_Name', 'Scientific_Name'. Values in the 'Common_Name' field are optional. Values in the 'Scientific_Name' field are required.
path_to_output_file	String specifying path to output species list with added NCBI taxonomies. The output file will be in CSV format.
path_to_taxonomy_edits	String specifying path to taxonomy edits file in CSV format. The file must contain the following fields: 'Old_Taxonomy', 'New_Taxonomy', 'Notes'. Old taxonomies are replaced with new taxonomies in the order the records appear in the file. The taxonomic levels in the 'Old_Taxonomy' and 'New_Taxonomy' fields should be delimited by a semi-colon. If no taxonomy edits are desired, then set this variable to NA (the default).
print_queries	Logical. Whether taxa queries should be printed. The default is TRUE.
...	Accepts former argument names for backwards compatibility.

Value

No return value. Writes an output CSV file with added taxonomies. Species which could not be found in the NCBI taxonomy database appear in the top records of the output file.

See Also

get_taxonomies.IUCN for formatting taxonomies from the IUCN Red List.

adjust_taxonomies for adjusting a taxonomy system.

Examples

```
# Get path to example input species binomials CSV file.
path_to_species_binomials<-system.file("extdata",
                                       "example_species_binomials.csv",
                                       package="LocaTT",
                                       mustWork=TRUE)

# Create a temporary file path for the output CSV file.
path_to_output_file<-tempfile(fileext=".csv")

# Fetch taxonomies from species binomials.
get_taxonomies.species_binomials(path_to_species_binomials=path_to_species_binomials,
```

```
                                path_to_output_file=path_to_output_file,
                                print_queries=FALSE)
```

`isolate_amplicon` *Trim DNA Sequences to an Amplicon Region Using Forward and Reverse Primer Sequences*

Description

Trims DNA sequences to an amplicon region using forward and reverse primer sequences. Ambiguous nucleotides in forward and reverse primers are supported.

Usage

```
isolate_amplicon(sequences, forward_primer, reverse_primer)
```

Arguments

`sequences`	A character vector of DNA sequences to trim to the amplicon region.
`forward_primer`	A string specifying the forward primer sequence. Can contain ambiguous nucleotides.
`reverse_primer`	A string specifying the reverse primer sequence. Can contain ambiguous nucletodies.

Details

For each DNA sequence, nucleotides matching and preceding the forward primer are removed, and nucleotides matching and following the reverse complement of the reverse primer are removed. The reverse complement of the reverse primer is internally derived from the reverse primer using the `reverse_complement` function. Ambiguous nucleotides in primers (*i.e.*, the forward and reverse primer arguments) are supported through the internal use of the `substitute_wildcards` function on the forward primer and the reverse complement of the reverse primer, and primer regions in DNA sequences are located using regular expressions. Trimming will fail for DNA sequences which contain ambiguous nucleotides in their primer regions (*e.g.*, Ns), resulting in `NA`s for those sequences.

Value

A character vector of DNA sequences trimmed to the amplicon region. `NA`s are returned for DNA sequences which could not be trimmed, which occurs when either primer region is missing from the DNA sequence or when the forward primer region occurs after a region matching the reverse complement of the reverse primer.

Examples

```
isolate_amplicon(sequences=c("ACACAATCGTGTTTATATTAACTTCAAGAGTGGGCATAGG",
                             "CGTGACAATCATGTTTGTGATTCGTACAAAAGTGCGTCCT"),
                 forward_primer="AATCRTGTTT",
                 reverse_primer="CSCACTHTTG")
```

`local_taxa_tool` *Perform Geographically-Conscious Taxonomic Assignment*

Description

Performs taxonomic assignment of DNA metabarcoding sequences while considering geographic location.

Usage

```
local_taxa_tool(
  path_to_query_sequences,
  path_to_BLAST_database,
  path_to_output_file,
  path_to_local_taxa_list = NA,
  include_missing = FALSE,
  blast_e_value = 1e-05,
  blast_max_target_seqs = 2000,
  blast_task = "megablast",
  full_names = FALSE,
  underscores = FALSE,
  separator = ", ",
  blastn_command = "blastn",
  ...
)
```

Arguments

`path_to_query_sequences`
: String specifying path to FASTA file containing sequences to classify. File path cannot contain spaces.

`path_to_BLAST_database`
: String specifying path to BLAST reference database in FASTA format. File path cannot contain spaces.

`path_to_output_file`
: String specifying path to output file of classified sequences in CSV format.

`path_to_local_taxa_list`
: String specifying path to list of local species in CSV format. The file should contain the following fields: 'Common_Name', 'Domain', 'Phylum', 'Class', 'Order', 'Family', 'Genus', 'Species'. There should be no 'NA's or blanks in

the taxonomy fields. The species field should contain the binomial name without subspecies or other information below the species level. There should be no duplicate species (*i.e.*, multiple records with the same species binomial and taxonomy) in the local species list. If local taxa suggestions are not desired, set this variable to NA (the default).

include_missing
: Logical. If TRUE, then additional fields are included in the output CSV file in which local sister taxonomic groups without reference sequences are added to the local taxa suggestions. If FALSE (the default), then this is not performed.

blast_e_value
: Numeric. Maximum E-value of returned BLAST hits (lower E-values are associated with more 'significant' matches). The default is 1e-05.

blast_max_target_seqs
: Numeric. Maximum number of BLAST target sequences returned per query sequence. Enough target sequences should be returned to ensure that all minimum E-value matches are returned for each query sequence. A warning will be produced if this value is not sufficient. The default is 2000.

blast_task
: String specifying BLAST task specification. Use 'megablast' (the default) to find very similar sequences (*e.g.*, intraspecies or closely related species). Use 'blastn-short' for sequences shorter than 50 bases. See the blastn program help documentation for additional options and details.

full_names
: Logical. If TRUE, then full taxonomies are returned in the output CSV file. If FALSE (the default), then only the lowest taxonomic levels (*e.g.*, species binomials instead of the full species taxonomies) are returned in the output CSV file.

underscores
: Logical. If TRUE, then taxa names in the output CSV file use underscores instead of spaces. If FALSE (the default), then taxa names in the output CSV file use spaces.

separator
: String specifying the separator to use between taxa names in the output CSV file. The default is ', '.

blastn_command
: String specifying path to the blastn program. The default ('blastn') should work for standard BLAST installations. The user can provide a path to the blastn program for non-standard BLAST installations.

...
: Accepts former argument names for backwards compatibility.

Details

Sequences are BLASTed against a global reference database, and the tool suggests locally occurring species which are most closely related (by taxonomy) to any of the best-matching BLAST hits (by bit score). Optionally, local sister taxonomic groups without reference sequences can be added to the local taxa suggestions by setting the include_missing argument to TRUE. If a local taxa list is not provided, then local taxa suggestions will be disabled, but all best-matching BLAST hits will still be returned. Alternatively, a reference database containing just the sequences of local species can be used, and local taxa suggestions can be disabled to return all best BLAST matches of local species. The reference database should be formatted with the format_reference_database function, and the local taxa lists can be prepared using the get_taxonomies.species_binomials and get_taxonomies.IUCN functions. Output field definitions are:

- Sequence_name: The query sequence name.
- Sequence: The query sequence.
- Best_match_references: Species binomials of all best-matching BLAST hits (by bit score) from the reference database.
- Best_match_E_value: The E-value associated with the best-matching BLAST hits.
- Best_match_bit_score: The bit score associated with the best-matching BLAST hits.
- Best_match_query_cover.mean: The mean query cover of all best-matching BLAST hits.
- Best_match_query_cover.SD: The standard deviation of query cover of all best-matching BLAST hits.
- Best_match_PID.mean: The mean percent identity of all best-matching BLAST hits.
- Best_match_PID.SD: The standard deviation of percent identity of all best-matching BLAST hits.
- Local_taxa (Field only present if a path to a local taxa list is provided): The finest taxonomic unit(s) which include both any species of the best-matching BLAST hits and any local species. If the species of any of the best-matching BLAST hits are local, then the finest taxonomic unit(s) are at the species level.
- Local_species (Field only present if a path to a local taxa list is provided): Species binomials of all local species which belong to the taxonomic unit(s) in the Local_taxa field.
- Local_taxa.include_missing (Field only present if both a path to a local taxa list is provided and the `include_missing` argument is set to `TRUE`): Local sister taxonomic groups without reference sequences are added to the local taxa suggestions from the Local_taxa field.
- Local_species.include_missing (Field only present if both a path to a local taxa list is provided and `include_missing` argument is set to `TRUE`): Species binomials of all local species which belong to the taxonomic unit(s) in the Local_taxa.include_missing field.

Value

No return value. Writes an output CSV file with fields defined in the details section.

References

A manuscript describing this taxonomic assignment method is in preparation.

See Also

`format_reference_database` for formatting reference databases.

`get_taxonomies.species_binomials` and `get_taxonomies.IUCN` for creating local taxa lists.

`adjust_taxonomies` for adjusting a taxonomy system.

Examples

```
# Get path to example query sequences FASTA file.
path_to_query_sequences<-system.file("extdata",
                                     "example_query_sequences.fasta",
                                     package="LocaTT",
                                     mustWork=TRUE)

# Get path to example BLAST reference database FASTA file.
path_to_BLAST_database<-system.file("extdata",
                                    "example_blast_database.fasta",
                                    package="LocaTT",
                                    mustWork=TRUE)

# Get path to example local taxa list CSV file.
path_to_local_taxa_list<-system.file("extdata",
                                     "example_local_taxa_list.csv",
                                     package="LocaTT",
                                     mustWork=TRUE)

# Create a temporary file path for the output CSV file.
path_to_output_file<-tempfile(fileext=".csv")

# Run the local taxa tool.
local_taxa_tool(path_to_query_sequences=path_to_query_sequences,
                path_to_BLAST_database=path_to_BLAST_database,
                path_to_output_file=path_to_output_file,
                path_to_local_taxa_list=path_to_local_taxa_list,
                include_missing=TRUE,
                full_names=TRUE,
                underscores=TRUE)
```

`merge_pairs` *Merge Forward and Reverse DNA Sequence Reads*

Description

Merges forward and reverse DNA sequence reads.

Usage

```
merge_pairs(forward_reads, reverse_reads, minimum_overlap = 10)
```

Arguments

`forward_reads`	A character vector of forward DNA sequence reads.
`reverse_reads`	A character vector of reverse DNA sequence reads.

`minimum_overlap`
: Numeric. The minimum length of an overlap that must be found between the end of the forward read and the start of the reverse complement of the reverse read in order for a read pair to be merged. The default is `10`.

Details

For each pair of forward and reverse DNA sequence reads, the reverse complement of the reverse read is internally derived using the `reverse_complement` function, and the read pair is merged into a single sequence if an overlap of at least the minimum length is found between the end of the forward read and the start of the reverse complement of the reverse read. If an overlap of the minimum length is not found, then an `NA` is returned for the merged read pair.

Value

A character vector of merged DNA sequence read pairs. `NA`s are returned for read pairs which could not be merged, which occurs when an overlap of at least the minimum length is not found between the end of the forward read and the start of the reverse complement of the reverse read.

See Also

`truncate_and_merge_pairs` for truncating and merging forward and reverse DNA sequence reads.

Examples

```
merge_pairs(forward_reads=c("CCTTACGAATCCTGT","TTCTCCACCCGCGGATA","CGCCCGGAGTCCCTGTAGTA"),
            reverse_reads=c("GACAAACAGGATTCG","CAATATCCGCGGGTG","TACTACAGGGACTCC"))
```

---

proportion *Grouped Proportion Plot*

Description

Generates proportion plots for multiple groups.

Usage

```
proportion(
  x,
  r = 1,
  b = 0.025,
  v = 1000,
  w = 1,
  f = 0.5,
  c = "lightskyblue",
  s = FALSE,
  a = TRUE,
  m = 3,
  ...
)
```

Arguments

x	A list of vectors named "g", "s", and "p". The elements of vector "g" (character, numeric, or factor) specify the group. The elements of vector "s" (character or numeric) specify the sample. The elements of vector "p" (numeric) specify the proportional abundance within sample "s".
r	Numeric scalar. Radius of plot circle (default = 1).
b	Numeric scalar. Plot radius buffer (proportion; default = 0.025).
v	Numeric scalar. Vertex count of plot circle (default = 1000).
w	Numeric scalar. Line width of outer circle (default = 1).
f	Numeric scalar. Line width of sectors as a proportion of w (default = 0.5).
c	Character string. Fill color of sector proportions (default = "lightskyblue").
s	Logical value. If FALSE (the default), sort samples alphabetically. If TRUE, sort samples by decreasing proportional abundance. Samples are sorted independently for each group.
a	Logical value. If FALSE, proportional abundance is represented by the fraction of filled sector radius to outer sector radius. If TRUE (the default), proportional abundance is represented by the fraction of filled sector area to outer sector area.
m	Numeric scalar. Maximum number of plot columns (default = 3).
...	Additional arguments passed to title.

Details

Produces a pie-chart-like proportion plot with grouping structure. Each circle represents a group, and each sector represents a sample. When a = TRUE (the default), then the proportion of each sector filled with color represents the within-sample proportional abundance. Groups are sorted alphabetically (or inherit factor level ordering) and arranged from left to right and top to bottom. When s = FALSE (the default), then samples are sorted alphabetically and arranged in a clockwise orientation (from angle zero). When s = TRUE, then samples are sorted by decreasing proportional abundance. Samples are sorted independently for each group. This plot design is specialized for visualizing proportional abundance data.

Value

No return value.

References

A manuscript describing this plot design is in preparation.

See Also

singular.proportion for singular proportion plots.

detection for grouped detection plots.

Examples

```
set.seed(1234)
n.groups<-6
n.samples<-6
data<-list(g=rep(x=LETTERS[1:n.groups],each=n.samples),
           s=rep(x=letters[1:n.samples],times=n.groups),
           p=stats::rbeta(n=n.groups*n.samples,
                          shape1=1,shape2=1))
proportion(x=data)
```

`read.fasta` *Read FASTA Files*

Description

Reads FASTA files. Supports the reading of FASTA files with sequences wrapping multiple lines.

Usage

```
read.fasta(file)
```

Arguments

`file`	A string specifying the path to a FASTA file to read.

Value

A data frame with fields for sequence names and sequences.

See Also

`write.fasta` for writing FASTA files.

`read.fastq` for reading FASTQ files.

`write.fastq` for writing FASTQ files.

Examples

```
# Get path to example FASTA file.
path_to_fasta_file<-system.file("extdata",
                                "example_query_sequences.fasta",
                                package="LocaTT",
                                mustWork=TRUE)

# Read the example FASTA file.
read.fasta(file=path_to_fasta_file)
```

`read.fastq` *Read FASTQ Files*

Description

Reads FASTQ files. Does not support the reading of FASTQ files with sequences or quality scores wrapping multiple lines.

Usage

```
read.fastq(file)
```

Arguments

`file`	A string specifying the path to a FASTQ file to read.

Value

A data frame with fields for sequence names, sequences, comments, and quality scores.

See Also

`write.fastq` for writing FASTQ files.

`read.fasta` for reading FASTA files.

`write.fasta` for writing FASTA files.

Examples

```
# Get path to example FASTQ file.
path_to_fastq_file<-system.file("extdata",
                                "example_query_sequences.fastq",
                                package="LocaTT",
                                mustWork=TRUE)

# Read the example FASTQ file.
read.fastq(file=path_to_fastq_file)
```

reverse_complement *Get the Reverse Complement of a DNA Sequence*

Description

Gets the reverse complement of a DNA sequence. Ambiguous nucleotides are supported.

Usage

```
reverse_complement(sequence)
```

Arguments

sequence	A string specifying the DNA sequence. Can contain ambiguous nucleotides.

Value

A string of the reverse complement of the DNA sequence.

Examples

```
reverse_complement(sequence="TTCTCCASCCGCGGATHTTG")
```

sector *Draw Sector Polygon*

Description

Draws sector polygon.

Usage

```
sector(s, e, r, v = 1000, ...)
```

Arguments

s	Numeric scalar of start angle (degrees).
e	Numeric scalar of end angle (degrees).
r	Numeric scalar of circle radius.
v	Numeric scalar of full-circle vertex count (default = 1000).
...	Additional arguments passed to polgyon.

Details

Draws a sector polygon given a start angle, end angle, and circle radius. The sector is drawn about the origin (*i.e.*, x = 0, y = 0). Intended for use with `template` to generate `detection` and `proportion` plots.

Value

No return value.

See Also

`circle` for plotting circle polygons.

Examples

```
template(l=1)
sector(s=0,e=45,r=1)
```

---

singular.detection *Singular Detection Plot*

Description

Generates a detection plot for a singular group.

Usage

```
singular.detection(
  x,
  r = 1,
  b = 0.025,
  v = 1000,
  w = 1,
  f = 0.5,
  c = "lightskyblue",
  t = "",
  ...
)
```

Arguments

x	A list of vectors named `"s"`, `"r"`, and `"d"`. The elements of vector `"s"` (character or numeric) specify the sample. The elements of vector `"r"` (numeric) specify the number of replicates within sample `"s"`. The elements of vector `"d"` (numeric) specify the number of replicates within sample `"s"` with detections.
r	Numeric scalar. Radius of plot circle (default = 1).

b	Numeric scalar. Plot radius buffer (proportion; default = 0.025).
v	Numeric scalar. Vertex count of plot circle (default = 1000).
w	Numeric scalar. Line width of outer circle (default = 1).
f	Numeric scalar. Line width of sectors as a proportion of w (default = 0.5).
c	Character string. Fill color of sub-sector detections (default = "lightskyblue").
t	Character string. Plot title (default = "").
...	Additional arguments passed to title.

Details

Produces a pie-chart-like detection plot without grouping structure. Each sector represents a sample, and each sub-sector represents a replicate. Filled replicates represent detections. Samples are sorted alphabetically and arranged in a clockwise orientation (from angle zero). This plot design is specialized for visualizing binary detection data.

Value

No return value.

References

A manuscript describing this plot design is in preparation.

See Also

detection for grouped detection plots.

proportion for grouped proportion plots.

Examples

```
set.seed(1234)
n.samples<-6
n.replicates<-3
data<-list(s=letters[1:n.samples],
           r=rep(x=n.replicates,times=n.samples),
           d=sample(x=0:n.replicates,size=n.samples,
                    replace=TRUE))
singular.detection(x=data)
```

`singular.proportion` *Singular Proportion Plot*

Description

Generates a proportion plot for a singular group.

Usage

```
singular.proportion(
  x,
  r = 1,
  b = 0.025,
  v = 1000,
  w = 1,
  f = 0.5,
  c = "lightskyblue",
  t = "",
  s = FALSE,
  a = TRUE,
  ...
)
```

Arguments

`x`	A list of vectors named `"s"` and `"p"`. The elements of vector `"s"` (character or numeric) specify the sample. The elements of vector `"p"` (numeric) specify the proportional abundance within sample `"s"`.
`r`	Numeric scalar. Radius of plot circle (default = `1`).
`b`	Numeric scalar. Plot radius buffer (proportion; default = `0.025`).
`v`	Numeric scalar. Vertex count of plot circle (default = `1000`).
`w`	Numeric scalar. Line width of outer circle (default = `1`).
`f`	Numeric scalar. Line width of sectors as a proportion of `w` (default = `0.5`).
`c`	Character string. Fill color of sector proportions (default = `"lightskyblue"`).
`t`	Character string. Plot title (default = `""`).
`s`	Logical value. If `FALSE` (the default), sort samples alphabetically. If `TRUE`, sort samples by decreasing proportional abundance.
`a`	Logical value. If `FALSE`, proportional abundance is represented by the fraction of filled sector radius to outer sector radius. If `TRUE` (the default), proportional abundance is represented by the fraction of filled sector area to outer sector area.
`...`	Additional arguments passed to `title`.

Details

Produces a pie-chart-like proportion plot without grouping structure. Each sector represents a sample. When a = TRUE (the default), then the proportion of each sector filled with color represents the within-sample proportional abundance. When s = FALSE (the default), then samples are sorted alphabetically and arranged in a clockwise orientation (from angle zero). When s = TRUE, then samples are sorted by decreasing proportional abundance. This plot design is specialized for visualizing proportional abundance data.

Value

No return value.

References

A manuscript describing this plot design is in preparation.

See Also

proportion for grouped proportion plots.

singular.detection for singular detection plots.

Examples

```
set.seed(1234)
n.samples<-6
data<-list(s=letters[1:n.samples],
           p=stats::rbeta(n=n.samples,
                          shape1=1,shape2=1))
singular.proportion(x=data)
```

---

substitute_wildcards *Substitute Wildcard Characters in a DNA Sequence*

Description

Substitutes wildcard characters in a DNA sequence with their associated nucleotides surrounded by square brackets. The output is useful for matching in regular expressions.

Usage

```
substitute_wildcards(sequence)
```

Arguments

sequence	A string specifying the DNA sequence containing wildcard characters.

Value

A string of the DNA sequence in which wildcard characters are replaced with their associated nucleotides surrounded by square brackets.

Examples

```
substitute_wildcards(sequence="CAADATCCGCGGSTGGAGAA")
```

`summarize_quality_scores` *Summarize Quality Scores*

Description

For each base pair position, summarizes read length, Phred quality score, and the cumulative probability that all bases were called correctly.

Usage

```
summarize_quality_scores(
  forward_files,
  reverse_files,
  n.total = 10000,
  n.each = ceiling(n.total/length(forward_files)),
  seed = NULL,
  FUN = mean,
  ...
)
```

Arguments

`forward_files`	A character vector of file paths to FASTQ files containing forward DNA sequence reads.
`reverse_files`	A character vector of file paths to FASTQ files containing reverse DNA sequence reads.
`n.total`	Numeric. The number of read pairs to randomly sample from the input FASTQ files. Ignored if `n.each` is specified. The default is `10000`.
`n.each`	Numeric. The number of read pairs to randomly sample from each pair of input FASTQ files. The default is `ceiling(n.total/length(forward_files))`.
`seed`	Numeric. The seed for randomly sampling read pairs. If `NULL` (the default), then a random seed is used.
`FUN`	A function to compute summary statistics of the quality scores. The default is `mean`.
`...`	Additional arguments passed to `FUN`.

Details

For each combination of base pair position and read direction, calculates summary statistics of read length, Phred quality score, and the cumulative probability that all bases were called correctly. The cumulative probability is calculated from the first base pair up to the current position. Quality scores are assumed to be encoded in Sanger format. Read pairs are selected by randomly sampling up to `n.each` read pairs from each pair of input FASTQ files. By default, `n.each` is derived from `n.total`, and `n.total` will be ignored if `n.each` is provided. By default, `mean` is used to compute the summary statistics, but the user may provide another summary function instead (*e.g.*, `median`). Functions which return multiple summary statistics are also supported (*e.g.*, `summary` and `quantile`). Arguments in `...` are passed to the summary function.

Value

Returns a data frame containing summary statistics of read length and quality score at each base pair position. The returned data frame contains the following fields:

- Direction: The read direction (*i.e.*, "Forward" or "Reverse").
- Position: The base pair position.
- Length: The summary statistic(s) of read lengths. If `FUN` returns multiple summary statistics, then a matrix of the summary statistics will be stored in this field, which can be accessed with `$Length`.
- Score: The summary statistic(s) of Phred quality scores. If `FUN` returns multiple summary statistics, then a matrix of the summary statistics will be stored in this field, which can be accessed with `$Score`.
- Probability: The summary statistic(s) of the cumulative probability that all bases were called correctly. If `FUN` returns multiple summary statistics, then a matrix of the summary statistics will be stored in this field, which can be accessed with `$Probability`.

See Also

`decode_quality_scores` for decoding quality scores.

Examples

```
# Get example forward FASTQ files.
forward_files<-system.file("extdata",
                           paste0("S0",1:3,"F.fastq"),
                           package="LocaTT",
                           mustWork=TRUE)

# Get example reverse FASTQ files.
reverse_files<-system.file("extdata",
                           paste0("S0",1:3,"R.fastq"),
                           package="LocaTT",
                           mustWork=TRUE)

# Summarize quality scores.
summarize_quality_scores(forward_files,reverse_files)
```

`template` *Initiate Template Plot*

Description

Initiates a blank template plot.

Usage

```
template(l, b = 0.025)
```

Arguments

`l`	Numeric scalar of axis limits (applies to both axes).
`b`	Numeric scalar to extend axis limits (see Details; default = `0.025`).

Details

Initiates a blank template plot with limits `l` and buffer `b` about the origin (*i.e.*, x = 0, y = 0). `l` is used for axis limits in both the negative and positive directions. `b` extends the limits beyond `l` by a fixed proportion (*i.e.*, `l` * (1 + `b`)). Intended for use with `circle` and `sector`.

Value

No return value.

See Also

`circle` for plotting circle polygons.

`sector` for plotting sector polygons.

Examples

```
template(l=1)
circle(r=1)
```

`trim_sequences` *Trim Target Nucleotide Sequence from DNA Sequences*

Description

Trims a target nucleotide sequence from the front or back of DNA sequences. Ambiguous nucleotides in the target nucleotide sequence are supported.

Usage

```
trim_sequences(
  sequences,
  target,
  anchor = "start",
  fixed = TRUE,
  required = TRUE,
  quality_scores
)
```

Arguments

`sequences`	A character vector of DNA sequences to trim.
`target`	A string specifying the target nucleotide sequence.
`anchor`	A string specifying whether the target nucleotide sequence should be trimmed from the start or end of the DNA sequences. Allowable values are `"start"` (the default) and `"end"`.
`fixed`	A logical value specifying whether the position of the target nucleotide sequence should be fixed at the ends of the DNA sequences. If `TRUE` (the default), then the position of the target nucleotide sequence is fixed at either the start or end of the DNA sequences, depending on the value of the anchor argument. If `FALSE`, then the target nucleotide sequence is searched for anywhere in the DNA sequences.
`required`	A logical value specifying whether trimming is required. If `TRUE` (the default), then sequences which could not be trimmed are returned as `NA`s. If `FALSE`, then untrimmed sequences are returned along with DNA sequences for which trimming was successful.
`quality_scores`	An optional character vector of DNA sequence quality scores. If supplied, these will be trimmed to their corresponding trimmed DNA sequences.

Details

For each DNA sequence, the target nucleotide sequence is searched for at either the front or back of the DNA sequence, depending on the value of the anchor argument. If the target nucleotide sequence is found, then it is removed from the DNA sequence. If the required argument is set to `TRUE`, then DNA sequences in which the target nucleotide sequence was not found will be returned as `NA`s. If the required argument is set to `FALSE`, then untrimmed DNA sequences will be returned along with DNA sequences for which trimming was successful. Ambiguous nucleotides in the

target nucleotide sequence are supported through the internal use of the substitute_wildcards function on the target nucleotide sequence, and a regular expression with a leading or ending anchor is used to search for the target nucleotide sequence in the DNA sequences. If the fixed argument is set to FALSE, then any number of characters are allowed between the start or end of the DNA sequences and the target nucleotide sequence. Trimming will fail for DNA sequences which contain ambiguous nucleotides (*e.g.*, Ns) in their target nucleotide sequence region, resulting in NAs for those sequences if the required argument is set to TRUE.

Value

If quality scores are not provided, then a character vector of trimmed DNA sequences is returned. If quality scores are provided, then a list containing two elements is returned. The first element is a character vector of trimmed DNA sequences, and the second element is a character vector of quality scores which have been trimmed to their corresponding trimmed DNA sequences.

Examples

```
trim_sequences(sequences=c("ATATAGCGCG","TGCATATACG","ATCTATCACCGC"),
               target="ATMTA",
               anchor="start",
               fixed=TRUE,
               required=TRUE,
               quality_scores=c("989!.C;F@\"","A((#-#;,2F","HD8I/+67=1>?"))
```

---

truncate_and_merge_pairs

Truncate and Merge Forward and Reverse DNA Sequence Reads

Description

Removes DNA read pairs containing ambiguous nucleotides, truncates reads by length and quality score, and merges forward and reverse reads.

Usage

```
truncate_and_merge_pairs(
  forward_files,
  reverse_files,
  output_files,
  truncation_length = NA,
  threshold.quality_score = 3,
  threshold.probability = 0.5,
  minimum_overlap = 10,
  cores = 1,
  progress = FALSE
)
```

Arguments

forward_files	A character vector of file paths to FASTQ files containing forward DNA sequence reads.
reverse_files	A character vector of file paths to FASTQ files containing reverse DNA sequence reads.
output_files	A character vector of file paths to output FASTA files.
truncation_length	Numeric. The length to truncate DNA sequences to (passed to the length argument of truncate_sequences.length). If NA (the default), then DNA sequences are not truncated by length. If a single value is supplied, then both forward and reverse reads are truncated to the same length. If two values are supplied in a numeric vector, then the first value is used to truncate the forward reads, and the second value is used to truncate the reverse reads. NA can also be supplied as either the first or second element of the numeric vector to prevent length truncation of the respective read direction while allowing the other read direction to be length truncated.
threshold.quality_score	Numeric. The Phred quality score threshold used for truncation (passed to the threshold argument of truncate_sequences.quality_score). The default is 3 (*i.e.*, each base in a truncated sequence has a greater than 50% probability of having been called correctly). If NA, then DNA sequences are not truncated by quality score threshold. If a single value is supplied, then both forward and reverse reads are truncated by the same quality score threshold. If two values are supplied in a numeric vector, then the first value is used to truncate the forward reads, and the second value is used to truncate the reverse reads. NA can also be supplied as either the first or second element of the numeric vector to prevent quality-score-threshold truncation of the respective read direction while allowing the other read direction to be quality-score-threshold truncated.
threshold.probability	Numeric. The probability threshold used for truncation (passed to the threshold argument of truncate_sequences.probability). The default is 0.5 (*i.e.*, each truncated sequence has a greater than 50% probability that all bases were called correctly). If NA, then DNA sequences are not truncated by probability threshold. If a single value is supplied, then both forward and reverse reads are truncated by the same probability threshold. If two values are supplied in a numeric vector, then the first value is used to truncate the forward reads, and the second value is used to truncate the reverse reads. NA can also be supplied as either the first or second element of the numeric vector to prevent probability-threshold truncation of the respective read direction while allowing the other read direction to be probability-threshold truncated.
minimum_overlap	Numeric. The minimum length of an overlap that must be found between the end of the forward read and the start of the reverse complement of the reverse read in order for a read pair to be merged (passed to merge_pairs). The default is 10.
cores	Numeric. If 1 (the default), then FASTQ file pairs are processed sequentially on a single core. If greater than 1, then FASTQ file pairs are processed in parallel

across the specified number of cores. Parallel processing is not supported on Windows.

progress — Logical. If TRUE, then a progress indicator is printed to the console. Ignored if cores > 1. If FALSE (the default), then no progress indicator is displayed.

Details

For each pair of input FASTQ files, removes DNA read pairs containing ambiguous nucleotides, truncates reads by length, quality score threshold, and probability threshold (in that order), and then merges forward and reverse reads. Merged reads are summarized by frequency of occurrence and written to a FASTA file. See contains_wildcards, truncate_sequences.length, truncate_sequences.quality_score, truncate_sequences.probability, and merge_pairs for methods. Quality scores are assumed to be encoded in Sanger format. Forward and reverse reads can be truncated by different thresholds (see truncation_length, threshold.quality_score, and threshold.probability arguments).

Multicore parallel processing is supported on Mac and Linux operating systems (not available on Windows). When cores > 1 (parallel processing enabled), warnings and errors are printed to the console in addition to being invisibly returned as a list (see the return value section), and errors produced while processing a pair of FASTQ files will not interrupt the processing of other FASTQ file pairs. When cores = 1, FASTQ file pairs are processed sequentially on a single core, and errors will prevent the processing of subsequent FASTQ file pairs (but warnings will not).

Value

If cores = 1, then no return value. Writes a FASTA file for each pair of input FASTQ files with DNA sequence counts stored in the header lines. If cores > 1, then also invisibly returns a list where each element contains warning or error messages associated with processing each pair of input FASTQ files. A NULL value in the returned list means that no warnings or errors were generated from processing the respective pair of FASTQ files.

References

A manuscript describing these methods is in preparation.

See Also

contains_wildcards for detecting ambiguous nucleotides in DNA sequences.

truncate_sequences.length for truncating DNA sequences to a specified length.

truncate_sequences.quality_score for truncating DNA sequences by Phred quality score.

truncate_sequences.probability for truncating DNA sequences by cumulative probability that all bases were called correctly.

merge_pairs for merging forward and reverse DNA sequence reads.

filter_sequences for filtering merged read pairs by PCR replicate.

Examples

```
# Get example forward FASTQ files.
forward_files<-system.file("extdata",
                           paste0("S0",1:3,"F.fastq"),
                           package="LocaTT",
                           mustWork=TRUE)

# Get example reverse FASTQ files.
reverse_files<-system.file("extdata",
                           paste0("S0",1:3,"R.fastq"),
                           package="LocaTT",
                           mustWork=TRUE)

# Create paths for temporary output files.
output_files<-tempfile(pattern=paste0("O",1:3),fileext=".fasta")

# Truncate and merge pairs.
truncate_and_merge_pairs(forward_files=forward_files,
                         reverse_files=reverse_files,
                         output_files=output_files)
```

---

truncate_sequences.length

Truncate DNA Sequences to Specified Length

Description

Truncates DNA sequences to a specified length.

Usage

```
truncate_sequences.length(sequences, length, quality_scores)
```

Arguments

sequences	A character vector of DNA sequences to truncate.
length	Numeric. The length to truncate DNA sequences to.
quality_scores	An optional character vector of DNA sequence quality scores. If supplied, these will be truncated to their corresponding truncated DNA sequences.

Value

If quality scores are not provided, then a character vector of truncated DNA sequences is returned. If quality scores are provided, then a list containing two elements is returned. The first element is a character vector of truncated DNA sequences, and the second element is a character vector of quality scores which have been truncated to their corresponding truncated DNA sequences.

See Also

truncate_sequences.quality_score for truncating DNA sequences by Phred quality score.

truncate_sequences.probability for truncating DNA sequences by cumulative probability that all bases were called correctly.

truncate_and_merge_pairs for truncating and merging forward and reverse DNA sequence reads.

Examples

```
truncate_sequences.length(sequences=c("ATATAGCGCG","TGCCGATATA","ATCTATCACCGC"),
                          length=5,
                          quality_scores=c("989!.C;F@\"","A((#-#;,2F","HD8I/+67=1>?"))
```

---

truncate_sequences.probability

Truncate DNA Sequences at Specified Probability that All Bases were Called Correctly

---

Description

Calculates the cumulative probability that all bases were called correctly along each DNA sequence and truncates the DNA sequence immediately prior to the first occurrence of a probability being equal to or less than a specified value.

Usage

```
truncate_sequences.probability(sequences, quality_scores, threshold = 0.5)
```

Arguments

sequences	A character vector of DNA sequences to truncate.
quality_scores	A character vector of DNA sequence quality scores encoded in Sanger format.
threshold	Numeric. The probability threshold used for truncation. The default is 0.5 (*i.e.*, each truncated sequence has a greater than 50% probability that all bases were called correctly).

Value

A list containing two elements. The first element is a character vector of truncated DNA sequences, and the second element is a character vector of quality scores which have been truncated to their corresponding truncated DNA sequences.

See Also

truncate_sequences.length for truncating DNA sequences to a specified length.

truncate_sequences.quality_score for truncating DNA sequences by Phred quality score.

truncate_and_merge_pairs for truncating and merging forward and reverse DNA sequence reads.

Examples

```
truncate_sequences.probability(sequences=c("ATATAGCGCG","TGCCGATATA","ATCTATCACCGC"),
                               quality_scores=c("989!.C;F@\"","A((#-#;,2F","HD8I/+67=1>?"),
                                  threshold=0.5)
```

---

truncate_sequences.quality_score

Truncate DNA Sequences at Specified Quality Score

Description

Truncates DNA sequences immediately prior to the first occurrence of a Phred quality score being equal to or less than a specified value.

Usage

```
truncate_sequences.quality_score(sequences, quality_scores, threshold = 3)
```

Arguments

sequences	A character vector of DNA sequences to truncate.
quality_scores	A character vector of DNA sequence quality scores encoded in Sanger format.
threshold	Numeric. The Phred quality score threshold used for truncation. The default is 3 (*i.e.*, each base in a truncated sequence has a greater than 50% probability of having been called correctly).

Value

A list containing two elements. The first element is a character vector of truncated DNA sequences, and the second element is a character vector of quality scores which have been truncated to their corresponding truncated DNA sequences.

See Also

truncate_sequences.length for truncating DNA sequences to a specified length.

truncate_sequences.probability for truncating DNA sequences by cumulative probability that all bases were called correctly.

truncate_and_merge_pairs for truncating and merging forward and reverse DNA sequence reads.

Examples

```
truncate_sequences.quality_score(sequences=c("ATATAGCGCG","TGCCGATATA","ATCTATCACCGC"),
                                 quality_scores=c("989!.C;F@\"","A((#-#;,2F","HD8I/+67=1>?"),
                                      threshold=3)
```

`write.fasta` *Write FASTA Files*

Description

Writes FASTA files.

Usage

```
write.fasta(names, sequences, file)
```

Arguments

`names`	A character vector of sequence names.
`sequences`	A character vector of sequences.
`file`	A string specifying the path to a FASTA file to write.

Value

No return value. Writes a FASTA file.

See Also

`read.fasta` for reading FASTA files.

`write.fastq` for writing FASTQ files.

`read.fastq` for reading FASTQ files.

Examples

```
# Get path to example sequences CSV file.
path_to_CSV_file<-system.file("extdata",
                              "example_query_sequences.csv",
                              package="LocaTT",
                              mustWork=TRUE)

# Read the example sequences CSV file.
df<-read.csv(file=path_to_CSV_file,stringsAsFactors=FALSE)

# Create a temporary file path for the FASTA file to write.
path_to_FASTA_file<-tempfile(fileext=".fasta")
```

```
# Write the example sequences as a FASTA file.
write.fasta(names=df$Name,
            sequences=df$Sequence,
            file=path_to_FASTA_file)
```

`write.fastq` *Write FASTQ Files*

Description

Writes FASTQ files.

Usage

```
write.fastq(names, sequences, quality_scores, file, comments)
```

Arguments

`names`	A character vector of sequence names.
`sequences`	A character vector of sequences.
`quality_scores`	A character vector of quality scores.
`file`	A string specifying the path to a FASTQ file to write.
`comments`	An optional character vector of sequence comments.

Value

No return value. Writes a FASTQ file.

See Also

`read.fastq` for reading FASTQ files.

`write.fasta` for writing FASTA files.

`read.fasta` for reading FASTA files.

Examples

```
# Get path to example sequences CSV file.
path_to_CSV_file<-system.file("extdata",
                              "example_query_sequences.csv",
                              package="LocaTT",
                              mustWork=TRUE)

# Read the example sequences CSV file.
df<-read.csv(file=path_to_CSV_file,stringsAsFactors=FALSE)

# Create a temporary file path for the FASTQ file to write.
```

```
path_to_FASTQ_file<-tempfile(fileext=".fastq")

# Write the example sequences as a FASTQ file.
write.fastq(names=df$Name,
            sequences=df$Sequence,
            quality_scores=df$Quality_score,
            file=path_to_FASTQ_file,
            comments=df$Comment)
```

Index