Package 'LocaTT'

October 30, 2023

Title Geographically-Conscious Taxonomic Assignment for Metabarcoding

Version 1.1.1

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/Local_Taxa_Tool_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 7.2.3

Imports utils, stats, taxize

NeedsCompilation no

Author Kenen Goodwin [aut, cre] (<https://orcid.org/0000-0002-9219-7693>), Taal Levi [aut]

Maintainer Kenen Goodwin <urodelan@gmail.com>

Repository CRAN

Date/Publication 2023-10-30 13:00:08 UTC

R topics documented:

binomial_test 2
blast_command_found 3
blast_version 3
contains_wildcards 4
decode_quality_scores 4
format_reference_database 5
get_consensus_taxonomy 7
get_taxonomic_level 8
get_taxonomies.IUCN 9

get_taxonomies.species_binomials 10
isolate_amplicon 11
local_taxa_tool 12
merge_pairs 16
read.fasta 17
read.fastq 17
reverse_complement 18
substitute_wildcards 19
trim_sequences 19
truncate_sequences.length 21
truncate_sequences.probability 22
truncate_sequences.quality_score 23
write.fasta 24
write.fastq 25

Index **26**

`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()
```

`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"))
```

`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@\"")
```

`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_reference_database,
  path_to_output_BLAST_database,
  input_reference_database_source = "MIDORI",
  path_to_taxonomy_edits = NA,
  path_to_sequence_edits = NA,
  path_to_list_of_local_taxa_to_subset = NA,
  makeblastdb_command = "makeblastdb"
)
```

Arguments

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

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

`input_reference_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 superkingdom 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_list_of_local_taxa_to_subset
: 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.

Value

No return value. Writes formatted BLAST database files.

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_reference_database=path_to_input_file,
                          path_to_output_BLAST_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.

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.

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_IUCN_taxonomies,
  path_to_IUCN_common_names,
  path_to_output_local_taxa_list,
  domain_name = "Eukaryota",
  path_to_taxonomy_edits = NA
)
```

Arguments

`path_to_IUCN_taxonomies`
: String specifying path to input IUCN Red List taxonomy.csv file.

`path_to_IUCN_common_names`
: String specifying path to input IUCN Red List common_names.csv file.

`path_to_output_local_taxa_list`
: String specifying path to output species list (in CSV format) with formatted taxonomies.

`domain_name`
: 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).

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.

Examples

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

# Get path to example common names CSV file.
path_to_common_names_file<-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_IUCN_taxonomies=path_to_taxonomy_file,
                    path_to_IUCN_common_names=path_to_common_names_file,
                    path_to_output_local_taxa_list=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.

Usage

```
get_taxonomies.species_binomials(
  path_to_input_species_binomials,
  path_to_output_local_taxa_list,
  path_to_taxonomy_edits = NA,
  print_taxize_queries = TRUE
)
```

Arguments

`path_to_input_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_local_taxa_list`
: 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_taxize_queries
: Logical. Whether taxa queries should be printed. The default is TRUE.

Value

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

See Also

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

Examples

```
# Get path to example input species binomials CSV file.
path_to_input_file<-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_input_species_binomials=path_to_input_file,
                                 path_to_output_local_taxa_list=path_to_output_file,
                                 print_taxize_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 nucleotodies.

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 NAs for those sequences.

Value

A character vector of DNA sequences trimmed to the amplicon region. NAs 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_sequences_to_classify,
  path_to_BLAST_database,
  path_to_output_file,
  path_to_list_of_local_taxa = 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_sequences_to_classify`
: 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_list_of_local_taxa`
: 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.

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.

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 reference database FASTA file.
path_to_reference_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_CSV_file<-tempfile(fileext=".csv")

# Run the local taxa tool.
local_taxa_tool(path_to_sequences_to_classify=path_to_query_sequences,
                path_to_BLAST_database=path_to_reference_database,
                path_to_output_file=path_to_output_CSV_file,
                path_to_list_of_local_taxa=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.

Examples

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

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")
```

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")
```

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 NAs. 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 NAs. 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_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.

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 trimmed 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.

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 trimmed 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.

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

binomial_test, 2
blast_command_found, 3
blast_version, 3

contains_wildcards, 4

decode_quality_scores, 4

format_reference_database, 5, *14*

get_consensus_taxonomy, 7
get_taxonomic_level, 8
get_taxonomies.IUCN, 9, *11*, *14*
get_taxonomies.species_binomials, *9*, 10, *14*

isolate_amplicon, 11

local_taxa_tool, *5*, *9*, 12

merge_pairs, 16

pbinom, *2*

read.fasta, 17, *18*, *24, 25*
read.fastq, *17*, 17, *24, 25*
reverse_complement, *12*, *16*, 18

stats, *2*
substitute_wildcards, *12*, 19, *20*

trim_sequences, 19
truncate_sequences.length, 21, *22, 23*
truncate_sequences.probability, *21*, 22, *23*
truncate_sequences.quality_score, *21*, *22*, 23

write.fasta, *17, 18*, 24, *25*
write.fastq, *17, 18*, *24*, 25