Package 'pwalign'

November 12, 2026

Title Perform pairwise sequence alignments

Description The two main functions in the package are pairwiseAlignment() and stringDist(). The former solves (Needleman-Wunsch) global alignment, (Smith-Waterman) local alignment, and (ends-free) overlap alignment problems. The latter computes the Levenshtein edit distance or pairwise alignment score matrix for a set of strings.

biocViews Alignment, SequenceMatching, Sequencing, Genetics

URL https://bioconductor.org/packages/pwalign

BugReports https://github.com/Bioconductor/pwalign/issues

Version 1.8.0

License Artistic-2.0

Encoding UTF-8

Depends BiocGenerics, S4Vectors, IRanges, Biostrings (>= 2.71.5)

Imports methods, utils

LinkingTo S4Vectors, IRanges, XVector, Biostrings

Enhances Rmpi

Suggests RUnit

Collate 00datacache.R utils.R InDel-class.R AlignedXStringSet-class.R PairwiseAlignments-class.R PairwiseAlignmentsSingleSubject-class.R PairwiseAlignments-io.R align-utils.R pid.R substitution_matrices.R pairwiseAlignment.R stringDist.R zzz.R

git_url https://git.bioconductor.org/packages/pwalign

git_branch RELEASE_3_23

git_last_commit a6c6c74

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Patrick Aboyoun [aut],
Robert Gentleman [aut],
Hervé Pagès [cre] (ORCID: <https://orcid.org/0009-0002-8272-4522>)

Maintainer Hervé Pagès <hpages.on.github@gmail.com>

Contents

align-utils 2
AlignedXStringSet-class 4
InDel-class 6
pairwiseAlignment 7
PairwiseAlignments-class 10
PairwiseAlignments-io 14
phiX174Phage 16
pid 18
predefined_scoring_matrices 19
stringDist 20
substitution_matrices 22

Index **25**

`align-utils` *Utility functions related to sequence alignment*

Description

A variety of different functions used to deal with sequence alignments.

Usage

```
nedit(x) # also nmatch and nmismatch

mismatchTable(x, shiftLeft=0L, shiftRight=0L, ...)
mismatchSummary(x, ...)

## S4 method for signature 'AlignedXStringSet0'
coverage(x, shift=0L, width=NULL, weight=1L)

## S4 method for signature 'PairwiseAlignmentsSingleSubject'
coverage(x, shift=0L, width=NULL, weight=1L)

compareStrings(pattern, subject)

## S4 method for signature 'PairwiseAlignmentsSingleSubject'
consensusMatrix(x,
                as.prob=FALSE, shift=0L, width=NULL,
                baseOnly=FALSE, gapCode="-", endgapCode="-")
```

Arguments

x	A character vector or matrix, XStringSet, XStringViews, PairwiseAlignments, or list of FASTA records containing the equal-length strings.
shiftLeft, shiftRight	Non-positive and non-negative integers respectively that specify how many preceding and succeeding characters to and from the mismatch position to include in the mismatch substrings.
...	Further arguments to be passed to or from other methods.
shift, width	See ?coverage.
weight	An integer vector specifying how much each element in x counts.
pattern, subject	The strings to compare. Can be of type character, XString, XStringSet, AlignedXStringSet, or, in the case of pattern, PairwiseAlignments. If the first argument of compareStrings() (pattern) is a PairwiseAlignments object, then the second argument (subject) must be missing. In this case compareStrings(x) is equivalent to compareStrings(pattern(x), subject(x)).
as.prob	If TRUE then probabilities are reported, otherwise counts (the default).
baseOnly	TRUE or FALSE. If TRUE, the returned vector only contains frequencies for the letters in the "base" alphabet i.e. "A", "C", "G", "T" if x is a "DNA input", and "A", "C", "G", "U" if x is "RNA input". When x is a BString object (or an XStringViews object with a BString subject, or a BStringSet object), then the baseOnly argument is ignored.
gapCode, endgapCode	The codes in the appropriate alphabet to use for the internal and end gaps.

Value

nedit(): An integer vector of the same length as the input PairwiseAlignments object reporting the number of edits (i.e. nb of mismatches + nb of indels) for each alignment.

mismatchTable(): A data.frame containing the positions and substrings of the mismatches for the AlignedXStringSet or PairwiseAlignments object.

mismatchSummary(): A list of data.frame objects containing counts and frequencies of the mismatches for the AlignedXStringSet or PairwiseAlignmentsSingleSubject object.

compareStrings(): Combines two equal-length strings that are assumed to be aligned into a single character string containing that replaces mismatches with "?", insertions with "+", and deletions with "-".

Author(s)

P. Aboyoun

See Also

pairwiseAlignment, consensusMatrix, XString-class, XStringSet-class, XStringViews-class, AlignedXStringSet-class, PairwiseAlignments-class, match-utils

Examples

```
## Compare two globally aligned strings
string1 <- "ACTTCACCAGCTCCCTGGCGGTAAGTTGATC---AAAGG---AAACGCAAAGTTTTCAAG"
string2 <- "GTTTCACTACTTCCTTTCGGGTAAGTAAATATATAAATATATAAAAATATAATTTTCATC"
compareStrings(string1, string2)

## Create a consensus matrix
nw1 <- pairwiseAlignment(
         AAStringSet(c("HLDNLKGTF", "HVDDMPNAL")), AAString("SMDDTEKMSMKL"),
         substitutionMatrix = "BLOSUM50", gapOpening = 3, gapExtension = 1)

consensusMatrix(nw1)

## Examine the consensus between the bacteriophage phi X174 genomes
data(phiX174Phage)
phageConsmat <- consensusMatrix(phiX174Phage, baseOnly = TRUE)
phageDiffs <- which(apply(phageConsmat, 2, max) < length(phiX174Phage))
phageDiffs
phageConsmat[,phageDiffs]
```

AlignedXStringSet-class

AlignedXStringSet and QualityAlignedXStringSet objects

Description

The AlignedXStringSet and QualityAlignedXStringSet classes are containers for storing an aligned XStringSet.

Details

Before we define the notion of alignment, we introduce the notion of "filled-with-gaps subsequence". A "filled-with-gaps subsequence" of a string string1 is obtained by inserting 0 or any number of gaps in a subsequence of s1. For example L-A–ND and A–N-D are "filled-with-gaps subsequences" of LAND. An alignment between two strings string1 and string2 results in two strings (align1 and align2) that have the same length and are "filled-with-gaps subsequences" of string1 and string2.

For example, this is an alignment between LAND and LEAVES:

```
L-A
LEA
```

An alignment can be seen as a compact representation of one set of basic operations that transforms string1 into align1. There are 3 different kinds of basic operations: "insertions" (gaps in align1), "deletions" (gaps in align2), "replacements". The above alignment represents the following basic operations:

```
insert E at pos 2
insert V at pos 4
insert E at pos 5
replace by S at pos 6 (N is replaced by S)
delete at pos 7 (D is deleted)
```

Note that "insert X at pos i" means that all letters at a position >= i are moved 1 place to the right before X is actually inserted.

There are many possible alignments between two given strings string1 and string2 and a common problem is to find the one (or those ones) with the highest score, i.e. with the lower total cost in terms of basic operations.

Accessor methods

In the code snippets below, x is a `AlignedXStringSet` or `QualityAlignedXStringSet` object.

`unaligned(x)`: The original string.

`aligned(x, degap = FALSE)`: If `degap = FALSE`, the "filled-with-gaps subsequence" representing the aligned substring. If `degap = TRUE`, the "gap-less subsequence" representing the aligned substring.

`ranges(x)`: The bounds of the aligned substring.

`start(x)`: The start of the aligned substring.

`end(x)`: The end of the aligned substring.

`width(x)`: The width of the aligned substring, ignoring gaps.

`indel(x)`: The positions, in the form of an `IRanges` object, of the insertions or deletions (depending on what x represents).

`nindel(x)`: A two-column matrix containing the length and sum of the widths for each of the elements returned by `indel`.

`length(x)`: The length of the `aligned(x)`.

`nchar(x)`: The nchar of the `aligned(x)`.

`alphabet(x)`: Equivalent to `alphabet(unaligned(x))`.

`as.character(x)`: Converts `aligned(x)` to a character vector.

`toString(x)`: Equivalent to `toString(as.character(x))`.

Subsetting methods

`x[i]`: Returns a new `AlignedXStringSet` or `QualityAlignedXStringSet` object made of the selected elements.

`rep(x, times)`: Returns a new `AlignedXStringSet` or `QualityAlignedXStringSet` object made of the repeated elements.

Author(s)

P. Aboyoun

See Also

pairwiseAlignment, PairwiseAlignments-class, XStringSet-class

Examples

```
pattern <- AAString("LAND")
subject <- AAString("LEAVES")
pa1 <- pairwiseAlignment(pattern, subject, substitutionMatrix="BLOSUM50",
                         gapOpening=3, gapExtension=1)

alignedPattern <- pattern(pa1)
class(alignedPattern)  # AlignedXStringSet object

unaligned(alignedPattern)
aligned(alignedPattern)
as.character(alignedPattern)
nchar(alignedPattern)
```

InDel-class *InDel objects*

Description

The InDel class is a container for storing insertion and deletion information.

Details

This is a generic class that stores any insertion and deletion information.

Accessor methods

In the code snippets below, x is a InDel object.

insertion(x): The insertion information.

deletion(x): The deletion information.

Author(s)

P. Aboyoun

See Also

pairwiseAlignment, PairwiseAlignments-class

Examples

```
pa <- PairwiseAlignments("-PA--W-HEAE", "HEAGAWGHE-E")
pa_indel <- indel(pa)  # an InDel object
insertion(pa_indel)
deletion(pa_indel)
```

pairwiseAlignment *Optimal Pairwise Alignment*

Description

Solves (Needleman-Wunsch) global alignment, (Smith-Waterman) local alignment, and (ends-free) overlap alignment problems.

Usage

```
pairwiseAlignment(pattern, subject, ...)

## S4 method for signature 'ANY,ANY'
pairwiseAlignment(pattern, subject,
                  patternQuality=PhredQuality(22L),
                  subjectQuality=PhredQuality(22L),
                  type="global",
                  substitutionMatrix=NULL, fuzzyMatrix=NULL,
                  gapOpening=10, gapExtension=4,
                  scoreOnly=FALSE)

## S4 method for signature 'QualityScaledXStringSet,QualityScaledXStringSet'
pairwiseAlignment(pattern, subject,
                  type="global",
                  substitutionMatrix=NULL, fuzzyMatrix=NULL,
                  gapOpening=10, gapExtension=4,
                  scoreOnly=FALSE)
```

Arguments

pattern	a character vector or XStringSet derivative of any length, or an XString derivative.
subject	a character vector or XStringSet derivative of length 1 or length(pattern), or an XString derivative.
patternQuality, subjectQuality	objects of class XStringQuality representing the respective quality scores for pattern and subject that are used in a quality-based method for generating a substitution matrix. These two arguments are ignored if !is.null(substitutionMatrix) or if its respective string set (pattern, subject) is of class QualityScaledXStringSet.
type	type of alignment. One of "global", "local", "overlap", "global-local", and "local-global" where "global" = align whole strings with end gap penalties, "local" = align string fragments, "overlap" = align whole strings without end gap penalties, "global-local" = align whole strings in pattern with consecutive subsequence of subject, "local-global" = align consecutive subsequence of pattern with whole strings in subject.

`substitutionMatrix`
: substitution matrix representing the fixed substitution scores for an alignment. It cannot be used in conjunction with `patternQuality` and `subjectQuality` arguments.

`fuzzyMatrix`
: fuzzy match matrix for quality-based alignments. It takes values between 0 and 1; where 0 is an unambiguous mismatch, 1 is an unambiguous match, and values in between represent a fraction of "matchiness". (See details section below.)

`gapOpening`
: the cost for opening a gap in the alignment.

`gapExtension`
: the incremental cost incurred along the length of the gap in the alignment.

`scoreOnly`
: logical to denote whether or not to return just the scores of the optimal pairwise alignment.

`...`
: optional arguments to generic function to support additional methods.

Details

Quality-based alignments are based on the paper the Bioinformatics article by Ketil Malde listed in the Reference section below. Let ϵ_i be the probability of an error in the base read. For `"Phred"` quality measures Q in $[0, 99]$, these error probabilities are given by $\epsilon_i = 10^{-Q/10}$. For `"Solexa"` quality measures Q in $[-5, 99]$, they are given by $\epsilon_i = 1 - 1/(1 + 10^{-Q/10})$. Assuming independence within and between base reads, the combined error probability of a mismatch when the underlying bases do match is $\epsilon_c = \epsilon_1 + \epsilon_2 - (n/(n-1)) * \epsilon_1 * \epsilon_2$, where n is the number of letters in the underlying alphabet (i.e. $n = 4$ for DNA input, $n = 20$ for amino acid input, otherwise n is the number of distinct letters in the input). Using ϵ_c, the substitution score is given by $b * \log_2(\gamma_{x,y} * (1 - \epsilon_c) * n + (1 - \gamma_{x,y}) * \epsilon_c * (n/(n-1)))$, where b is the bit-scaling for the scoring and $\gamma_{x,y}$ is the probability that characters x and y represents the same underlying information (e.g. using IUPAC, $\gamma_{A,A} = 1$ and $\gamma_{A,N} = 1/4$. In the arguments listed above `fuzzyMatch` represents $\gamma_{x,y}$ and `patternQuality` and `subjectQuality` represents ϵ_1 and ϵ_2 respectively.

If `scoreOnly == FALSE`, a pairwise alignment with the maximum alignment score is returned. If more than one pairwise alignment produces the maximum alignment score, then the alignment with the smallest initial deletion whose mismatches occur before its insertions and deletions is chosen. For example, if `pattern = "AGTA"` and `subject = "AACTAACTA"`, then the alignment `pattern: [1] AG-TA; subject: [1] AACTA` is chosen over `pattern: [1] A-GTA; subject: [1] AACTA` or `pattern: [1] AG-TA; subject: [5] AACTA` if they all achieve the maximum alignment score.

Value

If `scoreOnly == FALSE` (the default), the function returns a `PairwiseAlignmentsSingleSubject` object (if a single subject was supplied) or a `PairwiseAlignments` object (if more than one subject was supplied). In both cases, the returned object contains N *optimal pairwise alignments* where N is the number of supplied patterns, that is, N = `length(pattern)` if `pattern` is a character vector or `XStringSet` derivative, or N = 1 if it's an `XString` derivative. If more than one subject was supplied, the alignments in the returned `PairwiseAlignments` object are obtained by aligning `pattern[[1]]` to `subject[[1]]`, `pattern[[2]]` to `subject[[2]]`, `pattern[[3]]` to `subject[[3]]`, etc...

If `scoreOnly == TRUE`, a numeric vector containing the scores for the N *optimal pairwise alignments* is returned.

Note

Use matchPattern or vmatchPattern if you need to find all the occurrences (eventually with indels) of a given pattern in a reference sequence or set of sequences.

Use matchPDict if you need to match a (big) set of patterns against a reference sequence.

Author(s)

P. Aboyoun

References

R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological Sequence Analysis, Cambridge UP 1998, sec 2.3.

B. Haubold, T. Wiehe, Introduction to Computational Biology, Birkhauser Verlag 2006, Chapter 2.

K. Malde, The effect of sequence quality on sequence alignment, Bioinformatics 2008 24(7):897-900.

See Also

writePairwiseAlignments, stringDist, PairwiseAlignments-class, XStringQuality-class, substitution_matrices, matchPattern

Examples

```
## Nucleotide global, local, and overlap alignments
s1 <-
  DNAString("ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAACGCAAAGTTTTCAAG")
s2 <-
  DNAString("GTTTCACTACTTCCTTTCGGGTAAGTAAATATATAAATATATAAAAATATAATTTTCATC")

# First use a fixed substitution matrix
mat <- nucleotideSubstitutionMatrix(match = 1, mismatch = -3, baseOnly = TRUE)
globalAlign <-
  pairwiseAlignment(s1, s2, substitutionMatrix = mat,
                    gapOpening = 5, gapExtension = 2)
localAlign <-
  pairwiseAlignment(s1, s2, type = "local", substitutionMatrix = mat,
                    gapOpening = 5, gapExtension = 2)
overlapAlign <-
  pairwiseAlignment(s1, s2, type = "overlap", substitutionMatrix = mat,
                    gapOpening = 5, gapExtension = 2)

# Then use quality-based method for generating a substitution matrix
pairwiseAlignment(s1, s2,
                  patternQuality = SolexaQuality(rep(c(22L, 12L), times = c(36, 18))),
                  subjectQuality = SolexaQuality(rep(c(22L, 12L), times = c(40, 20))),
                  scoreOnly = TRUE)

# Now assume can't distinguish between C/T and G/A
pairwiseAlignment(s1, s2,
```

```
                  patternQuality = SolexaQuality(rep(c(22L, 12L), times = c(36, 18))),
                  subjectQuality = SolexaQuality(rep(c(22L, 12L), times = c(40, 20))),
                  type = "local")
mapping <- diag(4)
dimnames(mapping) <- list(DNA_BASES, DNA_BASES)
mapping["C", "T"] <- mapping["T", "C"] <- 1
mapping["G", "A"] <- mapping["A", "G"] <- 1
pairwiseAlignment(s1, s2,
                  patternQuality = SolexaQuality(rep(c(22L, 12L), times = c(36, 18))),
                  subjectQuality = SolexaQuality(rep(c(22L, 12L), times = c(40, 20))),
                  fuzzyMatrix = mapping,
                  type = "local")

## Amino acid global alignment
pairwiseAlignment(AAString("PAWHEAE"), AAString("HEAGAWGHEE"),
                  substitutionMatrix = "BLOSUM50",
                  gapOpening = 0, gapExtension = 8)
```

PairwiseAlignments-class *PairwiseAlignments, PairwiseAlignmentsSingleSubject, and PairwiseAlignmentsSingleSubjectSummary objects*

Description

The PairwiseAlignments class is a container for storing a set of pairwise alignments.

The PairwiseAlignmentsSingleSubject class is a container for storing a set of pairwise alignments with a single subject.

The PairwiseAlignmentsSingleSubjectSummary class is a container for storing the summary of a set of pairwise alignments.

Usage

```
## Constructors:
## When subject is missing, pattern must be of length 2
## S4 method for signature 'XString,XString'
PairwiseAlignments(pattern, subject,
  type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = 1)
## S4 method for signature 'XStringSet,missing'
PairwiseAlignments(pattern, subject,
  type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = 1)
## S4 method for signature 'character,character'
PairwiseAlignments(pattern, subject,
  type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = 1,
  baseClass = "BString")
## S4 method for signature 'character,missing'
PairwiseAlignments(pattern, subject,
  type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = 1,
  baseClass = "BString")
```

Arguments

pattern	a character vector of length 1 or 2, an XString, or an XStringSet object of length 1 or 2.
subject	a character vector of length 1 or an XString object.
type	type of alignment. One of "global", "local", "overlap", "global-local", and "local-global" where "global" = align whole strings with end gap penalties, "local" = align string fragments, "overlap" = align whole strings without end gap penalties, "global-local" = align whole strings in pattern with consecutive subsequence of subject, "local-global" = align consecutive subsequence of pattern with whole strings in subject.
substitutionMatrix	substitution matrix for the alignment. If NULL, the diagonal values and off-diagonal values are set to 0 and 1 respectively.
gapOpening	the cost for opening a gap in the alignment.
gapExtension	the incremental cost incurred along the length of the gap in the alignment.
baseClass	the base XString class to use in the alignment.

Details

Before we define the notion of alignment, we introduce the notion of "filled-with-gaps subsequence". A "filled-with-gaps subsequence" of a string string1 is obtained by inserting 0 or any number of gaps in a subsequence of s1. For example L-A–ND and A–N-D are "filled-with-gaps subsequences" of LAND. An alignment between two strings string1 and string2 results in two strings (align1 and align2) that have the same length and are "filled-with-gaps subsequences" of string1 and string2.

For example, this is an alignment between LAND and LEAVES:

```
L-A
LEA
```

An alignment can be seen as a compact representation of one set of basic operations that transforms string1 into align1. There are 3 different kinds of basic operations: "insertions" (gaps in align1), "deletions" (gaps in align2), "replacements". The above alignment represents the following basic operations:

```
insert E at pos 2
insert V at pos 4
insert E at pos 5
replace by S at pos 6 (N is replaced by S)
delete at pos 7 (D is deleted)
```

Note that "insert X at pos i" means that all letters at a position >= i are moved 1 place to the right before X is actually inserted.

There are many possible alignments between two given strings string1 and string2 and a common problem is to find the one (or those ones) with the highest score, i.e. with the lower total cost in terms of basic operations.

Object extraction methods

In the code snippets below, x is a PairwiseAlignments object, except otherwise noted.

`alignedPattern(x)`, `alignedSubject(x)`: Extract the aligned patterns or subjects as an XStringSet object. The 2 objects returned by `alignedPattern(x)` and `alignedSubject(x)` are guaranteed to have the same shape (i.e. same `length()` and `width()`).

`pattern(x)`, `subject(x)`: Extract the aligned patterns or subjects as an AlignedXStringSet0 object.

`summary(object, ...)`: Generates a summary for the PairwiseAlignments object.

General information methods

In the code snippets below, x is a PairwiseAlignments object, except otherwise noted.

`alphabet(x)`: Equivalent to `alphabet(unaligned(subject(x)))`.

`length(x)`: The common length of `alignedPattern(x)` and `alignedSubject(x)`. There is a method for PairwiseAlignmentsSingleSubjectSummary as well.

`type(x)`: The type of the alignment ("global", "local", "overlap", "global-local", or "local-global"). There is a method for PairwiseAlignmentsSingleSubjectSummary as well.

Aligned sequence methods

In the code snippets below, x is a PairwiseAlignmentsSingleSubject object, except otherwise noted.

`aligned(x, degap = FALSE, gapCode="-", endgapCode="-")`: If degap = FALSE, "align" the alignments by returning an XStringSet object containing the aligned patterns without insertions. If degap = TRUE, returns `aligned(pattern(x), degap=TRUE)`. The gapCode and endgapCode arguments denote the code in the appropriate alphabet to use for the internal and end gaps.

`as.character(x)`: Equivalent to `as.character(alignedPattern(x))`.

`as.matrix(x)`: Returns an "exploded" character matrix representation of `aligned(x)`.

`toString(x)`: Equivalent to `toString(as.character(x))`.

Subject position methods

In the code snippets below, x is a PairwiseAlignmentsSingleSubject object, except otherwise noted.

`consensusMatrix(x, as.prob=FALSE, baseOnly=FALSE, gapCode="-", endgapCode="-")`: See 'consensusMatrix' for more information.

`consensusString(x)`: See 'consensusString' for more information.

`coverage(x, shift=0L, width=NULL, weight=1L)`: See 'coverage,PairwiseAlignmentsSingleSubject-method' for more information.

`Views(subject, start=NULL, end=NULL, width=NULL, names=NULL)`: The XStringViews object that represents the pairwise alignments along `unaligned(subject(subject))`. The start and end arguments must be either NULL/NA or an integer vector of length 1 that denotes the offset from `start(subject(subject))`.

Numeric summary methods

In the code snippets below, x is a PairwiseAlignments object, except otherwise noted.

nchar(x): The nchar of the aligned(pattern(x)) and aligned(subject(x)). There is a method for PairwiseAlignmentsSingleSubjectSummary as well.

insertion(x): An CompressedIRangesList object containing the locations of the insertions from the perspective of the pattern.

deletion(x): An CompressedIRangesList object containing the locations of the deletions from the perspective of the pattern.

indel(x): An InDel object containing the locations of the insertions and deletions from the perspective of the pattern.

nindel(x): An InDel object containing the number of insertions and deletions.

score(x): The score of the alignment. There is a method for PairwiseAlignmentsSingleSubjectSummary as well.

Subsetting methods

x[i]: Returns a new PairwiseAlignments object made of the selected elements.

rep(x, times): Returns a new PairwiseAlignments object made of the repeated elements.

Author(s)

P. Aboyoun

See Also

pairwiseAlignment, writePairwiseAlignments, AlignedXStringSet-class, XString-class, XStringViews-class, align-utils, pid

Examples

```
PairwiseAlignments("-PA--W-HEAE", "HEAGAWGHE-E")

pattern <- AAStringSet(c("HLDNLKGTF", "HVDDMPNAKLLL"))
subject <- AAString("SHLDTEKMSMKLL")
pa1 <- pairwiseAlignment(pattern, subject, substitutionMatrix="BLOSUM50",
                         gapOpening=3, gapExtension=1)
pa1

alignedPattern(pa1)
alignedSubject(pa1)
stopifnot(identical(width(alignedPattern(pa1)),
                    width(alignedSubject(pa1))))

as.character(pa1)

aligned(pa1)
as.matrix(pa1)
nchar(pa1)
score(pa1)
```

PairwiseAlignments-io *Write a PairwiseAlignments object to a file*

Description

The writePairwiseAlignments function writes a PairwiseAlignments object to a file. Only the "pair" format is supported at the moment.

Usage

```
writePairwiseAlignments(x, file="", Matrix=NA, block.width=50)
```

Arguments

x	A PairwiseAlignments object, typically returned by the pairwiseAlignment function.
file	A connection, or a character string naming the file to print to. If "" (the default), writePairwiseAlignments prints to the standard output connection (aka the console) unless redirected by sink. If it is "|cmd", the output is piped to the command given by cmd, by opening a pipe connection.
Matrix	A single string containing the name of the substitution matrix (e.g. "BLOSUM50") used for the alignment. See the substitutionMatrix argument of the pairwiseAlignment function for the details. See ?substitution_matrices for a list of predefined substitution matrices available in the **pwalign** package.
block.width	A single integer specifying the maximum number of sequence letters (including the "-" letter, which represents gaps) per line.

Details

The "pair" format is one of the numerous pairwise sequence alignment formats supported by the EMBOSS software. See http://emboss.sourceforge.net/docs/themes/AlignFormats.html for a brief (and rather informal) description of this format.

Value

Nothing (invisible NULL).

Note

This brief description of the "pair" format suggests that it is best suited for *global* pairwise alignments, because, in that case, the original pattern and subject sequences can be inferred (by just removing the gaps).

However, even though the "pair" format can also be used for non global pairwise alignments (i.e. for *global-local*, *local-global*, and *local* pairwise alignments), in that case the original pattern and subject sequences *cannot* be inferred. This is because the alignment written to the file doesn't necessarily span the entire pattern (if type(x) is local-global or local) or the entire subject (if type(x) is global-local or local).

As a consequence, the `writePairwiseAlignments` function can be used on a PairwiseAlignments object `x` containing non global alignments (i.e. with `type(x) != "global"`), but with the 2 following caveats:

1. The type of the alignments (`type(x)`) is not written to the file.
2. The original pattern and subject sequences cannot be inferred. Furthermore, there is no way to infer their lengths (because we don't know whether they were trimmed or not).

Also note that the `pairwiseAlignment` function interprets the `gapOpening` and `gapExtension` arguments differently than most other alignment tools. As a consequence the values of the Gap_penalty and Extend_penalty fields written to the file are not the same as the values that were passed to the `gapOpening` and `gapExtension` arguments. With the following relationship:

- Gap_penalty = gapOpening + gapExtension
- Extend_penalty = gapExtension

Author(s)

H. Pagès

References

http://emboss.sourceforge.net/docs/themes/AlignFormats.html

See Also

- pairwiseAlignment
- PairwiseAlignments-class
- substitution_matrices

Examples

```
## ---------------------------------------------------------------------
## A. WITH ONE PAIR
## ---------------------------------------------------------------------
pattern <- DNAString("CGTACGTAACGTTCGT")
subject <- DNAString("CGTCGTCGTCCGTAA")
pa1 <- pairwiseAlignment(pattern, subject)
pa1
writePairwiseAlignments(pa1)
writePairwiseAlignments(pa1, block.width=10)
## The 2 bottom-right numbers (16 and 15) are the lengths of
## the original pattern and subject, respectively.

pa2 <- pairwiseAlignment(pattern, subject, type="global-local")
pa2  # score is different!
writePairwiseAlignments(pa2)
## By just looking at the file, we can't tell the length of the
## original subject! Could be 13, could be more...

pattern <- DNAString("TCAACTTAACTT")
```

```
subject <- DNAString("GGGCAACAACGGG")
pa3 <- pairwiseAlignment(pattern, subject, type="global-local",
                         gapOpening=-2, gapExtension=-1)
writePairwiseAlignments(pa3)

## ---------------------------------------------------------------------
## B. WITH MORE THAN ONE PAIR (AND NAMED PATTERNS)
## ---------------------------------------------------------------------
pattern <- DNAStringSet(c(myp1="ACCA", myp2="ACGCA", myp3="ACGGCA"))
pa4 <- pairwiseAlignment(pattern, subject)
pa4
writePairwiseAlignments(pa4)

## ---------------------------------------------------------------------
## C. REPRODUCING THE ALIGNMENT SHOWN AT
##    http://emboss.sourceforge.net/docs/themes/alnformats/align.pair
## ---------------------------------------------------------------------
pattern <- c("TSPASIRPPAGPSSRPAMVSSRRTRPSPPGPRRPTGRPCCSAAPRRPQAT",
             "GGWKTCSGTCTTSTSTRHRGRSGWSARTTTAACLRASRKSMRAACSRSAG",
             "SRPNRFAPTLMSSCITSTTGPPAWAGDRSHE")
subject <- c("TSPASIRPPAGPSSRRPSPPGPRRPTGRPCCSAAPRRPQATGGWKTCSGT",
             "CTTSTSTRHRGRSGWRASRKSMRAACSRSAGSRPNRFAPTLMSSCITSTT",
             "GPPAWAGDRSHE")
pattern <- unlist(AAStringSet(pattern))
subject <- unlist(AAStringSet(subject))
pattern  # original pattern
subject  # original subject
data(BLOSUM62)
pa5 <- pairwiseAlignment(pattern, subject,
                         substitutionMatrix=BLOSUM62,
                         gapOpening=9.5, gapExtension=0.5)
pa5
writePairwiseAlignments(pa5, Matrix="BLOSUM62")
```

`phiX174Phage` *Versions of bacteriophage phiX174 complete genome and sample short reads*

Description

Six versions of the complete genome for bacteriophage ϕ X174 as well as a small number of Solexa short reads, qualities associated with those short reads, and counts for the number times those short reads occurred.

Format

`phiX174Phage`: A `DNAStringSet` containing the following six naturally occurring versions of the bacteriophage ϕ X174 genome cited in Smith et al.:

1. Genbank: The version of the genome from GenBank (NC_001422.1, GI:9626372).

2. RF70s: A preparation of ϕ X double-stranded replicative form (RF) of DNA by Clyde A. Hutchison III from the late 1970s.
3. SS78: A preparation of ϕ X virion single-stranded DNA from 1978.
4. Bull: The sequence of wild-type ϕ X used by Bull et al.
5. G'97: The ϕ X replicative form (RF) of DNA from Bull et al.
6. NEB'03: A ϕ X replicative form (RF) of DNA from New England BioLabs (NEB).

`srPhiX174`: A `DNAStringSet` containing short reads from a Solexa machine.

`quPhiX174`: A `BStringSet` containing Solexa quality scores associated with `srPhiX174`.

`wtPhiX174`: An integer vector containing counts associated with `srPhiX174`.

Author(s)

P. Aboyoun

References

- http://www.genome.jp/dbget-bin/www_bget?refseq+NC_001422
- Bull, J. J., Badgett, M. R., Wichman, H. A., Huelsenbeck, Hillis, D. M., Gulati, A., Ho, C. & Molineux, J. (1997) Genetics 147, 1497-1507.
- Smith, Hamilton O.; Clyde A. Hutchison, Cynthia Pfannkoch, J. Craig Venter (2003-12-23). "Generating a synthetic genome by whole genome assembly: {phi}X174 bacteriophage from synthetic oligonucleotides". Proceedings of the National Academy of Sciences 100 (26): 15440-15445. doi:10.1073/pnas.2237126100.

Examples

```
data(phiX174Phage)
nchar(phiX174Phage)
genBankPhage <- phiX174Phage[[1]]
genBankSubstring <- substring(genBankPhage, 2793-34, 2811+34)

data(srPhiX174)
srPhiX174
quPhiX174
summary(wtPhiX174)

alignPhiX174 <- pairwiseAlignment(srPhiX174, genBankSubstring,
                                  patternQuality=SolexaQuality(quPhiX174),
                                  subjectQuality=SolexaQuality(99L),
                                  type="global-local")

summary(alignPhiX174, weight=wtPhiX174)
```

pid *Percent Sequence Identity*

Description

Calculates the percent sequence identity for a pairwise sequence alignment.

Usage

```
pid(x, type="PID1")
```

Arguments

x	a PairwiseAlignments object.
type	one of percent sequence identity. One of "PID1", "PID2", "PID3", and "PID4". See Details for more information.

Details

Since there is no universal definition of percent sequence identity, the pid function calculates this statistic in the following types:

"PID1": 100 * (identical positions) / (aligned positions + internal gap positions)

"PID2": 100 * (identical positions) / (aligned positions)

"PID3": 100 * (identical positions) / (length shorter sequence)

"PID4": 100 * (identical positions) / (average length of the two sequences)

Value

A numeric vector containing the specified sequence identity measures.

Author(s)

P. Aboyoun

References

A. May, Percent Sequence Identity: The Need to Be Explicit, Structure 2004, 12(5):737.

G. Raghava and G. Barton, Quantification of the variation in percentage identity for protein sequence alignments, BMC Bioinformatics 2006, 7:415.

See Also

pairwiseAlignment, PairwiseAlignments-class, match-utils

Examples

```
s1 <- DNAString("AGTATAGATGATAGAT")
s2 <- DNAString("AGTAGATAGATGGATGATAGATA")

palign1 <- pairwiseAlignment(s1, s2)
palign1
pid(palign1)

palign2 <-
  pairwiseAlignment(s1, s2,
    substitutionMatrix =
    nucleotideSubstitutionMatrix(match = 2, mismatch = 10, baseOnly = TRUE))
palign2
pid(palign2, type = "PID4")
```

`predefined_scoring_matrices` *Predefined scoring matrices*

Description

Predefined scoring matrices for nucleotide and amino acid alignments.

Usage

```
data(BLOSUM45)
data(BLOSUM50)
data(BLOSUM62)
data(BLOSUM80)
data(BLOSUM100)
data(PAM30)
data(PAM40)
data(PAM70)
data(PAM120)
data(PAM250)
```

Format

The BLOSUM and PAM matrices are square symmetric matrices with integer coefficients, whose row and column names are identical and unique: each name is a single letter representing a nucleotide or an amino acid.

Details

The BLOSUM and PAM matrices are not unique. For example, the definition of the widely used BLOSUM62 matrix varies depending on the source, and even a given source can provide different versions of "BLOSUM62" without keeping track of the changes over time. NCBI provides many

matrices here ftp://ftp.ncbi.nih.gov/blast/matrices/ but their definitions don't match those of the matrices bundled with their stand-alone BLAST software available here ftp://ftp.ncbi.nih.gov/blast/

The BLOSUM45, BLOSUM62, BLOSUM80, PAM30 and PAM70 matrices were taken from NCBI stand-alone BLAST software.

The BLOSUM50, BLOSUM100, PAM40, PAM120 and PAM250 matrices were taken from ftp://ftp.ncbi.nih.gov/blast/matric

Author(s)

H. Pagès and P. Aboyoun

See Also

nucleotideSubstitutionMatrix, pairwiseAlignment, PairwiseAlignments-class, DNAString-class, AAString-class, PhredQuality-class, SolexaQuality-class, IlluminaQuality-class

Examples

```
## Align two amino acid sequences with the BLOSUM62 matrix:
aa1 <- AAString("HXBLVYMGCHFDCXVBEHIKQZ")
aa2 <- AAString("QRNYMYCFQCISGNEYKQN")
pairwiseAlignment(aa1, aa2, substitutionMatrix="BLOSUM62",
                            gapOpening=3, gapExtension=1)

## See how the gap penalty influences the alignment:
pairwiseAlignment(aa1, aa2, substitutionMatrix="BLOSUM62",
                            gapOpening=6, gapExtension=2)

## See how the substitution matrix influences the alignment:
pairwiseAlignment(aa1, aa2, substitutionMatrix="BLOSUM50",
                            gapOpening=3, gapExtension=1)

if (interactive()) {
  ## Compare our BLOSUM62 with BLOSUM62 from
  ## ftp://ftp.ncbi.nih.gov/blast/matrices/:
  data(BLOSUM62)
  BLOSUM62["Q", "Z"]
  file <- "ftp://ftp.ncbi.nih.gov/blast/matrices/BLOSUM62"
  b62 <- as.matrix(read.table(file, check.names=FALSE))
  b62["Q", "Z"]
}
```

---

stringDist *String Distance/Alignment Score Matrix*

Description

Computes the Levenshtein edit distance or pairwise alignment score matrix for a set of strings.

Usage

```
stringDist(x, method = "levenshtein", ignoreCase = FALSE, diag = FALSE, upper = FALSE, ...)
## S4 method for signature 'XStringSet'
stringDist(x, method = "levenshtein", ignoreCase = FALSE, diag = FALSE,
                  upper = FALSE, type = "global", quality = PhredQuality(22L),
                 substitutionMatrix = NULL, fuzzyMatrix = NULL, gapOpening = 0,
                  gapExtension = 1)
## S4 method for signature 'QualityScaledXStringSet'
stringDist(x, method = "quality", ignoreCase = FALSE,
             diag = FALSE, upper = FALSE, type = "global", substitutionMatrix = NULL,
                  fuzzyMatrix = NULL, gapOpening = 0, gapExtension = 1)
```

Arguments

x	a character vector or an XStringSet object.
method	calculation method. One of "levenshtein", "hamming", "quality", or "substitutionMatrix".
ignoreCase	logical value indicating whether to ignore case during scoring.
diag	logical value indicating whether the diagonal of the matrix should be printed by print.dist.
upper	logical value indicating whether the upper triangle of the matrix should be printed by print.dist.
type	(applicable when method = "quality" or method = "substitutionMatrix"). type of alignment. One of "global", "local", and "overlap", where "global" = align whole strings with end gap penalties, "local" = align string fragments, "overlap" = align whole strings without end gap penalties.
quality	(applicable when method = "quality"). object of class XStringQuality representing the quality scores for x that are used in a quality-based method for generating a substitution matrix.
substitutionMatrix	(applicable when method = "substitutionMatrix"). symmetric matrix representing the fixed substitution scores in the alignment.
fuzzyMatrix	(applicable when method = "quality"). fuzzy match matrix for quality-based alignments. It takes values between 0 and 1; where 0 is an unambiguous mismatch, 1 is an unambiguous match, and values in between represent a fraction of "matchiness".
gapOpening	(applicable when method = "quality" or method = "substitutionMatrix"). penalty for opening a gap in the alignment.
gapExtension	(applicable when method = "quality" or method = "substitutionMatrix"). penalty for extending a gap in the alignment
...	optional arguments to generic function to support additional methods.

Details

When method = "hamming", uses the underlying neditStartingAt code to calculate the distances, where the Hamming distance is defined as the number of substitutions between two strings of equal length. Otherwise, uses the underlying pairwiseAlignment code to compute the distance/alignment score matrix.

Value

Returns an object of class "dist".

Author(s)

P. Aboyoun

See Also

dist, agrep, pairwiseAlignment, substitution_matrices

Examples

```
stringDist(c("lazy", "HaZy", "crAzY"))
stringDist(c("lazy", "HaZy", "crAzY"), ignoreCase = TRUE)

data(phiX174Phage)
plot(hclust(stringDist(phiX174Phage), method = "single"))

data(srPhiX174)
stringDist(srPhiX174[1:4])
stringDist(srPhiX174[1:4], method = "quality",
           quality = SolexaQuality(quPhiX174[1:4]),
           gapOpening = 10, gapExtension = 4)
```

---

substitution_matrices *Utilities to generate substitution matrices*

Description

Utilities to generate substitution matrices.

Usage

```
nucleotideSubstitutionMatrix(match=1, mismatch=0, baseOnly=FALSE,
                             type="DNA", symmetric=TRUE)

qualitySubstitutionMatrices(fuzzyMatch=c(0, 1), alphabetLength=4L,
                            qualityClass="PhredQuality", bitScale=1)

errorSubstitutionMatrices(errorProbability, fuzzyMatch=c(0, 1),
                          alphabetLength=4L, bitScale=1)
```

Arguments

match	the scoring for a nucleotide match.
mismatch	the scoring for a nucleotide mismatch.
baseOnly	TRUE or FALSE. If TRUE, only uses the letters in the "base" alphabet i.e. "A", "C", "G", "T".
type	either "DNA" or "RNA".
symmetric	TRUE or FALSE. Default is TRUE. If FALSE, the resulting matrix will be asymmetric.
fuzzyMatch	a named or unnamed numeric vector representing the base match probability.
errorProbability	a named or unnamed numeric vector representing the error probability.
alphabetLength	an integer representing the number of letters in the underlying string alphabet. For DNA and RNA, this would be 4L. For Amino Acids, this could be 20L.
qualityClass	a character string of "PhredQuality", "SolexaQuality", or "IlluminaQuality".
bitScale	a numeric value to scale the quality-based substitution matrices. By default, this is 1, representing bit-scale scoring.

Details

The quality matrices computed in qualitySubstitutionMatrices are based on the paper by Ketil Malde. Let ϵ_i be the probability of an error in the base read. For "Phred" quality measures Q in $[0, 99]$, these error probabilities are given by $\epsilon_i = 10^{-Q/10}$. For "Solexa" quality measures Q in $[-5, 99]$, they are given by $\epsilon_i = 1 - 1/(1 + 10^{-Q/10})$. Assuming independence within and between base reads, the combined error probability of a mismatch when the underlying bases do match is $\epsilon_c = \epsilon_1 + \epsilon_2 - (n/(n-1)) * \epsilon_1 * \epsilon_2$, where n is the number of letters in the underlying alphabet. Using ϵ_c, the substitution score is given by when two bases match is given by $b * \log_2(\gamma_{x,y} * (1 - \epsilon_c) * n + (1 - \gamma_{x,y}) * \epsilon_c * (n/(n-1)))$, where b is the bit-scaling for the scoring and $\gamma_{x,y}$ is the probability that characters x and y represents the same underlying information (e.g. using IUPAC, $\gamma_{A,A} = 1$ and $\gamma_{A,N} = 1/4$. In the arguments listed above fuzzyMatch represents $\gamma_{x,y}$ and errorProbability represents ϵ_i.

Value

A matrix.

Author(s)

P. Aboyoun, with contribution from Albert Vill (support for asymmetric matrices in nucleotideSubstitutionMatrix())

References

K. Malde, The effect of sequence quality on sequence alignment, Bioinformatics, Feb 23, 2008.

See Also

predefined_scoring_matrices, pairwiseAlignment, PairwiseAlignments-class, DNAString-class, AAString-class, PhredQuality-class, SolexaQuality-class, IlluminaQuality-class

Examples

```
s1 <- DNAString("ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAACGCAAAGTTTTCAAG")
s2 <- DNAString("GTTTCACTACTTCCTTTCGGGTAAGTAAATATATAAATATATAAAAATATAATTTTCATC")
s3 <- DNAString("NCTTCRCCAGCTCCCTGGMGGTAAGTTGATCAAAGVAAACGCAAAGTTNTCAAG")

## Fit a global pairwise alignment using edit distance scoring:
nsm <- nucleotideSubstitutionMatrix(0, -1, TRUE)
pairwiseAlignment(s1, s2, substitutionMatrix=nsm,
                          gapOpening=0, gapExtension=1)

## Align sequences using an asymmetric substitution matrix.
## The asymmetry of the matrix means that the query sequence is not
## penalized for ambiguous bases in the subject / consensus sequence:
ansm <- nucleotideSubstitutionMatrix(symmetric=FALSE)
ansm["M", c("A","C","G","T")]
#   A   C   G   T
# 0.5 0.5 0.0 0.0
ansm[c("A","C","G","T"), "M"]
# A C G T
# 1 1 0 0
ansm["M", "H"]
# 1
ansm["H", "M"]
# 0.6666667

## Due to this asymmetry, the order of the sequences is important:
pairwiseAlignment(s1, s3, substitutionMatrix=ansm)
pairwiseAlignment(s3, s1, substitutionMatrix=ansm)

## Examine quality-based match and mismatch bit scores for DNA/RNA
## strings in pairwiseAlignment. By default patternQuality and
## subjectQuality are PhredQuality(22L):
qualityMatrices <- qualitySubstitutionMatrices()
qualityMatrices["22", "22", "1"]
qualityMatrices["22", "22", "0"]

pairwiseAlignment(s1, s2)

## Get the substitution scores when the error probability is 0.1:
subscores <- errorSubstitutionMatrices(errorProbability=0.1)
submat <- matrix(subscores[ , , "0"], 4, 4)
diag(submat) <- subscores[ , , "1"]
dimnames(submat) <- list(DNA_ALPHABET[1:4], DNA_ALPHABET[1:4])
submat
pairwiseAlignment(s1, s2, substitutionMatrix=submat)
```

Index

∗ **character**
stringDist, 20
∗ **classes**
AlignedXStringSet-class, 4
InDel-class, 6
PairwiseAlignments-class, 10
∗ **cluster**
stringDist, 20
∗ **datasets**
phiX174Phage, 16
predefined_scoring_matrices, 19
∗ **data**
predefined_scoring_matrices, 19
∗ **manip**
PairwiseAlignments-io, 14
∗ **methods**
align-utils, 2
AlignedXStringSet-class, 4
InDel-class, 6
pairwiseAlignment, 7
PairwiseAlignments-class, 10
pid, 18
∗ **models**
pairwiseAlignment, 7
∗ **multivariate**
stringDist, 20
∗ **utilities**
PairwiseAlignments-io, 14
substitution_matrices, 22

AAString-class, *20*, *23*
agrep, *22*
align-utils, 2, *13*
aligned *(*AlignedXStringSet-class*)*, 4
aligned,AlignedXStringSet0-method *(*AlignedXStringSet-class*)*, 4
aligned,PairwiseAlignmentsSingleSubject-method *(*PairwiseAlignments-class*)*, 10
alignedPattern *(*PairwiseAlignments-class*)*, 10
alignedPattern,PairwiseAlignments-method *(*PairwiseAlignments-class*)*, 10
alignedSubject *(*PairwiseAlignments-class*)*, 10
alignedSubject,PairwiseAlignments-method *(*PairwiseAlignments-class*)*, 10
AlignedXStringSet *(*AlignedXStringSet-class*)*, 4
AlignedXStringSet-class, *3*, 4, *13*
AlignedXStringSet0 *(*AlignedXStringSet-class*)*, 4
AlignedXStringSet0-class *(*AlignedXStringSet-class*)*, 4
alphabet, *3*, *12*
as.character,AlignedXStringSet0-method *(*AlignedXStringSet-class*)*, 4
as.character,PairwiseAlignmentsSingleSubject-method *(*PairwiseAlignments-class*)*, 10
as.matrix,PairwiseAlignmentsSingleSubject-method *(*PairwiseAlignments-class*)*, 10

BLOSUM100 *(*predefined_scoring_matrices*)*, 19
BLOSUM45 *(*predefined_scoring_matrices*)*, 19
BLOSUM50 *(*predefined_scoring_matrices*)*, 19
BLOSUM62 *(*predefined_scoring_matrices*)*, 19
BLOSUM80 *(*predefined_scoring_matrices*)*, 19
BString, *3*
BStringSet, *3*

class:AlignedXStringSet *(*AlignedXStringSet-class*)*, 4
class:AlignedXStringSet0 *(*AlignedXStringSet-class*)*, 4
class:InDel *(*InDel-class*)*, 6

class:PairwiseAlignments
(PairwiseAlignments-class), 10
class:PairwiseAlignmentsSingleSubject
(PairwiseAlignments-class), 10
class:PairwiseAlignmentsSingleSubjectSummary
(PairwiseAlignments-class), 10
class:QualityAlignedXStringSet
(AlignedXStringSet-class), 4
compareStrings (align-utils), 2
compareStrings,AlignedXStringSet0,AlignedXStringSet0-method
(align-utils), 2
compareStrings,character,character-method
(align-utils), 2
compareStrings,PairwiseAlignments,missing-method
(align-utils), 2
compareStrings,XString,XString-method
(align-utils), 2
compareStrings,XStringSet,XStringSet-method
(align-utils), 2
CompressedIRangesList, *13*
consensusMatrix, *3*, *12*
consensusMatrix,PairwiseAlignmentsSingleSubject-method
(align-utils), 2
consensusString, *12*
coverage, *3*
coverage,AlignedXStringSet0-method
(align-utils), 2
coverage,PairwiseAlignmentsSingleSubject-method,
12
coverage,PairwiseAlignmentsSingleSubject-method
(align-utils), 2
coverage,PairwiseAlignmentsSingleSubjectSummary-method
(align-utils), 2

deletion (InDel-class), 6
deletion,InDel-method (InDel-class), 6
deletion,PairwiseAlignments-method
(PairwiseAlignments-class), 10
dist, *22*
DNAString-class, *20*, *23*

end,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
errorSubstitutionMatrices
(substitution_matrices), 22

IlluminaQuality-class, *20*, *23*
InDel (InDel-class), 6
indel (AlignedXStringSet-class), 4
indel,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
indel,PairwiseAlignments-method
(PairwiseAlignments-class), 10
InDel-class, 6
insertion (InDel-class), 6
insertion,InDel-method (InDel-class), 6
insertion,PairwiseAlignments-method
(PairwiseAlignments-class), 10
length,PairwiseAlignmentsSingleSubjectSummary-method
(PairwiseAlignments-class), 10

match-utils, *3*, *18*
matchPattern, *9*
matchPDict, *9*
mismatch,AlignedXStringSet0,missing-method
(align-utils), 2
mismatchSummary (align-utils), 2
mismatchSummary,AlignedXStringSet0-method
(align-utils), 2
mismatchSummary,PairwiseAlignmentsSingleSubject-method
(align-utils), 2
mismatchSummary,PairwiseAlignmentsSingleSubjectSummary-met
(align-utils), 2
mismatchSummary,QualityAlignedXStringSet-method
(align-utils), 2
mismatchTable (align-utils), 2
mismatchTable,AlignedXStringSet0-method
(align-utils), 2
mismatchTable,PairwiseAlignments-method
(align-utils), 2
mismatchTable,QualityAlignedXStringSet-method
(align-utils), 2

nchar,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
nchar,PairwiseAlignments-method
(PairwiseAlignments-class), 10
nchar,PairwiseAlignmentsSingleSubjectSummary-method
(PairwiseAlignments-class), 10
nedit (align-utils), 2
nedit,PairwiseAlignments-method
(align-utils), 2
nedit,PairwiseAlignmentsSingleSubjectSummary-method
(align-utils), 2
nindel (AlignedXStringSet-class), 4
nindel,AlignedXStringSet0-method
(AlignedXStringSet-class), 4

nindel,PairwiseAlignments-method
(PairwiseAlignments-class), 10
nindel,PairwiseAlignmentsSingleSubjectSummary-method
(PairwiseAlignments-class), 10
nmatch,PairwiseAlignments,missing-method
(align-utils), 2
nmatch,PairwiseAlignmentsSingleSubjectSummary,missing-method
(align-utils), 2
nmismatch,AlignedXStringSet0,missing-method
(align-utils), 2
nmismatch,PairwiseAlignments,missing-method
(align-utils), 2
nmismatch,PairwiseAlignmentsSingleSubjectSummary,missing-method
(align-utils), 2
nucleotideSubstitutionMatrix, *20*
nucleotideSubstitutionMatrix
(substitution_matrices), 22

pairwiseAlignment, *3*, *6*, 7, *13–15*, *18*, *20*,
22, *23*
pairwiseAlignment,ANY,ANY-method
(pairwiseAlignment), 7
pairwiseAlignment,ANY,QualityScaledXStringSet-method
(pairwiseAlignment), 7
pairwiseAlignment,QualityScaledXStringSet,ANY-method
(pairwiseAlignment), 7
pairwiseAlignment,QualityScaledXStringSet,QualityScaledXStringSet-method
(pairwiseAlignment), 7
PairwiseAlignments, *8*, *14*, *15*, *18*
PairwiseAlignments
(PairwiseAlignments-class), 10
PairwiseAlignments,character,character-method
(PairwiseAlignments-class), 10
PairwiseAlignments,character,missing-method
(PairwiseAlignments-class), 10
PairwiseAlignments,XString,XString-method
(PairwiseAlignments-class), 10
PairwiseAlignments,XStringSet,missing-method
(PairwiseAlignments-class), 10
PairwiseAlignments-class, *3*, *9*, 10, *15*, *18*,
20, *23*
PairwiseAlignments-io, 14
PairwiseAlignmentsSingleSubject, *8*
PairwiseAlignmentsSingleSubject
(PairwiseAlignments-class), 10
PairwiseAlignmentsSingleSubject,character,character-method
(PairwiseAlignments-class), 10
PairwiseAlignmentsSingleSubject,character,missing-method
(PairwiseAlignments-class), 10
PairwiseAlignmentsSingleSubject,XString,XString-method
(PairwiseAlignments-class), 10
PairwiseAlignmentsSingleSubject,XStringSet,missing-method
(PairwiseAlignments-class), 10
PairwiseAlignmentsSingleSubject-class
(PairwiseAlignments-class), 10
PairwiseAlignmentsSingleSubjectSummary
(PairwiseAlignments-class), 10
PairwiseAlignmentsSingleSubjectSummary-class
(PairwiseAlignments-class), 10
PAM120 (predefined_scoring_matrices), 19
PAM250 (predefined_scoring_matrices), 19
PAM30 (predefined_scoring_matrices), 19
PAM40 (predefined_scoring_matrices), 19
PAM70 (predefined_scoring_matrices), 19
parallel_slot_names,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
parallel_slot_names,PairwiseAlignments-method
(PairwiseAlignments-class), 10
parallelVectorNames,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
pattern (PairwiseAlignments-class), 10
pattern,PairwiseAlignments-method
(PairwiseAlignments-class), 10
phiX174Phage, 16
PhredQuality-class, *20*, *23*
pid, *13*, 18
pid,PairwiseAlignments-method (pid), 18
predefined_scoring_matrices, 19, *23*

QualityAlignedXStringSet
(AlignedXStringSet-class), 4
QualityAlignedXStringSet-class
(AlignedXStringSet-class), 4
QualityScaledXStringSet, *7*
qualitySubstitutionMatrices
(substitution_matrices), 22
quPhiX174 (phiX174Phage), 16

ranges,AlignedXStringSet0-method
(AlignedXStringSet-class), 4

score,PairwiseAlignments-method
(PairwiseAlignments-class), 10
score,PairwiseAlignmentsSingleSubjectSummary-method
(PairwiseAlignments-class), 10
seqtype,AlignedXStringSet0-method
(AlignedXStringSet-class), 4

seqtype,PairwiseAlignments-method
(PairwiseAlignments-class), 10
show,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
show,PairwiseAlignments-method
(PairwiseAlignments-class), 10
show,PairwiseAlignmentsSingleSubjectSummary-method
(PairwiseAlignments-class), 10
SolexaQuality-class, *20*, *23*
srPhiX174 (phiX174Phage), 16
start,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
stringDist, *9*, 20
stringDist,character-method
(stringDist), 20
stringDist,QualityScaledXStringSet-method
(stringDist), 20
stringDist,XStringSet-method
(stringDist), 20
subject,PairwiseAlignments-method
(PairwiseAlignments-class), 10
substitution_matrices, *9*, *14*, *15*, *22*, 22
summary,PairwiseAlignmentsSingleSubject-method
(PairwiseAlignments-class), 10

toString,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
toString,PairwiseAlignmentsSingleSubject-method
(PairwiseAlignments-class), 10
type (PairwiseAlignments-class), 10
type,PairwiseAlignments-method
(PairwiseAlignments-class), 10
type,PairwiseAlignmentsSingleSubjectSummary-method
(PairwiseAlignments-class), 10

unaligned (AlignedXStringSet-class), 4
unaligned,AlignedXStringSet0-method
(AlignedXStringSet-class), 4

Views,PairwiseAlignmentsSingleSubject-method
(PairwiseAlignments-class), 10
vmatchPattern, *9*

width,AlignedXStringSet0-method
(AlignedXStringSet-class), 4
writePairwiseAlignments, *9*, *13*
writePairwiseAlignments
(PairwiseAlignments-io), 14
wtPhiX174 (phiX174Phage), 16

XString, *7*, *8*, *11*
XString-class, *3*, *13*
XStringQuality, *7*, *21*
XStringQuality-class, *9*
XStringSet, *7*, *8*, *11*, *21*
XStringSet-class, *3*
XStringViews, *3*
XStringViews-class, *3*, *13*