### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAString-class</td>
<td>2</td>
</tr>
<tr>
<td>AlignedXStringSet-class</td>
<td>3</td>
</tr>
<tr>
<td>align-utils</td>
<td>5</td>
</tr>
<tr>
<td>AMINO_ACID_CODE</td>
<td>6</td>
</tr>
<tr>
<td>basecontent</td>
<td>7</td>
</tr>
<tr>
<td>Biostrings internals</td>
<td>8</td>
</tr>
<tr>
<td>BOC_SubjectString-class</td>
<td>8</td>
</tr>
<tr>
<td>chartr</td>
<td>9</td>
</tr>
<tr>
<td>complementSeq</td>
<td>10</td>
</tr>
<tr>
<td>DNAString-class</td>
<td>11</td>
</tr>
<tr>
<td>findPalindromes</td>
<td>12</td>
</tr>
<tr>
<td>GENETIC_CODE</td>
<td>14</td>
</tr>
<tr>
<td>gregexpr2</td>
<td>16</td>
</tr>
<tr>
<td>InDel-class</td>
<td>17</td>
</tr>
<tr>
<td>injectHardMask</td>
<td>17</td>
</tr>
<tr>
<td>IUPAC_CODE_MAP</td>
<td>19</td>
</tr>
<tr>
<td>letterFrequency</td>
<td>20</td>
</tr>
<tr>
<td>letter</td>
<td>23</td>
</tr>
<tr>
<td>longestConsecutive</td>
<td>24</td>
</tr>
<tr>
<td>MaskedXString-class</td>
<td>25</td>
</tr>
<tr>
<td>maskMotif</td>
<td>27</td>
</tr>
<tr>
<td>matchLRPatterns</td>
<td>29</td>
</tr>
<tr>
<td>matchPattern</td>
<td>31</td>
</tr>
<tr>
<td>matchPDict</td>
<td>34</td>
</tr>
<tr>
<td>matchPDict-inexact</td>
<td>39</td>
</tr>
<tr>
<td>matchProbePair</td>
<td>42</td>
</tr>
<tr>
<td>matchprobes</td>
<td>43</td>
</tr>
<tr>
<td>matchPWM</td>
<td>44</td>
</tr>
<tr>
<td>match-utils</td>
<td>46</td>
</tr>
<tr>
<td>MIndex-class</td>
<td>50</td>
</tr>
<tr>
<td>needwunsQS</td>
<td>51</td>
</tr>
<tr>
<td>nucleotideFrequency</td>
<td>52</td>
</tr>
<tr>
<td>PairwiseAlignedXStringSet-class</td>
<td>55</td>
</tr>
<tr>
<td>pairwiseAlignment</td>
<td>59</td>
</tr>
<tr>
<td>PDict-class</td>
<td>61</td>
</tr>
<tr>
<td>phiX174Phage</td>
<td>65</td>
</tr>
<tr>
<td>pid</td>
<td>66</td>
</tr>
</tbody>
</table>
AAString-class

Description

An AAString object allows efficient storage and manipulation of a long amino acid sequence.

Details

The AAString class is a direct XString subclass (with no additional slot). Therefore all functions and methods described in the XString man page also work with an AAString object (inheritance).

Unlike the BString container that allows storage of any single string (based on a single-byte character set) the AAString container can only store a string based on the Amino Acid alphabet (see below).

The Amino Acid alphabet

This alphabet contains all letters from the Single-Letter Amino Acid Code (see ?AMINO_ACID_CODE) + the stop ("*"), the gap ("-" ) and the hard masking ("+" ) letters. It is stored in the AA_ALPHABET constant (character vector). The alphabet method also returns AA_ALPHABET when applied to an AAString object and is provided for convenience only.

Constructor-like functions and generics

In the code snippet below, x can be a single string (character vector of length 1) or a BString object.

AAString(x="", start=1, nchar=NA): Tries to convert x into an AAString object by reading nchar letters starting at position start in x.
AlignedXStringSet-class

Accessor methods

In the code snippet below, x is an AAString object.

`alphabet(x)`: If x is an AAString object, then return the Amino Acid alphabet (see above).

See the corresponding man pages when x is a BString, DNAString or RNAString object.

Author(s)

H. Pages

See Also

AMINO_ACID_CODE, letter, XString-class, alphabetFrequency

Examples

```r
AA_ALPHABET
a <- AAString("MARKSEMSIR*")
length(a)
alphabet(a)
```

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 $\geq i$ 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.
- `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 and H. Pages

**See Also**

`pairwiseAlignment, PairwiseAlignedXStringSet-class, XStringSet-class`

**Examples**

```r
pattern <- AString("LAND")
subject <- AString("LEAVES")
nw1 <- pairwiseAlignment(pattern, subject, substitutionMatrix = "BLOSUM50", gapOpening
alignedPattern <- pattern(nw1)
unaligned(alignedPattern)
```
align-utils

Utility functions related to sequence alignment

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

Usage

\[
\text{nedit}(x) \quad \# \text{ also nmatch and nmismatch}
\]

\[
\text{mismatchTable}(x, \text{shiftLeft} = 0L, \text{shiftRight} = 0L, \ldots)
\]

\[
\text{mismatchSummary}(x, \ldots)
\]

## S4 method for signature 'AlignedXStringSet0':
\[
\text{coverage}(x, \text{start} = \text{NA}, \text{end} = \text{NA}, \text{shift} = 0L, \text{width} = \text{NULL}, \text{weight} = 1L)
\]

## S4 method for signature 'PairwiseAlignedFixedSubject':
\[
\text{coverage}(x, \text{start} = \text{NA}, \text{end} = \text{NA}, \text{shift} = 0L, \text{width} = \text{NULL}, \text{weight} = 1L)
\]

\[
\text{compareStrings}(\text{pattern}, \text{subject})
\]

## S4 method for signature 'PairwiseAlignedFixedSubject':
\[
\text{consensusMatrix}(x, \text{baseOnly} = \text{FALSE}, \text{freq} = \text{FALSE},
\quad \text{gapCode} = \text{"-"}, \text{endgapCode} = \text{"-"})
\]

Arguments

\begin{itemize}
\item \textbf{x}\text{ A character vector or matrix, XStringSet, XStringViews, PairwiseAlignedXStringSet, or list of FASTA records containing the equal-length strings.}
\item \textbf{shiftLeft, shiftRight}\text{ 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.}
\item \textbf{...}\text{ Further arguments to be passed to or from other methods.}
\item \textbf{start, end, shift, width}\text{ See ?coverage.}
\item \textbf{weight}\text{ An integer vector specifying how much each element in x counts.}
\item \textbf{pattern, subject}\text{ The strings to compare. Can be of type character, XString, XStringSet, AlignedXStringSet, or, in the case of pattern, PairwiseAlignedXStringSet. If pattern is a PairwiseAlignedXStringSet object, then subject must be missing.}
\item \textbf{baseOnly}\text{ 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.}
\item \textbf{freq}\text{ If TRUE, then letter frequencies (per position) are reported, otherwise counts.}
\item \textbf{gapCode, endgapCode}\text{ The codes in the appropriate alphabet to use for the internal and end gaps.}
\end{itemize}
mismatchTable: a data.frame containing the positions and substrings of the mismatches for the AlignedXStringSet or PairwiseAlignedXStringSet object.
mismatchSummary: a list of data.frame objects containing counts and frequencies of the mismatches for the AlignedXStringSet or PairwiseAlignedFixedSubject 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 "-".

See Also

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

Examples

```r
## Compare two globally aligned strings
string1 <- "ACTTCACCAGCTCCCTGGCGGTAAGTTGATC---AAAG---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]
```

AMINO_ACID_CODE

The Single-Letter Amino Acid Code

Description

Named character vector mapping single-letter amino acid representations to 3-letter amino acid representations.

See Also

AAString, GENETIC_CODE
Examples

```r
## See all the 3-letter codes
AMINO_ACID_CODE

## Convert an AAString object to a vector of 3-letter amino acid codes
aa <- AAString("LANDEECQW")
AMINO_ACID_CODE[strsplit(as.character(aa), NULL)[[1]]]
```

Description

WARNING: Both `basecontent` and `countbases` have been deprecated in favor of `alphabetFrequency`.

These functions accept a character vector representing the nucleotide sequences and compute the frequencies of each base (A, C, G, T).

Usage

```r
basecontent(seq)
countbases(seq, dna = TRUE)
```

Arguments

- `seq`: Character vector.
- `dna`: Logical value indicating whether the sequence is DNA (TRUE) or RNA (FALSE).

Details

The base frequencies are calculated separately for each element of `x`. The elements of `x` can be in upper case, lower case or mixed.

Value

A matrix with 4 columns and `length(x)` rows. The columns are named A, C, T, G, and the values in each column are the counts of the corresponding bases in the elements of `x`. When `dna=FALSE`, the T column is replaced with a U column.

Author(s)

R. Gentleman, W. Huber, S. Falcon

See Also

`alphabetFrequency`, `reverseComplement`
Examples

```r
v <- c("AAACT", "GGGTT", "ggAtT")

## Do not use these functions anymore:
if (interactive()) {
  basecontent(v)
  countbases(v)
}

## But use more efficient alphabetFrequency() instead:
v <- DNAStringSet(v)
alphabetFrequency(v, baseOnly=TRUE)

## Comparing efficiencies:
if (interactive()) {
  library(hgu95av2probe)
  system.time(y1 <- countbases(hgu95av2probe$sequence))
  x <- DNAStringSet(hgu95av2probe$sequence)
  system.time(y2 <- alphabetFrequency(x, baseOnly=TRUE))
}
```

Biostrings internals

Biostrings objects, classes and methods that are not intended to be used directly.

Author(s)

H. Pages

BOC_SubjectString-class

BOC_SubjectString and BOC2_SubjectString objects

Description

The BOC_SubjectString and BOC2_SubjectString classes are experimental and might not work properly.

Please DO NOT TRY TO USE them for now. Thanks for your comprehension!

Author(s)

H. Pages
**chartr**  
Translating letters of a sequence

**Description**

Translate letters of a sequence.

**Usage**

```r
## S4 method for signature 'ANY, ANY, XString':
chartr(old, new, x)
```

**Arguments**

- `old` A character string specifying the characters to be translated.
- `new` A character string specifying the translations.
- `x` The sequence or set of sequences to translate. If `x` is an `XString`, `XStringSet`, `XStringViews` or `MaskedXString` object, then the appropriate `chartr` method is called, otherwise the standard `chartr` R function is called.

**Details**

See `?chartr` for the details.

Note that, unlike the standard `chartr` R function, the methods for `XString`, `XStringSet`, `XStringViews` and `MaskedXString` objects do NOT support character ranges in the specifications.

**Value**

An object of the same class and length as the original object.

**See Also**

`chartr, replaceLetterAt, XString-class, XStringSet-class, XStringViews-class, MaskedXString-class, alphabetFrequency, matchPattern, reverseComplement`

**Examples**

```r
x <- BString("MiXeD cAsE 123")
chartr("iXs", "why", x)
```

```r
# TRANSFORMING DNA WITH BISULFITE (AND SEARCHING IT...)
# -----------------------------------------------
library(BSgenome.Celegans.UCSC.ce2)
chrII <- Celegans["chrII"]
alphabetFrequency(chrII)
pattern <- DNAString("TGGGTGTATTTA")

# Transforming and searching the + strand
plus_strand <- chartr("C", "T", chrII)
alphabetFrequency(plus_strand)
```
### complementSeq

**Complementary sequence.**

**Description**

**WARNING:** `complementSeq` has been deprecated in favor of `complement`. Function to obtain the complementary sequence.

**Usage**

```r
complementSeq(seq, start=1, stop=0)
```

**Arguments**

- `seq`: Character vector consisting of the letters A, C, G and T.
- `start`: Numeric scalar: the sequence position at which to start complementing. If 1, start from the beginning.
- `stop`: Numeric scalar: the sequence position at which to stop complementing. If 0, go until the end.

**Details**

The complemented sequence for each element of the input is computed and returned. The complement is given by the mapping: A -> T, C -> G, G -> C, T -> A.

An important special case is `start=13, stop=13`: If `seq` is a vector of 25mer sequences on an Affymetrix GeneChip, `complementSeq(seq, start=13, stop=13)` calculates the so-called mismatch sequences.

The function deals only with sequences that represent DNA. These can consist only of the letters A, C, T or G. Upper, lower or mixed case is allowed and honored.

**Value**

A character vector of the same length as `seq` is returned. Each component represents the transformed sequence for the input value.

**Author(s)**

R. Gentleman, W. Huber

**See Also**

`alphabetFrequency`, `reverseComplement`
DNAString-class

Examples

```r
## EXAMPLE 1
##
seq <- c("AAACT", "GGGTT")

## Don't do this anymore (deprecated):
if (interactive()) {
  complementSeq(seq) # inefficient on large vectors
}
## But do this instead:
complement(DNAStringSet(seq)) # more efficient

## EXAMPLE 2
##
seq <- c("CGACTAGACAGACCAACAG", "CCCGCATCATCTTTCCTGTGCTCTT")

## Don't do this anymore (deprecated):
if (interactive()) {
  complementSeq(seq, start=13, stop=13)
}
## But do this instead:
pm2mm <- function(probes)
{
  probes <- DNAStringSet(probes)
  subseq(probes, start=13, end=13) <- complement(subseq(probes, start=13, end=13))
  probes
}
pm2mm(seq)
```

## SPEED OF complementSeq() VS complement()

```r
if (interactive()) {
  library(hgu95av2probe)
  system.time(y1 <- complementSeq(hgu95av2probe$sequence))
  probes <- DNAStringSet(hgu95av2probe$sequence)
  system.time(y2 <- complement(probes))
}
```

DNAString-class

DNAString objects

Description

A DNAString object allows efficient storage and manipulation of a long DNA sequence.

Details

The DNAString class is a direct XString subclass (with no additional slot). Therefore all functions and methods described in the XString man page also work with a DNAString object (inheritance). Unlike the BString container that allows storage of any single string (based on a single-byte character set) the DNAString container can only store a string based on the DNA alphabet (see below).
In addition, the letters stored in a DNAString object are encoded in a way that optimizes fast search algorithms.

The DNA alphabet

This alphabet contains all letters from the IUPAC Extended Genetic Alphabet (see `?IUPAC_CODE_MAP`) + the gap ("-") and the hard masking ("+") letters. It is stored in the DNA_ALPHABET constant (character vector). The alphabet method also returns DNA_ALPHABET when applied to a DNAString object and is provided for convenience only.

Constructor-like functions and generics

In the code snippet below, x can be a single string (character vector of length 1), a BString object or an RNAString object.

```r
DNAString(x="", start=1, nchar=NA): Tries to convert x into a DNAString object by reading nchar letters starting at position start in x.
```

Accessor methods

In the code snippet below, x is a DNAString object.

```r
alphabet(x, baseOnly=FALSE): If x is a DNAString object, then return the DNA alphabet (see above). See the corresponding man pages when x is a BString, RNAString or AAString object.
```

Author(s)

H. Pages

See Also

`IUPAC_CODE_MAP`, `letter`, `XString-class`, `RNAString-class`, `reverseComplement`, `alphabetFrequency`

Examples

```r
DNA_BASES
DNA_ALPHABET
d <- DNAString("TTGAAA-CTC-N")
length(d)
alphabet(d)                   # DNA_ALPHABET
alphabet(d, baseOnly=TRUE)    # DNA_BASES
```

---

**findPalindromes**

Searching a sequence for palindromes or complemented palindromes

Description

The `findPalindromes` and `findComplementedPalindromes` functions can be used to find palindromic or complemented palindromic regions in a sequence. `palindromeArmLength`, `palindromeLeftArm`, `palindromeRightArm`, `complementedPalindromeArmLength`, `complementedPalindromeLeftArm` and `complementedPalindromeRightArm` are utility functions for operating on palindromic or complemented palindromic sequences.
findPalindromes

Usage

findPalindromes(subject, min.armlength=4, max.looplength=1, min.looplength=0, palindromArmLength(x, max.mismatch=0, ...)
palindromeLeftArm(x, max.mismatch=0, ...)
palindromeRightArm(x, max.mismatch=0, ...)

findComplementedPalindromes(subject, min.armlength=4, max.looplength=1, min.looplength=0, max.mismatch=0, ...)
complementedPalindromeArmLength(x, max.mismatch=0, ...)
complementedPalindromeLeftArm(x, max.mismatch=0, ...)
complementedPalindromeRightArm(x, max.mismatch=0, ...)

Arguments

subject An XString object containing the subject string, or an XStringViews object.
min.armlength An integer giving the minimum length of the arms of the palindromes (or complemented palindromes) to search for.
max.looplength An integer giving the maximum length of "the loop" (i.e. the sequence separating the 2 arms) of the palindromes (or complemented palindromes) to search for. Note that by default (max.looplength=1), findPalindromes will search for strict palindromes (or complemented palindromes) only.
min.looplength An integer giving the minimum length of "the loop" of the palindromes (or complemented palindromes) to search for.
max.mismatch The maximum number of mismatching letters allowed between the 2 arms of the palindromes (or complemented palindromes) to search for.
x An XString object containing a 2-arm palindrome or complemented palindrome, or an XStringViews object containing a set of 2-arm palindromes or complemented palindromes.
... Additional arguments to be passed to or from methods.

Details

The findPalindromes function finds palindromic substrings in a subject string. The palindromes that can be searched for are either strict palindromes or 2-arm palindromes (the former being a particular case of the latter) i.e. palindromes where the 2 arms are separated by an arbitrary sequence called "the loop".

Use the findComplementedPalindromes function to find complemented palindromic substrings in a DNAString subject (in a complemented palindrome the 2 arms are reverse-complementary sequences).

Value

findPalindromes and findComplementedPalindromes return an XStringViews object containing all palindromes (or complemented palindromes) found in subject (one view per palindromic substring found).
palindromeArmLength and complementedPalindromeArmLength return the arm length (integer) of the 2-arm palindrome (or complemented palindrome) x. It will raise an error if x has no arms. Note that any sequence could be considered a 2-arm palindrome if we were OK with arms
of length 0 but we are not: x must have arms of length greater or equal to 1 in order to be considered a 2-arm palindrome. The same apply to 2-arm complemented palindromes. When applied to an \texttt{XStringViews} object \(x\), \texttt{palindromeArmLength} and \texttt{complementedPalindromeArmLength} behave in a vectorized fashion by returning an integer vector of the same length as \(x\).

\texttt{palindromeLeftArm} and \texttt{complementedPalindromeLeftArm} return an object of the same class as the original object \(x\) and containing the left arm of \(x\).

\texttt{palindromeRightArm} does the same as \texttt{palindromeLeftArm} but on the right arm of \(x\).

Like \texttt{palindromeArmLength}, both \texttt{palindromeLeftArm} and \texttt{palindromeRightArm} will raise an error if \(x\) has no arms. Also, when applied to an \texttt{XStringViews} object \(x\), both behave in a vectorized fashion by returning an \texttt{XStringViews} object of the same length as \(x\).

Author(s)

H. Pages

See Also

\texttt{maskMotif,matchPattern,matchLRPatterns,matchProbePair, XStringViews-class, DNAString-class}

Examples

```r
## Note that complemented palindromes (like palindromes) can be nested
findComplementedPalindromes(DNAString("ACGTTNAACGT-ACGTTNAACGT"))

## A real use case
library(BSgenome.Dmelanogaster.UCSC.dm3)
chrX <- Dmelanogaster$chrX
chrX_pals <- findComplementedPalindromes(chrX, min.armlength=50, max.looplength=20)
complementedPalindromeArmLength(chrX_pals) # 251

## Of course, whitespaces matter
palindromeArmLength(BString("was it a car or a cat I saw"))

## Note that the 2 arms of a strict palindrome (or strict complemented
## palindrome) are equal to the full sequence.
palindromeLeftArm(BString("Delia saw I was aileD"))
complementedPalindromeLeftArm(DNAString("N-ACGTT-AACGT-N"))
palindromeLeftArm(DNAString("N-AAA-N-N-TTT-N"))
```

\section*{GENETIC_CODE}

\textit{The Standard Genetic Code}

Description

Two predefined objects (\texttt{GENETIC_CODE} and \texttt{RNA_GENETIC_CODE}) that represent The Standard Genetic Code.

Usage

\texttt{GENETIC_CODE}
\texttt{RNA_GENETIC_CODE}
Details

Formally, a genetic code is a mapping between tri-nucleotide sequences called codons, and amino acids.

The Standard Genetic Code (aka The Canonical Genetic Code, or simply The Genetic Code) is the particular mapping that encodes the vast majority of genes in nature.

GENETIC_CODE and RNA_GENETIC_CODE are predefined named character vectors that represent this mapping.

Value

GENETIC_CODE and RNA_GENETIC_CODE are both named character vectors of length 64 (the number of all possible tri-nucleotide sequences) where each element is a single letter representing either an amino acid or the stop codon "*" (aka termination codon).

The names of the GENETIC_CODE vector are the DNA codons i.e. the tri-nucleotide sequences (directed 5' to 3') that are assumed to belong to the "coding DNA strand" (aka "sense DNA strand" or "non-template DNA strand") of the gene.

The names of the RNA_GENETIC_CODE are the RNA codons i.e. the tri-nucleotide sequences (directed 5' to 3') that are assumed to belong to the mRNA of the gene.

Note that the values in the GENETIC_CODE and RNA_GENETIC_CODE vectors are the same, only their names are different. The names of the latter are those of the former where all occurrences of T (thymine) have been replaced by U (uracil).

Author(s)

H. Pages

References


See Also

AA_ALPHABET, AMINO_ACID_CODE, translate, trinucleotideFrequency, DNAString, RNAString, AASTring

Examples

GENETIC_CODE
GENETIC_CODE["ATG"] # codon ATG is translated into M (Methionine)
sort(table(GENETIC_CODE)) # the same amino acid can be encoded by 1 # to 6 different codons

RNA_GENETIC_CODE
all(GENETIC_CODE == RNA_GENETIC_CODE) # TRUE
**gregexpr2**

**A replacement for R standard gregexpr function**

**Description**

This is a replacement for the standard gregexpr function that does exact matching only. Standard gregexpr() misses matches when they are overlapping. The gregexpr2 function finds all matches but it only works in "fixed" mode i.e. for exact matching (regular expressions are not supported).

**Usage**

```r
gregexpr2(pattern, text)
```

**Arguments**

- **pattern** character string to be matched in the given character vector
- **text** a character vector where matches are sought

**Value**

A list of the same length as text each element of which is an integer vector as in gregexpr, except that the starting positions of all (even overlapping) matches are given. Note that, unlike gregexpr, gregexpr2 doesn't attach a "match.length" attribute to each element of the returned list because, since it only works in "fixed" mode, then all the matches have the length of the pattern. Another difference with gregexpr is that with gregexpr2, the pattern argument must be a single (non-NA, non-empty) string.

**Author(s)**

H. Pages

**See Also**

- `gregexpr`
- `matchPattern`

**Examples**

```r
gregexpr("aa", c("XaaaYaa", "a"), fixed=TRUE)
gregexpr2("aa", c("XaaaYaa", "a"))
```
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, PairwiseAlignedXStringSet-class

injectHardMask

Injecting a hard mask in a sequence

Description
injectHardMask allows the user to "fill" the masked regions of a sequence with an arbitrary letter (typically the "+" letter).

Usage
 injectHardMask(x, letter="+")

Arguments
 x A MaskedXString or XStringViews object.
 letter A single letter.
Details

The name of the `injectHardMask` function was chosen because of the primary use that it is intended for: converting a pile of active "soft masks" into a "hard mask". Here the pile of active "soft masks" refers to the active masks that have been put on top of a sequence. In Biostrings, the original sequence and the masks defined on top of it are bundled together in one of the dedicated containers for this: the `MaskedBString`, `MaskedDNAString`, `MaskedRNAString` and `MaskedAAString` containers (this is the `MaskedXString` family of containers). The original sequence is always stored unmodified in a `MaskedXString` object so no information is lost. This allows the user to activate/deactivate masks without having to worry about losing the letters that are in the regions that are masked/unmasked. Also this allows better memory management since the original sequence never needs to be copied, even when the set of active/inactive masks changes.

However, there are situations where the user might want to really get rid of the letters that are in some particular regions by replacing them with a junk letter (e.g. "+") that is guaranteed to not interfere with the analysis that s/he is currently doing. For example, it’s very likely that a set of motifs or short reads will not contain the "+" letter (this could easily be checked) so they will never hit the regions filled with "+". In a way, it’s like the regions filled with "+" were masked but we call this kind of masking "hard masking".

Some important differences between "soft" and "hard" masking:

- `injectHardMask` creates a (modified) copy of the original sequence. Using "soft masking" does not.
- A function that is "mask aware" like `alphabetFrequency` or `matchPattern` will really skip the masked regions when "soft masking" is used i.e. they will not walk thru the regions that are under active masks. This might lead to some speed improvements when a high percentage of the original sequence is masked. With "hard masking", the entire sequence is walked thru.
- Matches cannot span over masked regions with "soft masking". With "hard masking" they can.

Value

An `XString` object of the same length as the original object `x` if `x` is a `MaskedXString` object, or of the same length as `subject(x)` if it’s an `XStringViews` object.

Author(s)

H. Pages

See Also

`maskMotif`, `MaskedXString-class`, `replaceLetterAt`, `chartr`, `XString`, `XStringViews-class`

Examples

```r
## A. WITH AN XStringViews OBJECT
v2 <- Views("abCDefgHIJK", start=c(8, 3), end=c(14, 4))
injectHardMask(v2)
injectHardMask(v2, letter="=")

## B. WITH A MaskedXString OBJECT
mask0 <- Mask(mask.width=29, start=c(3, 10, 25), width=c(6, 8, 5))
```
x <- DNAString("ACACAACTAGATAGNACTNNGAGAGACGC")
masks(x) <- mask0
x
subject <- injectHardMask(x)

## Matches can span over masked regions with "hard masking":
matchPattern("ACggggggA", subject, max.mismatch=6)
## but not with "soft masking":
matchPattern("ACggggggA", x, max.mismatch=6)

---

**IUPAC_CODE_MAP**

The IUPAC Extended Genetic Alphabet

**Description**

The IUPAC_CODE_MAP named character vector contains the mapping from the IUPAC nucleotide ambiguity codes to their meaning.

The mergeIUPACLetters function provides the reverse mapping.

**Usage**

```r
IUPAC_CODE_MAP
mergeIUPACLetters(x)
```

**Arguments**

- `x`: A vector of non-empty character strings made of IUPAC letters.

**Details**

IUPAC nucleotide ambiguity codes are used for representing sequences of nucleotides where the exact nucleotides that occur at some given positions are not known with certainty.

**Value**

IUPAC_CODE_MAP is a named character vector where the names are the IUPAC nucleotide ambiguity codes and the values are their corresponding meanings. The meaning of each code is described by a string that enumerates the base letters ("A", "C", "G" or "T") associated with the code.

The value returned by mergeIUPACLetters is an unnamed character vector of the same length as its argument `x` where each element is an IUPAC nucleotide ambiguity code.

**Author(s)**

H. Pages

**References**

http://www.chick.manchester.ac.uk/SiteSeer/IUPAC_codes.html
See Also

DNAString, RNAString

Examples

IUPAC_CODE_MAP
some_iupac_codes <- c("R", "M", "G", "N", "V")
IUPAC_CODE_MAP[some_iupac_codes]
mergeIUPACLetters(IUPAC_CODE_MAP[some_iupac_codes])
mergeIUPACLetters(c("Ca", "Acc", "aA", "MAAmC", "gM", "AB", "bS", "mk"))

letterFrequency  Calculate the frequency of letters in a biological sequence, or the consensus matrix of a set of sequences

Description

Given a biological sequence (or a set of biological sequences), the alphabetFrequency function computes the frequency of each letter in the (base) alphabet. The consensusMatrix function computes the consensus matrix of a set of sequences, and the consensusString function creates the consensus sequence based on a 50% + 1 vote from the consensus matrix (using the "?" letter to represent the lack of consensus).

In this man page we call "DNA input" (or "RNA input") an XString, XStringSet, XStringViews or MaskedXString object of base type DNA (or RNA).

Usage

alphabetFrequency(x, baseOnly=FALSE, freq=FALSE, ...)
hasOnlyBaseLetters(x)
uniqueLetters(x)

## S4 method for signature 'character':
consensusMatrix(x, freq=FALSE)
## S4 method for signature 'XStringSet':
consensusMatrix(x,  
    baseOnly=FALSE, freq=FALSE, shift=0L, width=NULL)

## S4 method for signature 'matrix':
consensusString(x)
## S4 method for signature 'XStringSet':
consensusString(x, shift=0L, width=NULL)
## S4 method for signature 'ANY':
consensusString(x)

Arguments

x  An XString, XStringSet, XStringViews or MaskedXString object for alphabetFrequency and uniqueLetters.
DNA or RNA input for hasOnlyBaseLetters.
A character vector, or an \texttt{XStringSet} or \texttt{XStringViews} object for \texttt{consensusMatrix}.

A consensus matrix (as returned by \texttt{consensusMatrix}), or an \texttt{XStringSet} or \texttt{XStringViews} object for \texttt{consensusString}.

\texttt{baseOnly} \ TRUE \ or \ \FALSE. If \texttt{TRUE}, the returned vector (or matrix) only contains the frequencies of the letters that belong to the "base" alphabet of \texttt{x} i.e. to the alphabet returned by \texttt{alphabet(x, baseOnly=TRUE)}. Note that when \texttt{x} is not a DNA or RNA input, then specifying \texttt{baseOnly} has no effect.

\texttt{freq} \ If \texttt{TRUE} then relative frequencies are reported, otherwise counts (the default).

\texttt{...} \ Further arguments to be passed to or from other methods. For the \texttt{XStringViews} and \texttt{XStringSet} methods, the \texttt{collapse} argument is accepted.

\texttt{shift} \ An integer vector (recycled to the length of \texttt{x}) specifying how each sequence in \texttt{x} should be (horizontally) shifted with respect to the first column of the consensus matrix to be returned. By default (\texttt{shift=0}), each sequence in \texttt{x} has its first letter aligned with the first column of the matrix. A positive \texttt{shift} value means that the corresponding sequence must be shifted to the right, and a negative \texttt{shift} value that it must be shifted to the left. For example, a shift of 5 means that it must be shifted 5 positions to the right (i.e. the first letter in the sequence must be aligned with the 6th column of the matrix), and a shift of -3 means that it must be shifted 3 positions to the left (i.e. the 4th letter in the sequence must be aligned with the first column of the matrix).

\texttt{width} \ The number of columns of the returned matrix for the \texttt{consensusMatrix} method for \texttt{XStringSet} objects. When \texttt{width=NULL} (the default), then this method returns a matrix that has just enough columns to have its last column aligned with the rightmost letter of all the sequences in \texttt{x} after those sequences have been shifted (see the \texttt{shift} argument above). This ensures that any wider consensus matrix would be a "padded with zeros" version of the matrix returned when \texttt{width=NULL}.

The length of the returned sequence for the \texttt{consensusString} method for \texttt{XStringSet} objects.

Details

\texttt{alphabetFrequency} is a generic function defined in the Biostrings package.

Value

\texttt{alphabetFrequency} returns a numeric vector when \texttt{x} is an \texttt{XString} or \texttt{MaskedXString} object. When \texttt{x} is an \texttt{XStringSet} or \texttt{XStringViews} object, then it returns a numeric matrix with length(\texttt{x}) rows where the \texttt{i}-th row contains the frequencies for \texttt{x[i]}. If \texttt{x} is a DNA or RNA input, then the returned vector is named with the letters in the alphabet. If the \texttt{baseOnly} argument is \texttt{TRUE}, then the returned vector has only 5 elements: 4 elements corresponding to the 4 nucleotides + the 'other' element.

\texttt{hasOnlyBaseLetters} returns \texttt{TRUE} or \texttt{FALSE} indicating whether or not \texttt{x} contains only base letters (i.e. As, Cs, Gs and Ts for DNA input and As, Cs, Gs and Us for RNA input).

\texttt{uniqueLetters} returns a vector of 1-letter or empty strings. The empty string is used to represent the nul character if \texttt{x} happens to contain any. Note that this can only happen if the base class of \texttt{x} is \texttt{BString}.

An integer matrix with letters as row names for \texttt{consensusMatrix}.

A standard character string for \texttt{consensusString}. 

Author(s)

H. Pages and P. Aboyoun

See Also

alphabet, coverage, oligonucleotideFrequency, countPDict, XString-class, XStringSet-class, XStringViews-class, MaskedXString-class, strsplit

Examples

## A. BASIC alphabetFrequency() EXAMPLES

```r
data(yeastSEQCHR1)
yeast1 <- DNAString(yeastSEQCHR1)
alphabetFrequency(yeast1)
alphabetFrequency(yeast1, baseOnly=TRUE)
hasOnlyBaseLetters(yeast1)
uniqueLetters(yeast1)
```

## With input made of multiple sequences:

```r
library(drosophila2probe)
probes <- DNAStringSet(drosophila2probe$sequence)
alphabetFrequency(probes[1:50], baseOnly=TRUE)
alphabetFrequency(probes, baseOnly=TRUE, collapse=TRUE)
```

## B. consensus*() EXAMPLES

```r
## Read in ORF data:
file <- system.file("extdata", "someORF.fa", package="Biostrings")
orf <- read.DNAStringSet(file, "fasta")
## To illustrate, the following example assumes the ORF data
## to be aligned for the first 10 positions (patently false):
orf10 <- DNAStringSet(orf, end=10)
consensusMatrix(orf10, baseOnly=TRUE)
## The following example assumes the first 10 positions to be aligned
## after some incremental shifting to the right (patently false):
consensusMatrix(orf10, baseOnly=TRUE, shift=0:6)
consensusMatrix(orf10, baseOnly=TRUE, shift=0:6, width=10)
## For the character matrix containing the "exploded" representation
## of the strings, do:
as.matrix(orf10, use.names=FALSE)
## consensusMatrix() can be used to just compute the alphabet frequency
## for each position in the input sequences:
consensusMatrix(probes, baseOnly=TRUE)
## After sorting, the first 5 probes might look similar (at least on
## their first bases):
consensusString(sort(probes)[1:5])
```
## C. RELATIONSHIP BETWEEN consensusMatrix() AND coverage()

Applying colSums() on a consensus matrix gives the coverage that
would be obtained by piling up (after shifting) the input sequences
on top of an (imaginary) reference sequence:

```r
cm <- consensusMatrix(orf10, shift=0:6, width=10)
colSums(cm)
```

## Note that this coverage can also be obtained with:

```r
as.integer(coverage(IRanges(rep(1, length(orf)), width(orf)), shift=0:6, width=10))
```

### Description

Extract a substring from a string by picking up individual letters by their position.

### Usage

```r
letter(x, i)
```

### Arguments

- `x`: A character vector, or an `XString`, `XStringViews` or `MaskedXString` object.
- `i`: An integer vector with no NAs.

### Details

Unlike with the `substr` or `substring` functions, `i` must contain valid positions.

### Value

A character vector of length 1 when `x` is an `XString` or `MaskedXString` object (the masks are ignored for the latter).

A character vector of the same length as `x` when `x` is a character vector or an `XStringViews` object.

Note that, because `i` must contain valid positions, all non-NA elements in the result are guaranteed to have exactly `length(i)` characters.

### See Also

`subseq`, `XString-class`, `XStringViews-class`, `MaskedXString-class`

### Examples

```r
x <- c("abcd", "ABC")
i <- c(3, 1, 1, 2, 1)

## With a character vector:
letter(x[1], 3:1)
letter(x, 3)
```
longestConsecutive

Obtain the length of the longest substring containing only 'letter'

Description

This function accepts a character vector and computes the length of the longest substring containing only \texttt{letter} for each element of \texttt{x}.

Usage

\begin{verbatim}
longestConsecutive(seq, letter)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{seq} Character vector.
  \item \texttt{letter} Character vector of length 1, containing one single character.
\end{itemize}

Details

The elements of \texttt{x} can be in upper case, lower case or mixed. NAs are handled.

Value

An integer vector of the same length as \texttt{x}.

Author(s)

W. Huber

See Also

\begin{verbatim}
complementSeq, basecontent, reverseSeq
\end{verbatim}

Examples

\begin{verbatim}
v = c("AAACTGTGFG", "GGGAATT", "CCAAAAAAAAAATT")
longestConsecutive(v, "A")
\end{verbatim}
MaskedXString-class

Description

The MaskedBString, MaskedDNAString, MaskedRNAString and MaskedAAString classes are containers for storing masked sequences.

All those containers derive directly (and with no additional slots) from the MaskedXString virtual class.

Details

In Biostrings, a pile of masks can be put on top of a sequence. A pile of masks is represented by a MaskCollection object and the sequence by an XString object. A MaskedXString object is the result of bundling them together in a single object.

Note that, no matter what masks are put on top of it, the original sequence is always stored unmodified in a MaskedXString object. This allows the user to activate/deactivate masks without having to worry about losing the information stored in the masked/unmasked regions. Also this allows efficient memory management since the original sequence never needs to be copied (modifying it would require to make a copy of it first - sequences cannot and should never be modified in place in Biostrings), even when the set of active/inactive masks changes.

Accessor methods

In the code snippets below, \( x \) is a MaskedXString object. For \( \text{masks}(x) \) and \( \text{masks}(x) \leftarrow y \), it can also be an XString object and \( y \) must be NULL or a MaskCollection object.

- \( \text{unmasked}(x) \): Turns \( x \) into an XString object by dropping the masks.
- \( \text{masks}(x) \leftarrow y \): If \( x \) is an XString object and \( y \) is NULL, then this doesn’t do anything.
  
  If \( x \) is an XString object and \( y \) is a MaskCollection object, then this turns \( x \) into a MaskedXString object by putting the masks in \( y \) on top of it.
  
  If \( x \) is a MaskedXString object and \( y \) is NULL, then this is equivalent to \( x \leftarrow \text{unmasked}(x) \).
  
  If \( x \) is a MaskedXString object and \( y \) is a MaskCollection object, then this replaces the masks currently on top of \( x \) by the masks in \( y \).

- \( \text{alphabet}(x) \): Equivalent to \( \text{alphabet}(\text{unmasked}(x)) \). See \( \text{?alphabet} \) for more information.

- \( \text{length}(x) \): Equivalent to \( \text{length}(\text{unmasked}(x)) \). See \( \text{?length,XString-method} \) for more information.

"maskedwidth" and related methods

In the code snippets below, \( x \) is a MaskedXString object.

- \( \text{maskedwidth}(x) \): Get the number of masked letters in \( x \). A letter is considered masked iff it’s masked by at least one active mask.

- \( \text{maskedratio}(x) \): Equivalent to \( \text{maskedwidth}(x) \) / \( \text{length}(x) \).

- \( \text{nchar}(x) \): Equivalent to \( \text{length}(x) - \text{maskedwidth}(x) \).
Coercion

In the code snippets below, \( x \) is a MaskedXString object.

\[
as(x, "XStringViews") : \text{Turns } x \text{ into an } XStringViews \text{ object where the views are the unmasked regions of the original sequence ("unmasked" means not masked by at least one active mask).}
\]

Other methods

In the code snippets below, \( x \) is a MaskedXString object.

\[
reduce(x) : \text{Reduce the set of masks in } x \text{ to a single mask made of all active masks.}
\]

\[
gaps(x) : \text{Reverses all the masks i.e. each mask is replaced by a mask where previously unmasked regions are now masked and previously masked regions are now unmasked.}
\]

Author(s)

H. Pages

See Also

maskMotif, injectHardMask, alphabetFrequency, reverse, MaskedXString-method, XString-class, MaskCollection-class, XStringViews-class, IRanges-utils

Examples

```r
## A. MASKING BY POSITION
mask0 <- Mask(mask.width=29, start=c(3, 10, 25), width=c(6, 8, 5))
x <- DNAString("ACACAACTAGATAGNACTNNGAGAGAGC")
length(x) # same as width(mask0)
nchar(x) # same as length(x)
masks(x) <- mask0
x
length(x) # has not changed
nchar(x) # has changed
gaps(x)

## Prepare a MaskCollection object of 3 masks ('mymasks') by running the examples in the man page for these objects:
example(MaskCollection, package="IRanges")

## Put it on 'x':

masks(x) <- mymasks
x

alphabetFrequency(x)

## Deactivate all masks:
active(masks(x)) <- FALSE
x

## Activate mask "C":
active(masks(x))['C'] <- TRUE
x
```
## Turn MaskedXString object into an XStringViews object:

as(x, "XStringViews")

## Drop the masks:

masks(x) <- NULL

x

alphabetFrequency(x)

## B. MASKING BY CONTENT

## See ?maskMotif for masking by content

---

**maskMotif**

**Masking by content (or by position)**

### Description

Functions for masking a sequence by content (or by position).

### Usage

```r
maskMotif(x, motif, min.block.width=1)
mask(x, start=NA, end=NA, pattern)
```

### Arguments

- **x**: The sequence to mask.
- **motif**: The motif to mask in the sequence.
- **min.block.width**: The minimum width of the blocks to mask.
- **start**: An integer vector containing the starting positions of the regions to mask.
- **end**: An integer vector containing the ending positions of the regions to mask.
- **pattern**: The motif to mask in the sequence.

### Value

A `MaskedXString` object for `maskMotif` and an `XStringViews` object for `mask`.

### Author(s)

H. Pages

### See Also

`read.Mask`, `XString-class`, `MaskedXString-class`, `XStringViews-class`, `MaskCollection-class`
Examples

```r
## EXAMPLE 1

maskMotif(BString("AbcbcbEEE"), "bcb")
maskMotif(BString("AbcbcbEEE"), "bcb")

## maskMotif() can be used in an incremental way to mask more than 1
## motif. Note that maskMotif() does not try to mask again what's
## already masked (i.e. the new mask will never overlaps with the
## previous masks) so the order in which the motifs are masked actually
## matters as it will affect the total set of masked positions.
x0 <- BString("AbcbEEEbcbbEcbcbcb")
x1 <- maskMotif(x0, "E")
x1
x2 <- maskMotif(x1, "bcb")
x2
x3 <- maskMotif(x2, "b")
x3

## Note that inverting the order in which "b" and "bcb" are masked would
## lead to a different final set of masked positions.
## Also note that the order doesn't matter if the motifs to mask don't
## overlap (we assume that the motifs are unique) i.e. if the prefix of
## each motif is not the suffix of any other motif. This is of course
## the case when all the motifs have only 1 letter.

## EXAMPLE 2

x <- DNAString("ACACAAGTAGATAAGNAGAGACG")

## Mask the N-blocks
x1 <- maskMotif(x, "N")
x1

## Mask the AC-blocks
x2 <- maskMotif(x1, "AC")
x2

## Mask the GA-blocks
x3 <- maskMotif(x2, "GA", min.block.width=5)
x3 # masks 2 and 3 overlap

## EXAMPLE 3

library(BSgenome.Dmelanogaster.UCSC.dm3)
chrU <- Dmelanogaster$chrU
```
chrU
alphabetFrequency(chrU)
chrU <- maskMotif(chrU, "N")
chrU
alphabetFrequency(chrU)
as(chrU, "XStringViews")
as(gaps(chrU), "XStringViews")

mask2 <- Mask(mask.width=length(chrU), start=c(50000, 350000, 543900), width=25000)
names(mask2) <- "some ugly regions"
masks(chrU) <- append(masks(chrU), mask2)
chrU
as(chrU, "XStringViews")
as(gaps(chrU), "XStringViews")

## EXAMPLE 4
## Note that unlike maskMotif(), mask() returns an XStringViews object!

## masking "by position"
mask("AxyxyxBC", 2, 6)

## masking "by content"
mask("AxyxyxBC", "xyx")
noN_chrU <- mask(chrU, "N")
noN_chrU
alphabetFrequency(noN_chrU, collapse=TRUE)

---

**matchLRPatterns**

*Find paired matches in a sequence*

**Description**

The **matchLRPatterns** function finds paired matches in a sequence i.e. matches specified by a left pattern, a right pattern and a maximum distance between the left pattern and the right pattern.

**Usage**

```r
matchLRPatterns(Lpattern, Rpattern, max.ngaps, subject,
    max.Lmismatch=0, max.Rmismatch=0,
    with.Lindels=FALSE, with.Rindels=FALSE,
    Lfixed=TRUE, Rfixed=TRUE)
```

**Arguments**

- **Lpattern**  The left part of the pattern.
- **Rpattern**  The right part of the pattern.
- **max.ngaps** The max number of gaps in the middle i.e the max distance between the left and right parts of the pattern.
- **subject**   An XString, XStringViews or MaskedXString object containing the target sequence.
matchLRPatterns

max.Lmismatch

The maximum number of mismatching letters allowed in the left part of the pattern. If non-zero, an inexact matching algorithm is used (see the matchPattern function for more information).

max.Rmismatch

Same as max.Lmismatch but for the right part of the pattern.

with.Lindels

If TRUE then indels are allowed in the left part of the pattern. In that case max.Lmismatch is interpreted as the maximum "edit distance" allowed in the left part of the pattern.

See the with.indels argument of the matchPattern function for more information.

with.Rindels

Same as with.Lindels but for the right part of the pattern.

Lfixed

Only with a DNAString or RNAString subject can a Lfixed value other than the default (TRUE) be used.

With Lfixed=FALSE, ambiguities (i.e. letters from the IUPAC Extended Genetic Alphabet (see IUPAC_CODE_MAP) that are not from the base alphabet) in the left pattern _and_ in the subject are interpreted as wildcards i.e. they match any letter that they stand for.

See the fixed argument of the matchPattern function for more information.

Rfixed

Same as Lfixed but for the right part of the pattern.

Value

An XStringViews object containing all the matches, even when they are overlapping (see the examples below), and where the matches are ordered from left to right (i.e. by ascending starting position).

Author(s)

H. Pages

See Also

matchPattern, matchProbePair, trimLRPatterns, findPalindromes, reverseComplement, XString-class, XStringViews-class, MaskedXString-class

Examples

library(BSgenome.Dmelanogaster.UCSC.dm3)
subject <- Dmelanogaster$chr3R
Lpattern <- "AGCTCCGAG"
Rpattern <- "TTGTTCACA"
matchLRPatterns(Lpattern, Rpattern, 500, subject) # 1 match

## Note that matchLRPatterns() will return all matches, even when they are
## overlapping:
subject <- DNAString("AAATTAACCCTT")
matchLRPatterns("AA", "TT", 0, subject) # 1 match
matchLRPatterns("AA", "TT", 1, subject) # 2 matches
matchLRPatterns("AA", "TT", 3, subject) # 3 matches
matchLRPatterns("AA", "TT", 7, subject) # 4 matches
**matchPattern**

String searching functions

**Description**

A set of functions for finding all the occurrences (aka "matches" or "hits") of a given pattern (typically short) in a (typically long) reference sequence or set of reference sequences (aka the subject).

**Usage**

```r
matchPattern(pattern, subject, algorithm="auto", max.mismatch=0, with.indels=FALSE, fixed=TRUE)
countPattern(pattern, subject, algorithm="auto", max.mismatch=0, with.indels=FALSE, fixed=TRUE)
vmatchPattern(pattern, subject, algorithm="auto", max.mismatch=0, with.indels=FALSE, fixed=TRUE)
vcountPattern(pattern, subject, algorithm="auto", max.mismatch=0, with.indels=FALSE, fixed=TRUE)
```

**Arguments**

- **pattern**: The pattern string.
- **subject**: An `XString`, `XStringViews` or `MaskedXString` object for `matchPattern` and `countPattern`. An `XStringSet` or `XStringViews` object for `vmatchPattern` and `vcountPattern`.
- **algorithm**: One of the following: "auto", "naive-exact", "naive-inexact", "boyer-moore", "shift-or" or "indels".
- **max.mismatch**: The maximum number of mismatching letters allowed (see `isMatchingAt` for the details). If non-zero, an inexact matching algorithm is used.
- **with.indels**: If TRUE then indels are allowed. In that case `max.mismatch` is interpreted as the maximum "edit distance" allowed between the pattern and a match. Note that in order to avoid pollution by redundant matches, only the "best local matches" are returned. Roughly speaking, a "best local match" is a match that is locally both the closest (to the pattern P) and the shortest. More precisely, a substring S' of the subject S is a "best local match" iff:

  (a) \( \text{nedit}(P, S') \leq \text{max.mismatch} \)
  (b) for every substring S1 of S': \( \text{nedit}(P, S1) > \text{nedit}(P, S') \)
  (c) for every substring S2 of S that contains S': \( \text{nedit}(P, S2) \leq \text{nedit}(P, S') \)

One nice property of "best local matches" is that their first and last letters are guaranteed to be aligned with letters in P (i.e. they match letters in P).

- **fixed**: If FALSE then IUPAC extended letters are interpreted as ambiguities (see `isMatchingAt` for the details).
matchPattern

Details

Available algorithms are: “naive exact”, “naive inexact”, “Boyer-Moore-like”, “shift-or” and “indels”. Not all of them can be used in all situations: restrictions depend on the length of the pattern, the class of the subject, and the values of \texttt{max.mismatch}, \texttt{with.indels} and \texttt{fixed}. All those parameters form the search criteria.

Note that the choice of an algorithm is not part of the search criteria. This is because algorithms are interchangeable, that is, if 2 different algorithms are compatible with a given search criteria, then choosing one over the other will not affect the result (but will most likely affect the performance). So there is no "wrong choice" of algorithm (strictly speaking).

Using \texttt{algorithm="auto"} is recommended because then the fastest algorithm will automatically be picked up among the set of compatible algorithms (if there is more than one).

Value

An \texttt{XStringViews} object for \texttt{matchPattern}.
A single integer for \texttt{countPattern}.
An \texttt{MIndex} object for \texttt{vmatchPattern}.
An integer vector for \texttt{vcountPattern}, with each element in the vector corresponding to the number of matches in the corresponding element of \texttt{subject}.

Note

Use \texttt{matchPDict} if you need to match a (big) set of patterns against a reference sequence.
Use \texttt{pairwiseAlignment} if you need to solve a (Needleman-Wunsch) global alignment, a (Smith-Waterman) local alignment, or an (ends-free) overlap alignment problem.

See Also

\texttt{matchPDict}, \texttt{pairwiseAlignment}, \texttt{ismatchingAt}, \texttt{mismatch}, \texttt{matchLRPatterns}, \texttt{matchProbePair}, \texttt{maskMotif}, \texttt{alphabetFrequency}, \texttt{XStringViews-class}, \texttt{MIndex-class}

Examples

```r
## A simple inexact matching example with a short subject:
x <- DNAString("AAGCGCGATATG")
m1 <- matchPattern("GCNNNAT", x)
m1
m2 <- matchPattern("GCNNNAT", x, fixed=FALSE)
m2
as.matrix(m2)

## With DNA sequence of yeast chromosome number 1:
data(yeastSEQCHR1)
yeast1 <- DNAString(yeastSEQCHR1)
PpiI <- "GAACNNNNNCTC" # a restriction enzyme pattern
match1.PpiI <- matchPattern(PpiI, yeast1, fixed=FALSE)
match2.PpiI <- matchPattern(PpiI, yeast1, max.mismatch=1, fixed=FALSE)

## With a genome containing isolated Ns:
```
matchPattern

library(BSgenome.Celegans.UCSC.ce2)
chrII <- Celegans["chrII"]
alphabetFrequency(chrII)
mismatchPattern("N", chrII)
mismatchPattern("TGGGTGTCTTT", chrII)  # no match
mismatchPattern("TGGGTGTCTTT", chrII, fixed=FALSE)  # 1 match

## Using wildcards ("N") in the pattern on a genome containing N-blocks:
library(BSgenome.Dmelanogaster.UCSC.dm3)
chrX <- maskMotif(Dmelanogaster$chrX, "N")
as(chrX, "XStringViews")  # 4 non masked regions
mismatchPattern("TTTATGNTTGGTA", chrX, fixed=FALSE)
## Can also be achieved with no mask:
masks(chrX) <- NULL
mismatchPattern("TTTATGNTTGGTA", chrX, fixed="subject")

## B. vmatchPattern()/vcountPattern()

Ebox <- DNAString("CANNTG")
subject <- Celegans$upstream5000
mindex <- vmatchPattern(Ebox, subject, fixed=FALSE)
count_index <- countIndex(mindex)  # Get the number of matches per
    # subject element.

sum(count_index)  # Total number of matches.
table(count_index)
i0 <- which(count_index == max(count_index))  # The subject element with most matches.

## The matches in 'subject[i0]' as an IRanges object:

mindex[[i0]]
## The matches in 'subject[i0]' as an XStringViews object:
Views(subject[[i0]], mindex[[i0]])

## C. WITH INDELS

library(BSgenome.Celegans.UCSC.ce2)
library(BSgenome.Dmelanogaster.UCSC.dm3)
pattern <- DNAString("ACGGACCTAATGTTATC")
subject <- Celegans$chrI

## Allowing up to 2 mismatching letters doesn't give any match:
mismatchPattern(pattern, subject, max.mismatch=2)

## But allowing up to 2 edit operations gives 3 matches:

system.time(m <- mismatchPattern(pattern, subject, max.mismatch=2, with.indels=TRUE))
m

## pairwiseAlignment() returns the (first) best match only:
if (interactive()) {
    mat <- nucleotideSubstitutionMatrix(match=1, mismatch=0, baseOnly=TRUE)
    Note that this call to pairwiseAlignment() will need to
    allocate 733.5 Mb of memory (i.e. length(pattern) * length(subject)
    * 3 bytes).
    system.time(pwa <- pairwiseAlignment(pattern, subject, type="local",
        substitutionMatrix=mat,

matchPDict

Searching a sequence for patterns stored in a preprocessed dictionary

Description

A set of functions for finding all the occurrences (aka "matches" or "hits") of a set of patterns (aka the dictionary) in a reference sequence or set of reference sequences (aka the subject).

The following functions differ in what they return: `matchPDict` returns the "where" information i.e. the positions in the subject of all the occurrences of every pattern; `countPDict` returns the "how many times" information i.e. the number of occurrences for each pattern; and `whichPDict` returns the "who" information i.e. which patterns in the preprocessed dictionary have at least one match. `vcountPDict` is similar to `countPDict` but it works on a set of reference sequences in a vectorized fashion.

This man page shows how to use these functions for exact matching of a constant width dictionary i.e. a dictionary where all the patterns have the same length (same number of nucleotides).

See `?matchPDict-inexact` for how to use these functions for inexact matching or when the original dictionary has a variable width.

Usage

```r
matchPDict(pdict, subject, algorithm="auto",
            max.mismatch=0, fixed=TRUE, verbose=FALSE)
countPDict(pdict, subject, algorithm="auto",
            max.mismatch=0, fixed=TRUE, verbose=FALSE)
whichPDict(pdict, subject, algorithm="auto",
            max.mismatch=0, fixed=TRUE, verbose=FALSE)
```

```r
vcountPDict(pdict, subject, algorithm="auto",
            max.mismatch=0, fixed=TRUE,
            collapse=FALSE, weight=1L, verbose=FALSE)
```
**matchPDict**

**Arguments**

- **pdict**: A `PDict` object containing the preprocessed dictionary.
- **subject**: An `XString` or `MaskedXString` object containing the subject sequence for `matchPDict`, `countPDict` and `whichPDict`. An `XStringSet` object containing the subject sequences for `vcountPDict`. For now, only subjects of base class `DNAString` are supported.
- **algorithm**: Not supported yet.
- **max.mismatch**: The maximum number of mismatching letters allowed (see `?isMatching` for the details). This man page focuses on exact matching of a constant width dictionary so `max.mismatch=0` in the examples below. See `¿matchPDict-inexact` for inexact matching.
- **fixed**: If `FALSE` then IUPAC extended letters are interpreted as ambiguities (see `?isMatching` for the details). This man page focuses on exact matching of a constant width dictionary so `fixed=TRUE` in the examples below. See `¿matchPDict-inexact` for inexact matching.
- **verbose**: `TRUE` or `FALSE`.
- **collapse,** `weight`:
  - `collapse` must be `FALSE`, 1, or 2.
  - If `collapse=FALSE` (the default), then `weight` is ignored and `vcountPDict` returns the full matrix of counts (`M0`). If `collapse=1`, then `M0` is collapsed "horizontally" i.e. it is turned into a vector with length equal to `length(pdict)`. If `weight=1L` (the default), then this vector is defined by `rowSums(M0)`. If `collapse=2`, then `M0` is collapsed "vertically" i.e. it is turned into a vector with length equal to `length(subject)`. If `weight=1L` (the default), then this vector is defined by `colSums(M0)`. If `collapse=1` or `collapse=2`, then the elements in `subject (collapse=1)` or in `pdict (collapse=2)` can be weighted thru the `weight` argument. In that case, the returned vector is defined by `M0 %% rep(weight, length.out=length(subject))` and `rep(weight, length.out=length(pdict)) %% M0`, respectively.

**Details**

In this man page, we assume that you know how to preprocess a dictionary of DNA patterns that can then be used with `matchPDict`, `countPDict`, `whichPDict` or `vcountPDict`. Please see `?PDict` if you don’t.

When using `matchPDict`, `countPDict`, `whichPDict` or `vcountPDict` for exact matching of a constant width dictionary, the standard way to preprocess the original dictionary is by calling the `PDict` constructor on it with no extra arguments. This returns the preprocessed dictionary in a `PDict` object that can be used with any of the functions described here.

**Value**

If `M` denotes the number of patterns in the `pdict` argument (`M <- length(pdict)`), then `matchPDict` returns an `MIndex` object of length `M`, and `countPDict` an integer vector of length `M`.

`whichPDict` returns an integer vector made of the indices of the patterns in the `pdict` argument that have at least one match.

If `N` denotes the number of sequences in the `subject` argument (`N <- length(subject)`), then `vcountPDict` returns an integer matrix with `M` rows and `N` columns, unless the `collapse`
argument is used. In that case, depending on the type of weight, an integer or numeric vector is returned (see above for the details).

Author(s)

H. Pages

References


See Also

PDict-class, MIndex-class, matchPDict-inexact, isMatching, coverage, MIndex-method, matchPattern, alphabetFrequency, DNAString-class, DNAStringSet-class, XStringViews-class, MaskedDNAString-class

Examples

```r
## A SIMPLE EXAMPLE OF EXACT MATCHING

## Creating the pattern dictionary:
library(drosophila2probe)
dict0 <- DNAStringSet(drosophila2probe$sequence)
dict0 # The original dictionary.
length(dict0) # Hundreds of thousands of patterns.
pdict0 <- PDict(dict0) # Store the original dictionary in a PDict object (preprocessing).

## Using the pattern dictionary on chromosome 3R:
library(BSgenome.Dmelanogaster.UCSC.dm3)
chr3R <- Dmelanogaster$chr3R # Load chromosome 3R
chr3R
mi0 <- matchPDict(pdict0, chr3R) # Search...

## Looking at the matches:
start_index <- startIndex(mi0) # Get the start index.
length(start_index) # Same as the original dictionary.
start_index[[8220]] # Starts of the 8220th pattern.
end_index <- endIndex(mi0) # Get the end index.
end_index[[8220]] # Ends of the 8220th pattern.
count_index <- countIndex(mi0) # Get the number of matches per pattern.
count_index[[8220]] # Get the matches for the 8220th pattern.
m0[[8220]] # Equivalent to startIndex(mi0)[[8220]].
sum(count_index) # Total number of matches.
table(count_index)
i0 <- which(count_index == max(count_index))
pdict0[[i0]] # The pattern with most occurrences.
m0[[i0]] # Its matches as an IRanges object.
Views(chr3R, m0[[i0]]) # And as an XStringViews object.

## Get the coverage of the original subject:
```
\texttt{cov3R} <- as.integer(coverage(mi0, width=length(chr3R)))
max(cov3R)
mean(cov3R)
sum(cov3R != 0) / length(cov3R)  # Only 2.44\% of chr3R is covered.
if (interactive()) {
  plotCoverage <- function(cx, start, end) {
    plot.new()
    plot.window(c(start, end), c(0, 20))
    axis(1)
    axis(2)
    axis(4)
    lines(start:end, cx[start:end], type="l")
  }
  plotCoverage(cov3R, 27600000, 27900000)
}

## B. NAMING THE PATTERNS

## The names of the original patterns, if any, are propagated to the
## PDict and MIndex objects:
names(dict0) <- mkAllStrings(letters, 4)[seq_len(length(dict0))]
dict0
dict0["abcd"]
pdict0n <- PDict(dict0)
names(pdict0n)[1:30]
pdict0n["abcd"]
mi0n <- matchPDict(pdict0n, chr3R)
names(mi0n)[1:30]
mi0n["abcd"]

## This is particularly useful when unlisting an MIndex object:
unlist(mi0)[1:10]
unlist(mi0n)[1:10]  # keep track of where the matches are coming from

## C. PERFORMANCE

## If getting the number of matches is what matters only (without
## regarding their positions), then countPDict() will be faster,
## especially when there is a high number of matches:
count_index0 <- countPDict(pdict0, chr3R)
identical(count_index0, count_index)  # TRUE

if (interactive()) {
  ## What's the impact of the dictionary width on performance?
  ## Below is some code that can be used to figure out (will take a long
  ## time to run). For different widths of the original dictionary, we
  ## look at:
  ## o pptime: preprocessing time (in sec.) i.e. time needed for
  ## building the PDict object from the truncated input
  ## sequences;
  ## o nnodes: nb of nodes in the resulting Aho-Corasick tree;
}
getPDictStats <- function(dict, subject)
{
  ans_width <- width(dict[1])
  ans_pptime <- system.time(pdict <- PDict(dict))["elapsed"]
  pptb <- pdict@threeparts@pptb
  ans_nnodes <- length(pptb@nodes) %/%
    Biostrings:::.ACtree.ints_per_acnode(pptb)
  ans_nupatt <- sum(!duplicated(pdict))
  ans_matchtime <- system.time(
    mi0 <- matchPDict(pdict, subject)
  )["elapsed"]
  ans_totalcount <- sum(countIndex(mi0))
}
stats <- lapply(6:25,
  function(width)
    getPDictStats(DNAStringSet(dict0, end=width), chr3R))
stats <- data.frame(do.call(rbind, stats))
stats

stats <- lapply(6:25,
  function(width)
    getPDictStats(DNAStringSet(dict0, end=width), chr3R))
stats <- data.frame(do.call(rbind, stats))
stats

## D. vcountPDict()
##

subject <- Dmelanogaster$upstream1000[1:200]
subject <- Dmelanogaster$upstream1000[1:200]

mat1 <- vcountPDict(pdict0, subject)

mat1 <- vcountPDict(pdict0, subject)
dim(mat1) # length(pdict0) x length(subject)
dim(mat1) # length(pdict0) x length(subject)
nhit_per_probe <- rowSums(mat1)
nhit_per_probe <- rowSums(mat1)
table(nhit_per_probe)
table(nhit_per_probe)

# Without vcountPDict(), 'mat1' could have been computed with:
#
mat2 <- sapply(unname(subject), function(x) countPDict(pdict0, x))
identical(mat1, mat2) # TRUE
#
# but using vcountPDict() is faster (10x or more, depending of the
# average length of the sequences in 'subject').

if (interactive()) {
  ## This will fail (with message "allocMatrix: too many elements
  ## specified") because, on most platforms, vectors and matrices in R
  ## are limited to 2^31 elements:
  subject <- Dmelanogaster$upstream1000
  vcountPDict(pdict0, subject)
  length(pdict0) * length(Dmelanogaster$upstream1000)
  1 * length(pdict0) * length(Dmelanogaster$upstream1000) # > 2^31
  ## But this will work:
  nhit_per_seq <- vcountPDict(pdict0, subject, collapse=2)
  nhit_per_seq <- vcountPDict(pdict0, subject, collapse=2)
matchPDict-inexact

Inexact matching with matchPDict()/countPDict()/whichPDict()

Description

The matchPDict, countPDict and whichPDict functions efficiently find the occurrences in a text (the subject) of all patterns stored in a preprocessed dictionary.

This man page shows how to use these functions for inexact matching or when the original dictionary has a variable width.

See ?matchPDict for how to use these functions for exact matching of a constant width dictionary i.e. a dictionary where all the patterns have the same length (same number of nucleotides).

Details

In this man page, we assume that you know how to preprocess a dictionary of DNA patterns that can then be used with matchPDict, countPDict or \( \text{\textbackslash code\{whichPDict\}} \). Please see ?PDict if you don’t.

When using matchPDict, countPDict or whichPDict for inexact matching or when the original dictionary has a variable width, a Trusted Band must be defined during the preprocessing step. This is done thru the \texttt{tb.start}, \texttt{tb.end} and \texttt{tb.width} arguments of the \texttt{PDict} constructor (see ?PDict for the details).
Then `matchPDict/countPDict/whichPDict` can be called with a null or non-null `max.mismatch` value and the search for exact or inexact matches happens in 2 steps: (1) find all the exact matches of all the elements in the Trusted Band; then (2) for each element in the Trusted Band that has at least one exact match, compare the head and the tail of this element with the flanking sequences of the matches found in (1).

Note that the number of exact matches found in (1) will decrease exponentially with the width of the Trusted Band. Here is a simple guideline in order to get reasonably good performance: if `TBW` is the width of the Trusted Band (`TBW <- tb.width(pdict)`) and L the number of letters in the subject (`L <- nchar(subject)`), then \( \frac{L}{4^{TBW}} \) should be kept as small as possible, typically < 10 or 20.

In addition, when a Trusted Band has been defined during preprocessing, then `matchPDict/countPDict/whichPDict` can be called with `fixed=FALSE`. In this case, IUPAC extended letters in the head or the tail of the `PDict` object are treated as ambiguities.

Author(s)

H. Pages

References


See Also

`PDict-class`, `MIndex-class`, `matchPDict`

Examples

```r
## A. USING AN EXPLICIT TRUSTED BAND FOR EXACT OR INEXACT MATCHING

library(drosophila2probe)
dict0 <- DNAStringSet(drosophila2probe$sequence)
dict0 # the original dictionary

dict9 <- PDict(dict0, tb.end=9)
dict9
tail(dict9)
sum(duplicated(dict9))
table(patternFrequency(dict9))

table(countPDict(dict9, chr3R, max.mismatch=1))
table(countPDict(dict9, chr3R, max.mismatch=3))
table(countPDict(dict9, chr3R, max.mismatch=5))

## B. COMPARISON WITH EXACT MATCHING
```

When the original dictionary is of constant width, exact matching
(i.e., 'max.mismatch=0' and 'fixed=TRUE') will be more efficient with
a full-width Trusted Band (i.e., a Trusted Band that covers the entire
dictionary) than with a Trusted Band of width < width(dict0).

```r
pdict0 <- PDict(dict0)
count0 <- countPDict(pdict0, chr3R)
count0b <- countPDict(pdict9, chr3R, max.mismatch=0)
identical(count0b, count0) # TRUE
```

C. USING AN EXPLICIT TRUSTED BAND TO HANDLE A VARIABLE WIDTH
DICTIONARY

Here is a small variable width dictionary that contains IUPAC
ambiguities (pattern 1 and 3 contain an N):
```
dict0 <- DNAStringSet(c("TACCNG", "TAGT", "CGGNT", "AGTAG", "TAGT"))
```
(Note that pattern 2 and 5 are identical.)

If we only want to do exact matching, then it is recommended to use
the widest possible Trusted Band i.e., to set its width to
'min(width(dict0))' because this is what will give the best
performance. However, when 'dict0' contains IUPAC ambiguities (like
in our case), it could be that one of them is falling into the
Trusted Band so we get an error (only base letters can go in the
Trusted Band for now):
```
Not run:
PDict(dict0, tb.end=min(width(dict0))) # Error!
```

In our case, the Trusted Band cannot be wider than 3:
pdict <- PDict(dict0, tb.end=3)
tail(pdict)
```
subject <- DNAString("TAGTACCAGTTTCGGG")
```
```
m <- matchPDict(pdict, subject)  # pattern 2 and 5 have 1 exact match
m[[2]]
```

We can take advantage of the fact that our Trusted Band doesn't cover
the entire dictionary to allow inexact matching on the uncovered parts
(the tail in our case):
```
WARNING: Support for 'fixed=FALSE' is currently broken (FIXME)
```
```
Not run:
m <- matchPDict(pdict, subject, fixed=FALSE)
countIndex(m) # now pattern 1 has 1 match too
m[[1]]
```
```
End(Not run)
```

```
m <- matchPDict(pdict, subject, max.mismatch=1)
countIndex(m) # now pattern 4 has 1 match too
m[[4]]
```
```
matchProbePair

Find "theoretical amplicons" mapped to a probe pair

Description

In the context of a computer-simulated PCR experiment, one wants to find the amplicons mapped to a given primer pair. The \texttt{matchProbePair} function can be used for this: given a forward and a reverse probe (i.e. the chromosome-specific sequences of the forward and reverse primers used for the experiment) and a target sequence (generally a chromosome sequence), the \texttt{matchProbePair} function will return all the "theoretical amplicons" mapped to this probe pair.

Usage

\begin{verbatim}
matchProbePair(Fprobe, Rprobe, subject, algorithm="auto", logfile=NULL, verbose=FALSE)
\end{verbatim}

Arguments

- **Fprobe**: The forward probe.
- **Rprobe**: The reverse probe.
- **subject**: A \texttt{DNAString} object (or an \texttt{XStringViews} object with a \texttt{DNAString} subject) containing the target sequence.
- **algorithm**: One of the following: "auto", "naive-exact", "naive-inexact", "boyer-moore" or "shift-or". See \texttt{matchPattern} for more information.
- **logfile**: A file used for logging.
- **verbose**: \texttt{TRUE} or \texttt{FALSE}.

Details

The \texttt{matchProbePair} function does the following: (1) find all the "plus hits" i.e. the Fprobe and Rprobe matches on the "plus" strand, (2) find all the "minus hits" i.e. the Fprobe and Rprobe matches on the "minus" strand and (3) from the set of all (plus_hit, minus_hit) pairs, extract and return the subset of "reduced matches" i.e. the (plus_hit, minus_hit) pairs such that (a) plus_hit <= minus_hit and (b) there are no hits (plus or minus) between plus_hit and minus_hit. This set of "reduced matches" is the set of "theoretical amplicons". 

```r
## WARNING: Support for 'fixed=FALSE' is currently broken (FIXME)
## Not run:
m <- matchPDict(pdict, subject, max.mismatch=1, fixed=FALSE)
countIndex(m)  # now pattern 3 has 1 match too
m[[3]]  # note that this match is "out of limit"
Views(subject, m[[3]])

## End(Not run)

m <- matchPDict(pdict, subject, max.mismatch=2)
countIndex(m)  # pattern 4 gets 1 additional match
m[[4]]

## Unlist all matches:
unlist(m)
```
matchprobes

Value

An `XStringViews` object containing the set of "theoretical amplicons".

Author(s)

H. Pages

See Also

`matchPattern`, `matchLRPatterns`, `findPalindromes`, `reverseComplement`, `XStringViews`

Examples

```r
library(BSgenome.Dmelanogaster.UCSC.dm3)
subject <- Dmelanogaster$chr3R

## With 20-nucleotide forward and reverse probes:
Fprobe <- "AGCTCCGAGTTCCCTGCAATA"
Rprobe <- "CGTTGTTCACAAATATGCGG"
matchProbePair(Fprobe, Rprobe, subject) # 1 "theoretical amplicon"

## With shorter forward and reverse probes, the risk of having multiple 
## "theoretical amplicons" increases:
Fprobe <- "AGCTCCGAGTTCC"
Rprobe <- "CGTTGTTCACAA"
matchProbePair(Fprobe, Rprobe, subject) # 2 "theoretical amplicons"
Fprobe <- "AGCTCCGAGTT"
Rprobe <- "CGTTGTTCACA"
matchProbePair(Fprobe, Rprobe, subject) # 9 "theoretical amplicons"
```

Description

The query sequence, a character string (probably representing a transcript of interest), is scanned for the presence of exact matches to the sequences in the character vector `records`. The indices of the set of matches are returned.

The function is inefficient: it works on R’s character vectors, and the actual matching algorithm is of time complexity \( \text{length}(\text{query}) \times \text{length}(\text{records}) \)!

See `matchPattern`, `vmatchPattern`, and `matchPDict` for more efficient sequence matching functions.

Usage

`matchprobes(query, records, probepos=FALSE)`
Arguments

query  A character vector. For example, each element may represent a gene (transcript) of interest. See Details.

records  A character vector. For example, each element may represent the probes on a DNA array.

probepos  A logical value. If TRUE, return also the start positions of the matches in the query sequence.

Details

toupper is applied to the arguments query and records before matching. The intention of this is to make the matching case-insensitive. The function is embarrassingly naive. The matching is done using the C library function strstr.

Value

A list. Its first element is a list of the same length as the input vector. Each element of the list is a numeric vector containing the indices of the probes that have a perfect match in the query sequence.

If probepos is TRUE, the returned list has a second element: it is of the same shape as described above, and gives the respective positions of the matches.

Author(s)

R. Gentleman, Laurent Gautier, Wolfgang Huber

See Also

matchPattern, vmatchPattern, matchPDict

Examples

```r
if(require("hgu95av2probe")){
  data("hgu95av2probe")
  seq <- hgu95av2probe$sequence[1:20]
  target <- paste(seq, collapse="")
  matchprobes(target, seq, probepos=TRUE)
}
```

matchPWM  A simple PWM matching function and related utilities

Description

A function implementing a simple algorithm for matching a set of patterns represented by a Position Weight Matrix (PWM) to a DNA sequence. PWM for amino acid sequences are not supported.
matchPWM

Usage

matchPWM(pwm, subject, min.score="80%")
countPWM(pwm, subject, min.score="80%")
PWMscoreStartingAt(pwm, subject, starting.at=1)

## Utility functions for basic manipulation of the Position Weight Matrix
maxWeights(pwm)
maxScore(pwm)
## S4 method for signature 'matrix':
reverseComplement(x, ...)

Arguments

pwm, x
A Position Weight Matrix (numeric matrix with row names A, C, G and T).
subject
A DNAString object containing the subject sequence.
min.score
The minimum score for counting a match. Can be given as a character string containing a percentage (e.g. "85%") of the highest possible score or as a single number.
starting.at
An integer vector specifying the starting positions of the Position Weight Matrix relatively to the subject.
...
Additional arguments are currently ignored by the reverseComplement method for matrix objects.

Value

An XStringViews object for matchPWM.
A single integer for countPWM.
A numeric vector containing the Position Weight Matrix-based scores for PWMscoreStartingAt.
A vector containing the max weight for each position in pwm for maxWeights.
The highest possible score for a given Position Weight Matrix for maxScore.
A PWM obtained by reverting the column order in PWM x and by reassigning each row to its complementary nucleotide for reverseComplement.

See Also

matchPattern, reverseComplement, DNAString-class, XStringViews-class

Examples

pwm <- rbind(A=c( 1, 0, 19, 20, 18, 1, 20, 7),
            C=c( 1, 0, 1, 0, 1, 18, 0, 2),
            G=c(17, 0, 0, 0, 1, 0, 0, 3),
            T=c( 1, 20, 0, 0, 0, 1, 0, 8))
maxWeights(pwm)
maxScore(pwm)
reverseComplement(pwm)

subject <- DNAString("AGTAAACA")
PWMscoreStartingAt(pwm, subject, starting.at=c(2:1, NA))

library(BSgenome.Dmelanogaster.UCSC.dm3)
match-utils

```r
chr3R <- unmasked(Dmelanogaster$chr3R)
chr3R
## Match the plus strand
matchPWM(pwm, chr3R)
countPWM(pwm, chr3R)
## Match the minus strand
matchPWM(reverseComplement(pwm), chr3R)
```

---

**match-utils**

*Utility functions related to pattern matching*

**Description**

In this man page we define precisely and illustrate what a "match" of a pattern P in a subject S is in the context of the Biostrings package. This definition of a "match" is central to most pattern matching functions available in this package: unless specified otherwise, most of them will adhere to the definition provided here.

`hasLetterAt` checks whether a sequence or set of sequences has the specified letters at the specified positions.

`neditStartingAt`, `neditEndingAt`, `isMatchingStartingAt` and `isMatchingEndingAt` are low-level matching functions that only check for matches at the specified positions.

Other utility functions related to pattern matching are described here: the `mismatch` function for getting the positions of the mismatching letters of a given pattern relatively to its matches in a given subject, the `nmatch` and `nmismatch` functions for getting the number of matching and mismatching letters produced by the `mismatch` function, and the `coverage` function that can be used to get the "coverage" of a subject by a given pattern or set of patterns.

**Usage**

```r
hasLetterAt(x, letter, at, fixed=TRUE)
neditStartingAt(pattern, subject, starting.at=1, with.indels=FALSE, fixed=TRUE)
neditEndingAt(pattern, subject, ending.at=1, with.indels=FALSE, fixed=TRUE)
neditAt(pattern, subject, at=1, with.indels=FALSE, fixed=TRUE)

isMatchingStartingAt(pattern, subject, starting.at=1,
  max.mismatch=0, with.indels=FALSE, fixed=TRUE)
isMatchingEndingAt(pattern, subject, ending.at=1,
  max.mismatch=0, with.indels=FALSE, fixed=TRUE)
isMatchingAt(pattern, subject, at=1,
  max.mismatch=0, with.indels=FALSE, fixed=TRUE)

mismatch(pattern, x, fixed=TRUE)
nmatch(pattern, x, fixed=TRUE)
nmismatch(pattern, x, fixed=TRUE)
## S4 method for signature 'MIndex':
coverage(x, start=NA, end=NA, shift=0L, width=NULL, weight=1L)
## S4 method for signature 'MaskedXString':
coverage(x, start=NA, end=NA, shift=0L, width=NULL, weight=1L)
```
Arguments

- **x**: A character vector, or an `XString` or `XStringSet` object for `hasLetterAt`. An `XStringViews` object for `mismatch` (typically, one returned by `matchPattern(pattern, subject)`).
- **letter**: An `MIndex` object for `coverage`, or any object for which a `coverage` method is defined. See `?coverage`.
- **at**: A character string or an `XString` object containing the letters to check.
- **at, starting.at, ending.at**: An integer vector specifying the starting (for `starting.at`) or ending (for `ending.at`) positions of the pattern relatively to the subject. For the `hasLetterAt` function, `letter` and `at` must have the same length.
- **pattern**: The pattern string.
- **subject**: An `XString`, `XStringSet` object, or character vector containing the subject sequence(s).
- **max.mismatch**: See details below.
- **with.indels**: See details below.
- **fixed**: Only with a `DNAString` or `RNAString`-based subject can a `fixed` value other than the default (TRUE) be used. With `fixed=FALSE`, ambiguities (i.e. letters from the IUPAC Extended Genetic Alphabet (see `IUPAC_CODE_MAP`) that are not from the base alphabet) in the pattern _and_ in the subject are interpreted as wildcards i.e. they match any letter that they stand for. `fixed` can also be a character vector, a subset of c("pattern", "subject"). `fixed=c("pattern", "subject")` is equivalent to `fixed=TRUE` (the default). An empty vector is equivalent to `fixed=FALSE`. With `fixed="subject"`, ambiguities in the pattern only are interpreted as wildcards. With `fixed="pattern"`, ambiguities in the subject only are interpreted as wildcards.
- **start, end, shift, width**: See `?coverage`.
- **weight**: An integer vector specifying how much each element in `x` counts.

Details

A "match" of pattern P in subject S is a substring S’ of S that is considered similar enough to P according to some distance (or metric) specified by the user. 2 distances are supported by most pattern matching functions in the Biostrings package. The first (and simplest) one is the "number of mismatching letters". It is defined only when the 2 strings to compare have the same length, so when this distance is used, only matches that have the same number of letters as P are considered. The second one is the "edit distance" (aka Levenshtein distance): it’s the minimum number of operations needed to transform P into S’, where an operation is an insertion, deletion, or substitution of a single letter. When this metric is used, matches can have a different number of letters than P.

The `neditStartingAt` (and `neditEndingAt`) function implements these 2 distances. If `with.indels` is `FALSE` (the default), then the first distance is used i.e. `neditStartingAt` returns the "number of mismatching letters" between the pattern P and the substring S’ of S starting at the positions specified in `starting.at` (note that `neditStartingAt` and `neditEndingAt` are vectorized so long vectors of integers can be passed thru the `starting.at` or `ending.at` arguments). If `with.indels` is `TRUE`, then the "edit distance" distance is used: for each position specified in `starting.at`, P is compared to all the substrings S’ of S starting at this position and the smallest distance is returned. Note that this distance is guaranteed to be reached for a substrings
match-utils

of length < 2*length(P) so, of course, in practice, P only needs to be compared to a small number of substrings for every starting position.

Value

`hasLetterAt`: A logical matrix with one row per element in x and one column per letter/position to check. When a specified position is invalid with respect to an element in x then the corresponding matrix element is set to NA.

`neditStartingAt` and `neditEndingAt`: If `subject` is an `XString` object, then return an integer vector of the same length as `starting.at` (or `ending.at`). If `subject` is an `XStringSet` object, then return the integer matrix with `length(starting.at)` (or `length(ending.at)`) rows and `length(subject)` columns defined by (in the case of `neditStartingAt`):

```r
sicr
  sapply(unname(subject),
    function(x) neditStartingAt(pattern, x, ...))
```

`isMatchingStartingAt` and `isMatchingEndingAt`: If `subject` is an `XString` object, then return the logical vector defined by `neditStartingAt(...) <= max.mismatch` or `neditEndingAt(...) <= max.mismatch`, respectively. If `subject` is an `XStringSet` object, then return the logical matrix with `length(starting.at)` (or `length(ending.at)`) rows and `length(subject)` columns defined by (in the case of `isMatchingStartingAt`):

```r
sicr
  sapply(unname(subject),
    function(x) isMatchingStartingAt(pattern, x, ...))
```

`neditAt` and `isMatchingAt` are convenience wrappers for `neditStartingAt` and `isMatchingStartingAt`, respectively.

`mismatch`: a list of integer vectors.

`nmismatch`: an integer vector containing the length of the vectors produced by `mismatch`.

`coverage`: an Rle object indicating the coverage of x. See `?coverage` for the details. If x is an `MIndex` object, the coverage of a given position in the underlying sequence (typically the subject used during the search that returned x) is the number of matches (or hits) it belongs to.

See Also

`nucleotideFrequencyAt`, `matchPattern`, `matchPDict`, `matchLRPatterns`, `trimLRPatterns`, `IUPAC_CODE_MAP`, `XString-class`, `XStringViews-class`, `MIndex-class`, `coverage`, `IRanges-class`, `MaskCollection-class`, `MaskedXString-class`, `align-utils`

Examples

```r
sicr

## hasLetterAt()

x <- DNAStringSet(c("AAACGT", "AACGT", "ACGT", "TAGGA"))
hasLetterAt(x, "AAAAAA", 1:6)

## hasLetterAt() can be used to answer questions like: "which elements in 'x' have an A at position 2 and a G at position 4?"

q1 <- hasLetterAt(x, "AG", c(2, 4))
which(rowSums(q1) == 2)

## or "how many probes in the drosophila2 chip have T, G, T, A at...
```
library(drosophila2probe)

probes <- DNAStringSet(drosophila2probe$sequence)

q2 <- hasLetterAt(probes, "TGTA", c(2, 4, 13, 20))
sum(rowSums(q2) == 4)

# or "what's the probability to have an A at position 25 if there is
# one at position 13?"

q3 <- hasLetterAt(probes, "AACGT", c(13, 25, 25, 25, 25))
sum(q3[, 1] & q3[, 2]) / sum(q3[, 1])

# Probabilities to have other bases at position 25 if there is an A
# at position 13:
sum(q3[, 1] & q3[, 3]) / sum(q3[, 1]) # C
sum(q3[, 1] & q3[, 4]) / sum(q3[, 1]) # G
sum(q3[, 1] & q3[, 5]) / sum(q3[, 1]) # T

# See ?nucleotideFrequencyAt for another way to get those results.

subject <- DNAString("GTATA")

# Pattern "AT" matches subject "GTATA" at position 3 (exact match)
neditAt("AT", subject, at=3)
isMatchingAt("AT", subject, at=3)

# ... but not at position 1
neditAt("AT", subject)
isMatchingAt("AT", subject)

# ... unless we allow 1 mismatching letter (inexact match)
isMatchingAt("AT", subject, max.mismatch=1)

# Here we look at 6 different starting positions and find 3 matches if
# we allow 1 mismatching letter
isMatchingAt("AT", subject, at=0:5, max.mismatch=1)

# No match
neditAt("NT", subject, at=1:4)
isMatchingAt("NT", subject, at=1:4)

# 2 matches if N is interpreted as an ambiguity (fixed=FALSE)
neditAt("NT", subject, at=1:4, fixed=FALSE)
isMatchingAt("NT", subject, at=1:4, fixed=FALSE)

# max.mismatch != 0 and fixed=FALSE can be used together
neditAt("NCA", subject, at=0:5, fixed=FALSE)
isMatchingAt("NCA", subject, at=0:5, max.mismatch=1, fixed=FALSE)

some_starts <- c(10:-10, NA, 6)
subject <- DNAString("ACGTGCA")
is_matching <- isMatchingAt("CAT", subject, at=some_starts, max.mismatch=1)
some_starts[is_matching]

# mismatch() / nmismatch()
m <- matchPattern("NCA", subject, max.mismatch=1, fixed=FALSE)
mismatch("NCA", m)
nmismatch("NCA", m)

## See ?matchPDict for examples of using coverage() on an MIndex object...

---

### Description

The MIndex class is the basic container for storing the matches of a set of patterns in a subject sequence.

### Details

An MIndex object contains the matches (start/end locations) of a set of patterns found in an XString object called "the subject string" or "the subject sequence" or simply "the subject". The `matchPDict` function returns an MIndex object.

### Accessor methods

In the code snippets below, `x` is an MIndex object.

- `length(x)`: The number of patterns that matches are stored for.
- `names(x)`: The names of the patterns that matches are stored for.
- `startIndex(x)`: A list containing the starting positions of the matches for each pattern.
- `endIndex(x)`: A list containing the ending positions of the matches for each pattern.
- `countIndex(x)`: An integer vector containing the number of matches for each pattern.

### Subsetting methods

In the code snippets below, `x` is an MIndex object.

- `x[[i]]`: Extract the matches for the i-th pattern as an IRanges object.

### Other utility methods and functions

In the code snippets below, `x` and `mindex` are MIndex objects and `subject` is the XString object containing the sequence in which the matches were found.

- `unlist(x, recursive=TRUE, use.names=TRUE)`: Return all the matches in a single IRanges object. `recursive` and `use.names` are ignored.
- `extractAllMatches(subject, mindex)`: Return all the matches in a single XStringViews object.
**needwunsQS**

**Author(s)**

H. Pages

**See Also**

`matchPDict`, `PDict-class`, `IRanges-class`, `XStringViews-class`

**Examples**

```
```

---

**needwunsQS** *(Deprecated) Needleman-Wunsch Global Alignment*

**Description**

Simple gap implementation of Needleman-Wunsch global alignment algorithm.

**Usage**

```
needwunsQS(s1, s2, substmat, gappen = 8)
```

**Arguments**

- `s1, s2` an R character vector of length 1 or an `XString` object.
- `substmat` matrix of alignment score values.
- `gappen` penalty for introducing a gap in the alignment.

**Details**

Follows specification of Durbin, Eddy, Krogh, Mitchison (1998). This function has been deprecated and is being replaced by `pairwiseAlignment`.

**Value**

An instance of class "PairwiseAlignedXStringSet".

**Author(s)**

Vince Carey (stvjc@channing.harvard.edu) (original author) and H. Pages (current maintainer).

**References**


**See Also**

`pairwiseAlignment`, `PairwiseAlignedXStringSet-class`, `substitution.matrices`
Example

```r
## Not run:
## This function has been deprecated
## Use 'pairwiseAlignment' instead.

## nucleotide alignment
mat <- matrix(-5L, nrow = 4, ncol = 4)
for (i in seq_len(4)) mat[i, i] <- 0L
rownames(mat) <- colnames(mat) <- DNA_ALPHABET[1:4]
s1 <- DNAString(paste(sample(DNA_ALPHABET[1:4], 1000, replace=TRUE), collapse=""))
s2 <- DNAString(paste(sample(DNA_ALPHABET[1:4], 1000, replace=TRUE), collapse=""))
nw0 <- needwunsQS(s1, s2, mat, gappen = 0)
nw1 <- needwunsQS(s1, s2, mat, gappen = 1)
nw5 <- needwunsQS(s1, s2, mat, gappen = 5)

## amino acid alignment
needwunsQS("PAWHEAE", "HEAGAWGHEE", substmat = "BLOSUM50")
## End(Not run)
```

---

**nucleotideFrequency**

*Calculate the frequency of oligonucleotides in a DNA or RNA sequence, plus some related functions*

**Description**

Given a DNA or RNA sequence (or a set of DNA or RNA sequences), the `nucleotideFrequency` function computes the frequency of all possible oligonucleotides of a given length (called the "width" in this particular context).

The `dinucleotideFrequency` and `trinucleotideFrequency` functions are convenient wrappers for calling `nucleotideFrequency` with `width=2` and `width=3`, respectively.

The `nucleotideFrequencyAt` function computes the frequency of the short sequences formed by extracting the nucleotides found at some fixed positions from each sequence of a set of DNA or RNA sequences.

In this man page we call "DNA input" (or "RNA input") an `XString`, `XStringSet`, `XStringViews` or `MaskedXString` object of base type DNA (or RNA).

**Usage**

```r
nucleotideFrequency(x, width, freq=FALSE, as.array=FALSE, 
                  fast.moving.side="right", with.labels=TRUE, ...)
```

```r
## S4 method for signature 'XStringSet':
nucleotideFrequency(x, 
                    width, freq=FALSE, as.array=FALSE, 
                    fast.moving.side="right", with.labels=TRUE, ...)
```

```r
dinucleotideFrequency(x, freq=FALSE, as.matrix=FALSE, 
                      fast.moving.side="right", with.labels=TRUE, ...)
```

```r
trinucleotideFrequency(x, freq=FALSE, as.array=FALSE, 
                       fast.moving.side="right", with.labels=TRUE, ...)
```
Arguments

\(x\)  
Any DNA or RNA input for the *Frequency and oligonucleotideTransitions functions.

\(\text{width}\)  
The number of nucleotides per oligonucleotide for \text{oligonucleotideFrequency}.

\(\text{at}\)  
An integer vector containing the positions to look at in each element of \(x\).

\(\text{freq}\)  
If \text{TRUE} then relative frequencies are reported, otherwise counts (the default).

\(\text{as.array, as.matrix}\)  
Controls the "shape" of the returned object. If \text{true} (the default for \text{nucleotideFrequencyAt}) then it's a numeric matrix (or array), otherwise it's just a "flat" numeric vector i.e. a vector with no dim attribute (the default for the *Frequency functions).

\(\text{fast.moving.side}\)  
Which side of the strings should move fastest? Note that, when \text{as.array} is \text{true}, then the supplied value is ignored and the effective value is "left".

\(\text{with.labels}\)  
If \text{true} then the returned object is named.

\(\ldots\)  
Further arguments to be passed to or from other methods.

\(\text{simplify.as}\)  
Together with the \text{as.array} and \text{as.matrix} arguments, controls the "shape" of the returned object when the input \(x\) is an \text{XStringSet} or \text{XStringViews} object. Supported \text{simplify.as} values are "matrix" (the default), "list," and "collapsed". If \text{simplify.as} is "matrix", the returned object is a matrix with length(\(x\)) rows where the \(i\)-th row contains the frequencies for \(\text{x}[i]\). If \text{simplify.as} is "list," the returned object is a list of the same length as length(\(x\)) where the \(i\)-th element contains the frequencies for \(\text{x}[i]\). If \text{simplify.as} is "collapsed", then the the frequencies are computed for the entire object \(x\) as a whole (i.e. frequencies cumulated across all sequences in \(x\)).

\(\text{left, right}\)  
The number of nucleotides per oligonucleotide for the rows and columns respectively in the transition matrix created by \text{oligonucleotideTransitions}.

\(\text{alphabet}\)  
The alphabet to use to make the strings.

Value

If \(x\) is an \text{XString} or \text{MaskedXString} object, the *Frequency functions return a numeric vector of length \(4^{\text{width}}\). If \text{as.array} (or \text{as.matrix}) is \text{true}, then this vector is formatted as an array (or matrix). If \(x\) is an \text{XStringSet} or \text{XStringViews} object, the returned object has the shape specified by the \text{simplify.as} argument.

Author(s)

H. Pages and P. Aboyoun
nucleotideFrequency

See Also
alphabetFrequency, alphabet.hasLetterAt, XString-class, XStringSet-class, XStringViews-class, MaskedXString-class, GENETIC_CODE, AMINO_ACID_CODE, reverse, XString-method, rev

Examples

## A. BASIC +Frequency() EXAMPLES
## ---------------------------------------------------------------------
data(yeastSEQCHR1)
yeast1 <- DNAString(yeastSEQCHR1)
dinucleotideFrequency(yeast1)
trinucleotideFrequency(yeast1)
oligonucleotideFrequency(yeast1, 4)

## Get the less and most represented 6-mers:
f6 <- oligonucleotideFrequency(yeast1, 6)
f6[f6 == min(f6)]
f6[f6 == max(f6)]

## Get the result as an array:
tri <- trinucleotideFrequency(yeast1, as.array=TRUE)
tri["A", "A", "C"] # == trinucleotideFrequency(yeast1)["AAC"]
tri["T", , ] # frequencies of trinucleotides starting with a "T"

## With input made of multiple sequences:
library(drosophila2probe)
probes <- DNAStringSet(drosophila2probe$sequence)
dfmat <- dinucleotideFrequency(probes) # a big matrix
dinucleotideFrequency(probes, simplify.as="collapsed")
dinucleotideFrequency(probes, simplify.as="collapsed", as.matrix=TRUE)

## B. nucleotideFrequencyAt()
## ---------------------------------------------------------------------
nucleotideFrequencyAt(probes, 13)
nucleotideFrequencyAt(probes, c(13, 20))
nucleotideFrequencyAt(probes, c(13, 20), as.array=FALSE)

nucleotideFrequencyAt() can be used to answer questions like: "how many probes in the drosophila2 chip have T, G, T, A at position 2, 4, 13 and 20, respectively?"
nucleotideFrequencyAt(probes, c(2, 4, 13, 20))['T', 'G', 'T', 'A']

or "what's the probability to have an A at position 25 if there is one at position 13?"
nf <- nucleotideFrequencyAt(probes, c(13, 25))
sum(nf["A", "A"]) / sum(nf["A",])

"Probabilities to have other bases at position 25 if there is an A at position 13:"
sum(nf["A", "C"]) / sum(nf["A",]) # C
sum(nf["A", "G"]) / sum(nf["A",]) # G
sum(nf["A", "T"]) / sum(nf["A",]) # T

See ?hasLetterAt for another way to get those results.
## C. oligonucleotideTransitions()

Get nucleotide transition matrices for yeast1

oligonucleotideTransitions(yeast1)

```r
oligonucleotideTransitions(yeast1, 2, freq=TRUE)
```

## D. ADVANCED Frequency() EXAMPLES

Note that when dropping the dimensions of the 'tri' array, elements in the resulting vector are ordered as if they were obtained with

```r
fast.moving.side="left"
```

```r
triL <- trinucleotideFrequency(yeast1, fast.moving.side="left")
all(as.vector(tri) == triL) # TRUE
```

Convert the trinucleotide frequency into the amino acid frequency based on translation:

```r
tri1 <- trinucleotideFrequency(yeast1)
names(tri1) <- GENETIC_CODE[names(tri1)]
sapply(split(tri1, names(tri1)), sum) # 12512 occurrences of the stop codon
```

When the returned vector is very long (e.g. width >= 10), using 'with.labels=FALSE' can improve performance significantly. Here for example, the observed speed up is between 25x and 500x:

```r
f12 <- oligonucleotideFrequency(yeast1, 12, with.labels=FALSE) # very fast!
```

Some related functions:

```r
dict1 <- mkAllStrings(LETTERS[1:3], 4)
dict2 <- mkAllStrings(LETTERS[1:3], 4, fast.moving.side="left")
identical(reverse(dict1), dict2) # TRUE
```

---

**PairwiseAlignedXStringSet-class**

PairwiseAlignedXStringSet, PairwiseAlignedFixedSubject, and PairwiseAlignedFixedSubjectSummary objects

### Description

The PairwiseAlignedXStringSet class is a container for storing an elementwise pairwise alignment. The PairwiseAlignedFixedSubject class is a container for storing a pairwise alignment with a single subject. The PairwiseAlignedFixedSubjectSummary class is a container for storing the summary of an alignment.

### Usage

```r
## Constructors:
## When subject is missing, pattern must be of length 2
## S4 method for signature 'XString, XString':
PairwiseAlignedXStringSet(pattern, subject,
    type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = -
## S4 method for signature 'XStringSet, missing':
PairwiseAlignedXStringSet(pattern, subject,
```

---

**PairwiseAlignedXStringSet-class**

PairwiseAlignedXStringSet, PairwiseAlignedFixedSubject, and PairwiseAlignedFixedSubjectSummary objects

### Description

The PairwiseAlignedXStringSet class is a container for storing an elementwise pairwise alignment. The PairwiseAlignedFixedSubject class is a container for storing a pairwise alignment with a single subject. The PairwiseAlignedFixedSubjectSummary class is a container for storing the summary of an alignment.

### Usage

```r
## Constructors:
## When subject is missing, pattern must be of length 2
## S4 method for signature 'XString, XString':
PairwiseAlignedXStringSet(pattern, subject,
    type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = -
## S4 method for signature 'XStringSet, missing':
PairwiseAlignedXStringSet(pattern, subject,
```
PairwiseAlignedXStringSet-class

```r
PairwiseAlignedXStringSet(pattern, subject,
                      type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = -1,
                      baseClass = "BString")

## S4 method for signature 'character, character':
PairwiseAlignedXStringSet(pattern, subject,
                      type = "global", substitutionMatrix = NULL, gapOpening = 0, gapExtension = -1,
                      baseClass = "BString")

## S4 method for signature 'character, missing':
PairwiseAlignedXStringSet(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 with end gap penalties on `pattern` and without end gap penalties on `subject`. "local-global" = align whole strings without end gap penalties on `pattern` and with end gap penalties on `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 `str1` is obtained by inserting 0 or any number of gaps in a subsequence of `str1`. For example L-A–ND and A–N-D are "filled-with-gaps subsequences" of LAND. An alignment between two strings `str1` and `str2` results in two strings (align1 and align2) that have the same length and are "filled-with-gaps subsequences" of `str1` and `str2`.

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 `str1` 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 \( \geq i \) 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 PairwiseAlignedXStringSet object, except otherwise noted.

- \( \text{pattern}(x) \): The AlignedXStringSet object for the pattern.
- \( \text{subject}(x) \): The AlignedXStringSet object for the subject.
- \( \text{summary(object, \ldots)} \): Generates a summary for the PairwiseAlignedXStringSet.

**General information methods**

In the code snippets below, \( x \) is a PairwiseAlignedXStringSet object, except otherwise noted.

- \( \text{alphabet}(x) \): Equivalent to \( \text{alphabet}(\text{unaligned(subject}(x))) \).
- \( \text{length}(x) \): The length of the aligned(pattern\( (x) \)) and aligned(subject\( (x) \)). There is a method for PairwiseAlignedFixedSubjectSummary as well.
- \( \text{type}(x) \): The type of the alignment ("global", "local", "overlap", "global-local", or "local-global"). There is a method for PairwiseAlignedFixedSubjectSummary as well.

**Aligned sequence methods**

In the code snippets below, \( x \) is a PairwiseAlignedFixedSubject object, except otherwise noted.

- \( \text{aligned}(x, \text{degap} = \text{FALSE}, \text{gapCode}="-", \text{endgapCode}="-") \): If \( \text{degap} = \text{FALSE} \), "align" the alignments by returning an XStringSet object containing the aligned patterns without insertions. If \( \text{degap} = \text{TRUE}, \text{returns aligned(pattern}(x), \text{degap=TRUE}) \). The gapCode and endgapCode arguments denote the code in the appropriate alphabet to use for the internal and end gaps.
- \( \text{as.character}(x) \): Converts \( \text{aligned}(x) \) to a character vector.
- \( \text{as.matrix}(x) \): Returns an "exploded" character matrix representation of \( \text{aligned}(x) \).
- \( \text{toString}(x) \): Equivalent to \( \text{toString(as.character}(x)) \).

**Subject position methods**

In the code snippets below, \( x \) is a PairwiseAlignedFixedSubject object, except otherwise noted.

- \( \text{consensusMatrix}(x, \text{baseOnly}=\text{FALSE}, \text{freq}=\text{FALSE}, \text{gapCode}="-", \text{endgapCode}="-")) \) See 'consensusMatrix' for more information.
- \( \text{consensusString}(x) \) See 'consensusString' for more information.
- \( \text{coverage}(x, \text{start}=\text{NA}, \text{end}=\text{NA}, \text{shift}=0L, \text{width}=\text{NULL}, \text{weight}=1L) \) See 'coverage,PairwiseAlignedFixedSubject-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 PairwiseAlignedXStringSet object, except otherwise noted.

nchar(x): The nchar of the aligned(pattern(x)) and aligned(subject(x)). There is a method for PairwiseAlignedFixedSubjectSummary as well.
nindel(x): An InDel object containing the number of insertions and deletions.
score(x): The score of the alignment. There is a method for PairwiseAlignedFixedSubjectSummary as well.

Subsetting methods

x[i]: Returns a new PairwiseAlignedXStringSet object made of the selected elements.
rep(x, times): Returns a new PairwiseAlignedXStringSet object made of the repeated elements.

Author(s)

P. Aboyoun

See Also

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

Examples

PairwiseAlignedXStringSet("-PA--W-HEAE", "HEAGAWGHE-E")
pattern <- AStringSet(c("HLDNLKGTF", "HVDDMPNAL"))
subject <- AString("SMDDEKSMK")
nw1 <- pairwiseAlignment(pattern, subject, substitutionMatrix = "BLOSUM50",
  gapOpening = -3, gapExtension = -1)
pattern(nw1)
subject(nw1)
aligned(nw1)
as.character(nw1)
as.matrix(nw1)
nchar(nw1)
score(nw1)
nw1
pairwiseAlignment

Optimal Pairwise Alignment

Description

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

Usage

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

## S4 method for signature 'XStringSet, XStringSet':

```r
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':

```r
pairwiseAlignment(pattern, subject,
    type = "global", substitutionMatrix = NULL, fuzzyMatrix = NULL,
    gapOpening = -10, gapExtension = -4, scoreOnly = FALSE)
```

Arguments

- `pattern`: a character vector of any length, an `XString`, or an `XStringSet` object.
- `subject`: a character vector of length 1 or an `XString` object.
- `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 with end gap penalties on `pattern` and without end gap penalties on `subject", "local-global" = align whole strings without end gap penalties on `pattern` and with end gap penalties on `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.
pairwiseAlignment

Details

Quality-based alignments are based on the paper the Bioinformatics article by Ketil Malde listed in the Reference section below. Let $\epsilon_i$ 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 $\epsilon_c$, the substitution score is given by when two bases match is given by

$$b \times \log_2(\gamma_{x,y} * (1 - \epsilon_c) * n + (1 - \gamma_{x,y}) * \epsilon_c * (n/(n-1)))$$

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 $\epsilon_1$ and $\epsilon_2$ respectively.

If scoreOnly == FALSE, the pairwise alignment with the maximum alignment score is returned. If more than one pairwise alignment has the maximum alignment score exists, the first alignment along the subject is returned. If there are multiple pairwise alignments with the maximum alignment score at the chosen subject location, then at each location along the alignment mismatches are given preference to insertions/deletions. For example, pattern: [1] ATTA; subject: [1] AT-A is chosen above pattern: [1] ATTA; subject: [1] A-TA if they both have the maximum alignment score.

Value

If scoreOnly == FALSE, an instance of class PairwiseAlignedXStringSet or PairwiseAlignedFixedSubject is returned. If scoreOnly == TRUE, a numeric vector containing the scores for the 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 and H. Pages

References


See Also

stringDist, PairwiseAlignedXStringSet-class, XStringQuality-class, substitution.matrices, matchPattern
Examples

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

PDict-class

PDict-class

**PDict objects**

Description

The PDict class is a container for storing a preprocessed dictionary of DNA patterns that can later be passed to the `matchPDict` function for fast matching against a reference sequence (the subject). `PDict` is the constructor function for creating new PDict objects.

Usage

```r
PDict(x, max.mismatch=NA, tb.start=NA, tb.end=NA, tb.width=NA,
      algorithm="ACTree2", skip.invalid.patterns=FALSE)
```
Arguments

x A character vector, a DNAStringSet object or an XStringViews object with a DNAString subject.

max.mismatch A single non-negative integer or NA. See the "Allowing a small number of mismatching letters" section below.

tb.start, tb.end, tb.width A single integer or NA. See the "Trusted Band" section below.

algorithm "ACtree2" (the default), "ACtree" or "Twobit".

skip.invalid.patterns This argument is not supported yet (and might in fact be replaced by the filter argument very soon).

Details

THIS IS STILL WORK IN PROGRESS!

If the original dictionary x is a character vector or an XStringViews object with a DNAString subject, then the PDict constructor will first try to turn it into a DNAStringSet object.

By default (i.e. if PDict is called with max.mismatch=NA, tb.start=NA, tb.end=NA and tb.width=NA) the following limitations apply: (1) the original dictionary can only contain base letters (i.e. only As, Cs, Gs and Ts), therefore IUPAC extended letters are not allowed; (2) all the patterns in the dictionary must have the same length ("constant width" dictionary); and (3) later matchPdict can only be used with max.mismatch=0.

A Trusted Band can be used in order to relax these limitations (see the "Trusted Band" section below).

If you are planning to use the resulting PDict object in order to do inexact matching where valid hits are allowed to have a small number of mismatching letters, then see the "Allowing a small number of mismatching letters" section below.

Three preprocessing algorithms are currently supported: algorithm="ACtree2" (the default), algorithm="ACtree" and algorithm="Twobit". With the "ACtree2" and "ACtree" algorithms, all the oligonucleotides in the Trusted Band are stored in a 4-ary Aho-Corasick tree. With the "Twobit" algorithm, the 2-bit-per-letter signatures of all the oligonucleotides in the Trusted Band are computed and the mapping from these signatures to the 1-based position of the corresponding oligonucleotide in the Trusted Band is stored in a way that allows very fast lookup. Only with PDict objects obtained with the "ACtree2" or "ACtree" algos can matchPdict then be called with fixed="pattern" (instead of fixed=TRUE, the default) so that IUPAC extended letters in the subject are treated as ambiguities. PDict objects obtained with the "Twobit" algo don’t allow this.

Trusted Band

What’s a Trusted Band?

A Trusted Band is a region defined in the original dictionary where the limitations described above will apply.

Why use a Trusted Band?

Because the limitations described above will apply to the Trusted Band only! For example the Trusted Band cannot contain IUPAC extended letters but the "head" and the "tail" can (see below for what those are). Also with a Trusted Band, if matchPdict is called with a non-null max.mismatch value then mismatching letters will be allowed in the head and the tail. Or, if
**PDict-class**

matchPdict is called with fixed="subject", then IUPAC extended letters in the head and the tail will be treated as ambiguities.

**How to specify a Trusted Band?**

Use the `tb.start`, `tb.end` and `tb.width` arguments of the `PDict` constructor in order to specify a Trusted Band. This will divide each pattern in the original dictionary into three parts: a left part, a middle part and a right part. The middle part is defined by its starting and ending nucleotide positions given relatively to each pattern thru the `tb.start`, `tb.end` and `tb.width` arguments. It must have the same length for all patterns (this common length is called the width of the Trusted Band). The left and right parts are defined implicitly: they are the parts that remain before (prefix) and after (suffix) the middle part, respectively. Therefore three `DNAStringSet` objects result from this division: the first one is made of all the left parts and forms the head of the PDict object, the second one is made of all the middle parts and forms the Trusted Band of the PDict object, and the third one is made of all the right parts and forms the tail of the PDict object.

In other words you can think of the process of specifying a Trusted Band as drawing 2 vertical lines on the original dictionary (note that these 2 lines are not necessarily straight lines but the horizontal space between them must be constant). When doing this, you are dividing the dictionary into three regions (from left to right): the head, the Trusted Band and the tail. Each of them is a `DNAStringSet` object with the same number of elements than the original dictionary and the original dictionary could easily be reconstructed from those three regions.

The width of the Trusted Band must be \( \geq 1 \) because Trusted Bands of width 0 are not supported.

Finally note that calling `PDict` with `tb.start=NA`, `tb.end=NA` and `tb.width=NA` (the default) is equivalent to calling it with `tb.start=1`, `tb.end=-1` and `tb.width=NA`, which results in a full-width Trusted Band i.e. a Trusted Band that covers the entire dictionary (no head and no tail).

**Allowing a small number of mismatching letters**

**TODO**

**Accessor methods**

In the code snippets below, \( x \) is a PDict object.

- `length(x)`: The number of patterns in \( x \).
- `width(x)`: A vector of non-negative integers containing the number of letters for each pattern in \( x \).
- `names(x)`: The names of the patterns in \( x \).
- `head(x)`: The head of \( x \) or NULL if \( x \) has no head.
- `tb(x)`: The Trusted Band defined on \( x \).
- `tb.width(x)`: The width of the Trusted Band defined on \( x \). Note that, unlike `width(tb(x))`, this is a single integer. And because the Trusted Band has a constant width, `tb.width(x)` is in fact equivalent to `unique(width(tb(x)))`, or to `width(tb(x))[1]`.
- `tail(x)`: The tail of \( x \) or NULL if \( x \) has no tail.

**Subsetting methods**

In the code snippets below, \( x \) is a PDict object.

- \( x[[i]] \): Extract the \( i \)-th pattern from \( x \) as a `DNAString` object.
Other methods

In the code snippet below, x is a PDict object.

\[
duplicated(x): [\text{TODO}]
\]
\[
\text{patternFrequency}(x): [\text{TODO}]
\]

Author(s)

H. Pages

References


See Also

matchPDict, DNA_ALPHABET, DNAStringSet-class, XStringViews-class

Examples

```r
# A. NO HEAD AND NO TAIL (THE DEFAULT)
library(drosophila2probe)
dict0 <- DNAStringSet(drosophila2probe$sequence)
dict0
length(dict0)  # The original dictionary.
u nique(nchar(dict0))  # Patterns are 25-mers.

pdict0 <- PDict(dict0)  # Store the original dictionary in
# a PDict object (preprocessing).
pdict0
class(pdict0)
length(pdict0)  # Same as length(dict0).
tb.width(pdict0)  # The width of the (implicit)
# Trusted Band.
sum(duplicated(pdict0))
table(patternFrequency(pdict0))  # 9 patterns are repeated 3 times.
pdict0[[1]]
pdict0[[5]]

# B. NO HEAD AND A TAIL

dict1 <- c("ACNG", "GT", "CGT", "AC")
pdict1 <- PDict(dict1, tb.end=2)
pdict1
class(pdict1)
length(pdict1)
width(pdict1)
head(pdict1)
tb(pdict1)
tb.width(pdict1)
width(tb(pdict1))
tail(pdict1)
```
phiX174Phage

pdict1[[3]]

| phiX174Phage | Versions of bacteriophage phiX174 complete genome and sample short reads |

**Description**
Six versions of the complete genome for bacteriophage φX174 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.

**Details**
The phiX174Phage object is a DNAStringSet containing the following six naturally occurring versions of the bacteriophage φX174 genome cited in Smith et al.:

**Genbank:** The version of the genome from GenBank (NC_001422.1, GI:9626372).
**RF70s:** A preparation of φX double-stranded replicative form (RF) of DNA by Clyde A. Hutchison III from the late 1970s.
**SS78:** A preparation of φX virion single-stranded DNA from 1978.
**Bull:** The sequence of wild-type φX used by Bull et al.
**G’97:** The φX replicative form (RF) of DNA from Bull et al.
**NEB’03:** A φX replicative form (RF) of DNA from New England BioLabs (NEB).

The srPhiX174 object is a DNAStringSet containing short reads from a Solexa machine.
The quPhiX174 object is a BStringSet containing Solexa quality scores associated with srPhiX174.
The wtPhiX174 object is an integer vector containing counts associated with srPhiX174.

**References**
http://www.genome.jp/dbget-bin/www_bget?refseq+NC_001422

**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 PairwiseAlignedXStringSet 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

See Also
pairwiseAlignment, PairwiseAlignedXStringSet-class, match-utils
Examples

```r
s1 <- DNAString("AGTATAGATAGATAGAT")
s2 <- DNAString("AGTAGATAGATGAGATAGATA")

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

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

pmatchPattern

Longest Common Prefix/Suffix/Substring searching functions

Description

Functions for searching the Longest Common Prefix/Suffix/Substring of two strings.

WARNING: These functions are experimental and might not work properly! Full documentation will come later.

Please send questions/comments to hpages@fhcrc.org

Thanks for your comprehension!

Usage

```r
lcprefix(s1, s2)
lcsuffix(s1, s2)
lcsubstr(s1, s2)

pmatchPattern(pattern, subject, maxlength.out=1L)
```

Arguments

- `s1` 1st string, a character string or an XString object.
- `s2` 2nd string, a character string or an XString object.
- `pattern` The pattern string.
- `subject` An XString object containing the subject string.
- `maxlength.out` The maximum length of the output i.e. the maximum number of views in the returned object.

See Also

matchPattern, XStringViews-class, XString-class
QualityScaledXStringSet-class

**QualityScaledBStringSet, QualityScaledDNAStringSet, QualityScaledRNAStringSet and QualityScaledAAStringSet objects**

Description

The QualityScaledBStringSet class is a container for storing a BStringSet object with an XStringQuality object.

Similarly, the QualityScaledDNAStringSet (or QualityScaledRNAStringSet, or QualityScaledAAStringSet) class is a container for storing a DNAStringSet (or RNAStringSet, or AAStringSet) objects with an XStringQuality object.

Usage

```r
## Constructors:
QualityScaledBStringSet(x, quality)
QualityScaledDNAStringSet(x, quality)
QualityScaledRNAStringSet(x, quality)
QualityScaledAAStringSet(x, quality)
```

Arguments

- `x` Either a character vector, or an XString, XStringSet or XStringViews object.
- `quality` An XStringQuality object.

Details

The QualityScaledBStringSet, QualityScaledDNAStringSet, QualityScaledRNAStringSet and QualityScaledAAStringSet functions are constructors that can be used to "naturally" turn `x` into a QualityScaledXStringSet object of the desired base type.

Accessor methods

The QualityScaledXStringSet class derives from the XStringSet class hence all the accessor methods defined for an XStringSet object can also be used on a QualityScaledXStringSet object. Common methods include (in the code snippets below, `x` is an QualityScaledXStringSet object):

- `length(x)`: The number of sequences in `x`.
- `width(x)`: A vector of non-negative integers containing the number of letters for each element in `x`.
- `nchar(x)`: The same as `width(x)`.
- `names(x)`: NULL or a character vector of the same length as `x` containing a short user-provided description or comment for each element in `x`.
- `quality(x)`: The quality of the strings.

Subsetting and appending

In the code snippets below, `x` and `values` are XStringSet objects, and `i` should be an index specifying the elements to extract.

- `x[i]`: Return a new QualityScaledXStringSet object made of the selected elements.
readFASTA

Functions to read/write FASTA formatted files

Description

FASTA is a simple file format for biological sequence data. A file may contain one or more sequences, for each sequence there is a description line which begins with a >.

Usage

```r
fasta.info(file, use.descs=TRUE)
readFASTA(file, checkComments=TRUE, strip.descs=TRUE)
writeFASTA(x, file="", append=FALSE, width=80)
```

Arguments

- **file**: Either a character string naming a file or a connection. If "" (the default for `writeFASTA`), then the function writes to the standard output connection (the console) unless redirected by `sink`.
- **use.descs**: TRUE or FALSE. Whether or not the description lines should be used to name the elements of the returned integer vector.
- **checkComments**: Whether or not comments, lines beginning with a semi-colon should be found and removed.
- **strip.descs**: Whether or not the ">" marking the beginning of the description lines should be removed. Note that this argument is new in Biostrings >= 2.8. In previous versions `readFASTA` was keeping the ">".
- **x**: A list as one returned by `readFASTA`.
- **append**: TRUE or FALSE. If TRUE output will be appended to file; otherwise, it will overwrite the contents of file. See `?cat` for the details.
- **width**: The maximum number of letters per line of sequence.
Details

FASTA is a widely used format in biology. It is a relatively simple markup. I am not aware of a standard. It might be nice to check to see if the data that were parsed are sequences of some appropriate type, but without a standard that does not seem possible.

There are many other packages that provide similar, but different capabilities. The one in the package seqinr seems most similar but they separate the biological sequence into single character strings, which is too inefficient for large problems.

Value

An integer vector (for `fasta.info`) or a list (for `readFASTA`) with one element for each sequence in the file. For `readFASTA`, the elements are in two parts, one the description and the second a character string of the biological sequence.

Author(s)

R. Gentleman, H. Pages

See Also

`read.BStringSet, read.DNAStringSet, read.RNAStringSet, read.AAStringSet, write.XStringSet, read.table, scan, write.table`

Examples

```r
f1 <- system.file("extdata", "someORF.fa", package="Biostrings")
fasta.info(f1)
ff <- readFASTA(f1, strip.descs=TRUE)
desc <- sapply(ff, function(x) x$desc)
## Keep the "reverse complement" sequences only
ff2 <- ff[grep("reverse complement", desc, fixed=TRUE)]
writeFASTA(ff2, file.path(tempdir(), "someORF2.fa"))
```

replaceLetterAt

Replacing letters in a sequence (or set of sequences) at some specified locations

Description

`replaceLetterAt` first makes a copy of a sequence (or set of sequences) and then replaces some of the original letters by new letters at the specified locations.

`.inplaceReplaceLetterAt` is the IN PLACE version of `replaceLetterAt`: it will modify the original sequence in place i.e. without copying it first. Note that in place modification of a sequence is fundamentally dangerous because it alters all objects defined in your session that make reference to the modified sequence. NEVER use `.inplaceReplaceLetterAt`, unless you know what you are doing!

Usage

```r
replaceLetterAt(x, at, letter, if.not.extending="replace", verbose=FALSE)
## NEVER USE THIS FUNCTION!
.inplaceReplaceLetterAt(x, at, letter)
```
Arguments

x

A DNAString or rectangular DNAStringSet object.

at

The locations where the replacements must occur.

If x is a DNAString object, then at is typically an integer vector with no NAs but a logical vector or Rle object is valid too. Locations can be repeated and in this case the last replacement to occur at a given location prevails.

If x is a rectangular DNAStringSet object, then at must be a matrix of logicals with the same dimensions as x.

letter

The new letters.

If x is a DNAString object, then letter must be a DNAString object or a character vector (with no NAs) with a total number of letters (sum(nchar(letter))) equal to the number of locations specified in at.

If x is a rectangular DNAStringSet object, then letter must be a DNAStringSet object or a character vector of the same length as x. In addition, the number of letters in each element of letter must match the number of locations specified in the corresponding row of at (all(width(letter) == rowSums(at))).

if.not.extending

What to do if the new letter is not "extending" the old letter? The new letter "extends" the old letter if both are IUPAC letters and the new letter is as specific or less specific than the old one (e.g. M extends A, Y extends Y, but Y doesn’t extend S). Possible values are "replace" (the default) for replacing in all cases, "skip" for not replacing when the new letter does not extend the old letter, "merge" for merging the new IUPAC letter with the old one, and "error" for raising an error.

Note that the gap ("-")) and hard masking ("+") letters are not extending or extended by any other letter.

Also note that "merge" is the only value for the if.not.extending argument that guarantees the final result to be independent on the order the replacement is performed (although this is only relevant when at contains duplicated locations, otherwise the result is of course always independent on the order, whatever the value of if.not.extending is).

verbose

When TRUE, a warning will report the number of skipped or merged letters.

Details

.replaceLetterAt semantic is equivalent to calling replaceLetterAt with if.not.extending="replace" and verbose=FALSE.

Never use .replaceLetterAt! It is used by the injectSNPs function in the BSgenome package, as part of the "lazy sequence loading" mechanism, for altering the original sequences of a BSgenome object at "sequence-load time". This alteration consists in injecting the IUPAC ambiguity letters representing the SNPs into the just loaded sequence, which is the only time where in place modification of the external data of an XString object is safe.

Value

A DNAString or DNAStringSet object of the same shape (i.e. length and width) as the original object x for replaceLetterAt.
reverseComplement

Author(s)
H. Pages

See Also
IUPAC_CODE_MAP, chartr, injectHardMask, DNAString, DNAStringSet, injectSNPs, BSgenome

Examples
## Replace letters of a DNAString object:
replaceLetterAt(DNAString("AAMAA"), c(5, 1, 3, 1), "TYNC")
replaceLetterAt(DNAString("AAMAA"), c(5, 1, 3, 1), "TYNC", if.not.extending="merge")

## Replace letters of a DNAStringSet object (sorry for the totally
## artificial example with absolutely no biological meaning):
library(drosophila2probe)
probes <- DNAStringSet(drosophila2probe$sequence)
at <- matrix(c(TRUE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE),
nrow=length(probes), ncol=width(probes)[1], byrow=TRUE)
letter_subject <- DNAString(paste(rep.int("-", width(probes)[1]), collapse=""))
letter <- as(Views(letter_subject, start=1, end=rowSums(at)), "XStringSet")
replaceLetterAt(probes, at, letter)

reverseComplement  Sequence reversing and complementing

Description
Use these functions for reversing sequences and/or complementing DNA or RNA sequences.

Usage
## S4 method for signature 'character':
reverse(x, ...)
## S4 method for signature 'XString':
reverse(x, ...)
complement(x, ...)
reverseComplement(x, ...)

Arguments

x  A character vector, or an XString, XStringSet, XStringViews or MaskedXString
   object for reverse.
   A DNAString, RNAString, DNAStringSet, RNAStringSet, XStringViews (with
   DNAString or RNAString subject), MaskedDNAString or MaskedRNAString
   object for complement and reverseComplement.

...  Additional arguments to be passed to or from methods.
Details

Given an XString object \( x \), \( \text{reverse}(x) \) returns an object of the same XString base type as \( x \) where letters in \( x \) have been reordered in the reverse order.

If \( x \) is a DNAString or RNAString object, \( \text{complement}(x) \) returns an object where each base in \( x \) is "complemented" i.e. A, C, G, T in a DNAString object are replaced by T, G, C, A respectively and A, C, G, U in a RNAString object are replaced by U, G, C, A respectively.

Letters belonging to the "IUPAC extended genetic alphabet" are also replaced by their complement (M <-> K, R <-> Y, S <-> S, V <-> B, W <-> W, H <-> D, N <-> N) and the gap ("-") and hard masking ("+")) letters are unchanged.

\( \text{reverseComplement}(x) \) is equivalent to \( \text{reverse}(\text{complement}(x)) \) but is faster and more memory efficient.

Value

An object of the same class and length as the original object.

See Also

DNAString-class, RNAString-class, DNAStringSet-class, RNAStringSet-class, XStringViews-class, MaskedXString-class, chartr, findPalindromes

Examples

```r
## A. SOME SIMPLE EXAMPLES
x <- DNAString("ACGT-YN-")
reverseComplement(x)
```

```r
library(drosophila2probe)
probes <- DNAStringSet(drosophila2probe$sequence)
probes
alphabetFrequency(probes, collapse=TRUE)
rcprobes <- reverseComplement(probes)
rcprobes
alphabetFrequency(rcprobes, collapse=TRUE)
```

```r
## B. OBTAINING THE MISMATCH PROBES OF A CHIP
pm2mm <- function(probes)
{
  probes <- DNAStringSet(probes)
  subseq(probes, start=13, end=13) <- complement(subseq(probes, start=13, end=13))
  probes
}
mmprobes <- pm2mm(probes)
mmprobes
alphabetFrequency(mmprobes, collapse=TRUE)
```

```r
## C. SEARCHING THE MINUS STRAND OF A CHROMOSOME
```
library(BSgenome.Dmelanogaster.UCSC.dm3)
chrX <- Dmelanogaster$chrX
pattern <- DNAString("ACCAACNNGGTTG")
matchPattern(pattern, chrX, fixed=FALSE) # 3 hits on strand +
rcpattern <- reverseComplement(pattern)
rcpattern
m0 <- matchPattern(rcpattern, chrX, fixed=FALSE)
m0 # 5 hits on strand -

library(BSgenome.Hsapiens.UCSC.hg18)
chr1 <- Hsapiens$chr1
matchPattern(pattern, reverseComplement(chr1)) # DON'T DO THIS!
matchPattern(reverseComplement(pattern), chr1) # DO THIS INSTEAD

reverseSeq

Reverse Sequence

Description

WARNING: The functions described in this man page have been deprecated in favor of reverse, XString-method and reverseComplement.

Functions to obtain the reverse and reverse complement of a sequence

Usage

reverseSeq(seq)
revcompDNA(seq)
revcompRNA(seq)

Arguments

seq Character vector. For revcompRNA and revcompDNA the sequence should consist of appropriate letter codes: \[ACGUN\] and \[ACGTN\], respectively.

Details

The function reverses the order of the constituent character strings of its argument.

Value

A character vector of the same length as seq.

Author(s)

R. Gentleman, W. Huber, S. Falcon

See Also

alphabetFrequency, reverseComplement

Examples

w <- c("hey there", "you silly fool")
if (interactive()) {
  reverseSeq(w) # deprecated (inefficient on large vectors)
}
reverse(BStringSet(w)) # more efficient

w <- "able was I ere I saw Elba"
if (interactive()) {
  reverseSeq(w) # deprecated (inefficient on large vectors)
}
reverse(BStringSet(w)) # more efficient

rnal <- "UGCA"
if (interactive()) {
  revcompRNA(rnal) # deprecated (inefficient on large vectors)
}
reverseComplement(RNAString(rnal)) # more efficient

dnal <- "TGCA"
if (interactive()) {
  revcompDNA(dnal) # deprecated (inefficient on large vectors)
}
reverseComplement(DNAString(dnal)) # more efficient

## Comparing efficiencies:
if (interactive()) {
  library(hgu95av2probe)
  system.time(y1 <- reverseSeq(hgu95av2probe$sequence))
x <- DNAStringSet(hgu95av2probe$sequence)
  system.time(y2 <- reverse(x))
system.time(y3 <- revcompDNA(hgu95av2probe$sequence))
system.time(y4 <- reverseComplement(x))

### RNAString-class RNAString objects

#### Description
An RNAString object allows efficient storage and manipulation of a long RNA sequence.

#### Details
The RNAString class is a direct XString subclass (with no additional slot). Therefore all functions and methods described in the XString man page also work with an RNAString object (inheritance). Unlike the BString container that allows storage of any single string (based on a single-byte character set) the RNAString container can only store a string based on the RNA alphabet (see below). In addition, the letters stored in an RNAString object are encoded in a way that optimizes fast search algorithms.

#### The RNA alphabet
This alphabet contains all letters from the IUPAC Extended Genetic Alphabet (see ?IUPAC_CODE_MAP) where "T" is replaced by "U" + the gap ("-") and the hard masking ("+") letters. It is stored in the RNA_ALPHABET constant (character vector). The alphabet method also returns RNA_ALPHABET when applied to an RNAString object and is provided for convenience only.

#### Constructor-like functions and generics
In the code snippet below, x can be a single string (character vector of length 1), a BString object or a DNAString object.

RNAString(x='', start=1, nchar=NA): Tries to convert x into an RNAString object by reading nchar letters starting at position start in x.

#### Accessor methods
In the code snippet below, x is an RNAString object.

alphabet(x, baseOnly=FALSE): If x is an RNAString object, then return the RNA alphabet (see above). See the corresponding man pages when x is a BString, DNAString or AAString object.

#### Author(s)
H. Pages

#### See Also
IUPAC_CODE_MAP, letter, XString-class, DNAString-class, reverseComplement, alphabetFrequency
**stringDist**

*String Distance/Alignment Score Matrix*

**Examples**

```r
RNA_BASES
RNA_ALPHABET
d <- DNAString("TTGAAAA-CTC-N")
r <- RNAString(d)
r
alphabet(r)  # RNA_ALPHABET
alphabet(r, baseOnly=TRUE)  # RNA_BASES
```

```r
## When comparing an RNAString object with a DNAString object,
## U and T are considered equals:
## r == d  # TRUE
```

**Description**

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

**Usage**

```r
stringDist(x, method = "levenshtein", ignoreCase = FALSE, diag = FALSE, upper = FALSE, type = "global", quality = PhredQuality(22L), substitutionMatrix = NULL, fuzzyMatrix = NULL, gapOpening = 0, gapExtension = -1)
```

**Arguments**

- `x` a character vector or an `XStringSet` object.
- `method` calculation method. One of "levenshtein", "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 diagonal 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

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

Scoring matrices

Description

Predefined substitution 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)
nucleotideSubstitutionMatrix(match = 1, mismatch = 0, baseOnly = FALSE, type = "DNA")
qualitySubstitutionMatrices(fuzzyMatch = c(0, 1), alphabetLength = 4L, qualityClass = "PhredQuality", bitScale = 1)
errorSubstitutionMatrices(errorProbability, fuzzyMatch = c(0, 1), alphabetLength = 4L, qualityClass = "PhredQuality", 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".
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 either "PhredQuality" or "SolexaQuality".
bitScale a numeric value to scale the quality-based substitution matrices. By default, this is 1, representing bit-scale scoring.

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.
nucleotideSubstitutionMatrix produces a substitution matrix for all IUPAC nucleic acid codes based upon match and mismatch parameters.
errorSubstitutionMatrices produces a two element list of numeric square symmetric matrices, one for matches and one for mismatches.
qualitySubstitutionMatrices produces the substitution matrices for Phred or Solexa quality-based reads.

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
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/matrices/
The quality matrices computed in qualitySubstitutionMatrices are based on the paper by Ketil Malde. Let $\epsilon_i$ 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 $\epsilon_c = \epsilon_1 + \epsilon_2 - (n/(n-1)) \times \epsilon_1 \times \epsilon_2$, where $n$ is the number of letters in the underlying alphabet. Using $\epsilon_c$, the substitution score is given by when two bases match is given by $b \times \log_2(\gamma_{x,y} \times (1-\epsilon_c) \times n + (1-\gamma_{x,y}) \times \epsilon_c \times (n/(n-1)))$, 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 $\epsilon_i$.

Author(s)
H. Pages and P. Aboyoun

References

See Also
pairwiseAlignment, PairwiseAlignedXStringSet-class, DNAString-class, AAString-class, PhredQuality-class, SolexaQuality-class

Examples
```r
s1 <- DNAString("ACTTCACCAGCTCCCTTGGCGGTAAGTGTGTAAGGAAACGGCAAGTTTTCAG")
s2 <- DNAString("GTTTCACTACTTCCTTTCGGGTAAGTAAATATATAAATATATAAAAATATAATTTTTCATC")

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

## 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)
```
subXString <- subXString(x, start=NA, end=NA, length=NA)

## S4 method for signature 'XString'
substr(x, start=NA, stop=NA)

## S4 method for signature 'XString'
substring(text, first=NA, last=NA)

### Arguments

**x**  
An XString object for subXString. A character vector, an XStringViews, XString, or MaskedXString object for substr or substring.

**start**  
A numeric vector.

**end**  
A numeric vector.

**length**  
A numeric vector.

**stop**  
A numeric vector.

**text**  
A character vector, an XStringViews or an XString object.

**first**  
A numeric vector.

**last**  
A numeric vector.

### Description

Functions for fast substring extraction.

### Usage

subXString(x, start=NA, end=NA, length=NA)

## S4 method for signature 'XString'
substr(x, start=NA, stop=NA)

## S4 method for signature 'XString'
substring(text, first=NA, last=NA)
toComplex

Details

subXString is deprecated in favor of subseq.

Value

An XString object of the same base type as x for subXString.

A character vector for substr and substring.

See Also

subseq, letter, XString-class, XStringViews-class

toComplex

Turning a DNA sequence into a vector of complex numbers

Description

The toComplex utility function turns a DNAString object into a complex vector.

Usage

toComplex(x, baseValues)

Arguments

x

A DNAString object.

baseValues

A named complex vector containing the values associated to each base e.g.
c(A=1+0i, G=0+1i, T=-1+0i, C=0-1i)

Value

A complex vector of the same length as x.

Author(s)

H. Pages

See Also

DNAString

Examples

seq <- DNAString("accacctgaccattgtcct")
baseValues1 <- c(A=1+0i, G=0+1i, T=-1+0i, C=0-1i)
toComplex(seq, baseValues1)

## GC content:
baseValues2 <- c(A=0, C=1, G=1, T=0)
sum(as.integer(toComplex(seq, baseValues2)))

## Note that there are better ways to do this (see ?alphabetFrequency)
DNA/RNA transcription and translation

Description

Functions for transcription and/or translation of DNA or RNA sequences, and related utilities.

Usage

- `transcribe(x)`
- `cDNA(x)`
- `codons(x)`
- `translate(x)`

## Related utilities

- `dna2rna(x)`
- `rna2dna(x)`

Arguments

- `x`: A DNAString object for `transcribe` and `dna2rna`. An RNAString object for `cDNA` and `rna2dna`. A DNAString, RNAString, MaskedDNAString or MaskedRNAString object for `codons`. A DNAString, RNAString, DNAStringSet, RNAStringSet, MaskedDNAString or MaskedRNAString object for `translate`.

Details

- `transcribe` reproduces the biological process of DNA transcription that occurs in the cell.
- `cDNA` reproduces the process of synthesizing complementary DNA from a mature mRNA template.
- `translate` reproduces the biological process of RNA translation that occurs in the cell. The input of the function can be either RNA or coding DNA. The Standard Genetic Code (see ?GENETIC_CODE) is used to translate codons into amino acids. `codons` is a utility for extracting the codons involved in this translation without translating them.
- `dna2rna` and `rna2dna` are low-level utilities for converting sequences from DNA to RNA and vice-versa. All what this conversion does is to replace each occurrence of T by a U and vice-versa.

Value

- An RNAString object for `transcribe` and `dna2rna`.
- A DNAString object for `cDNA` and `rna2dna`.

Note that if the sequence passed to `transcribe` or `cDNA` is considered to be oriented 5’-3’, then the returned sequence is oriented 3’-5’.

An XStringViews object with 1 view per codon for `codons`. When `x` is a MaskedDNAString or MaskedRNAString object, its masked parts are interpreted as introns and filled with the + letter in the returned object. Therefore codons that span across masked regions are represented by views that have a width > 3 and contain the + letter. Note that each view is guaranteed to contain exactly 3 base letters.

An AAString object for `translate`. 

trimLRPatterns

Trim Flanking Patterns from Sequences

Description

The trimLRPatterns function trims left and/or right flanking patterns from sequences.

Usage

trimLRPatterns(Lpattern = "", Rpattern = "", subject, max.Lmismatch = 0, max.Rmismatch = 0, with.Lindels = FALSE, with.Rindels = FALSE, Lfixed = TRUE, Rfixed = TRUE, ranges = FALSE)

Arguments

Lpattern  The left part of the pattern.
Rpattern  The right part of the pattern.
subject   An XString or XStringSet object containing the target sequence(s).
max.Lmismatch  Either an integer vector of length nLp = nchar(Lpattern) whose elements max.Lmismatch[i] represent the maximum number of acceptable mismatching letters when aligning substring(Lpattern, nLp - i +
trimLRPatterns

1, nLp) with substring(subject, 1, i) or a single numeric value in (0, 1) that represents a constant maximum mismatch rate for each of the nL alignments. Negative numbers in integer vector inputs are used to prevent trimming at the i-th location. If an integer vector input has length(max.Lmismatch) < nLp, then max.Lmismatch will be augmented with enough -1's at the beginning of the vector to bring it up to length nLp.

If non-zero, an inexact matching algorithm is used (see the matchPattern function for more information).

max.Rmismatch

Either an integer vector of length nRp = nchar(Rpattern) whose elements max.Rmismatch[i] represent the maximum number of acceptable mismatching letters when aligning substring(Rpattern, nRp - i + 1, nRp) with substring(subject, 1, i) or a single numeric value in (0, 1) that represents a constant maximum mismatch rate for each of the nR alignments. Negative numbers in integer vector inputs are used to prevent trimming at the i-th location. If an integer vector input has length(max.Rmismatch) < nRp, then max.Rmismatch will be augmented with enough -1's at the beginning of the vector to bring it up to length nRp.

If non-zero, an inexact matching algorithm is used (see the matchPattern function for more information).

with.Lindels

If TRUE then indels are allowed in the left part of the pattern. In that case max.Lmismatch is interpreted as the maximum "edit distance" allowed in the left part of the pattern.

See the with.indels argument of the matchPattern function for more information.

with.Rindels

Same as with.Lindels but for the right part of the pattern.

Lfixed

Only with a DNAString or RNAString subject can a Lfixed value other than the default (TRUE) be used.

With Lfixed=FALSE, ambiguities (i.e. letters from the IUPAC Extended Genetic Alphabet (see IUPAC_CODE_MAP) that are not from the base alphabet) in the left pattern and in the subject are interpreted as wildcards i.e. they match any letter that they stand for.

See the fixed argument of the matchPattern function for more information.

Rfixed

Same as Lfixed but for the right part of the pattern.

ranges

If TRUE, then return the ranges to use to trim subject. If FALSE, then returned the trimmed subject.

Value

A new XString or XStringSet object with the flanking patterns within the specified edit distances removed.

Author(s)

P. Aboyoun

See Also

matchPattern, matchLRPatterns, match-utils, XString-class, XStringSet-class
Examples

```r
Lpattern <- "TTCTGCTTG"
Rpattern <- "GATCGGAAG"
subject <- DNAString("TTCTGCTTGACGTGATCGGA")
subjectSet <- DNAStringSet(c("TGCTTGACGCACTGAGG", "TTCTGCTTGACGTGATCGGA"))

## Only allow for perfect matches on the flanks
trimLRPatterns(Lpattern = Lpattern, subject = subject)
trimLRPatterns(Rpattern = Rpattern, subject = subject)
trimLRPatterns(Lpattern = Lpattern, Rpattern = Rpattern, subject = subjectSet)

## Allow for perfect matches on the flanking overlaps
trimLRPatterns(Lpattern = Lpattern, Rpattern = Rpattern, subject = subjectSet,
               max.Lmismatch = rep(0, 9), max.Rmismatch = rep(0, 9))

## Allow for mismatches on the flanks
trimLRPatterns(Lpattern = Lpattern, Rpattern = Rpattern, subject = subject,
               max.Lmismatch = 0.2, max.Rmismatch = 0.2)
maxMismatches <- as.integer(0.2 * 1:9)
maxMismatches
trimLRPatterns(Lpattern = Lpattern, Rpattern = Rpattern, subject = subjectSet,
               max.Lmismatch = maxMismatches, max.Rmismatch = maxMismatches)

## Produce ranges that can be an input into other functions
trimLRPatterns(Lpattern = Lpattern, Rpattern = Rpattern, subject = subjectSet,
               max.Lmismatch = rep(0, 9), max.Rmismatch = rep(0, 9),
               ranges = TRUE)
trimLRPatterns(Lpattern = Lpattern, Rpattern = Rpattern, subject = subject,
               max.Lmismatch = 0.2, max.Rmismatch = 0.2, ranges = TRUE)
```

---

**xscat**

| Concatenate sequences contained in XString, XStringSet and/or XStringViews objects |

---

Description

This function mimics the semantic of `paste(..., sep="")` but accepts XString, XStringSet or XStringViews arguments and returns an XString or XStringSet object.

Usage

```r
xscat(...)```

Arguments

... One or more character vectors (with no NAs), XString, XStringSet or XStringViews objects.

Value

An XString object if all the arguments are either XString objects or character strings. An XStringSet object otherwise.
## Return a BString object:
```r
xscat(BString("abc"), BString("EF"))
xscat(BString("abc"), "EF")
xscat("abc", "EF")
```

## Return a BStringSet object:
```r
xscat(BStringSet("abc"), "EF")
```

## Return a DNAStringSet object:
```r
xscat(c("t", "a"), DNAString("N"))
```

## Arguments are recycled to the length of the longest argument:
```r
xscat("x", LETTERS, c("3", "44", "555"))
```

## Concatenating big XStringSet objects:
```r
library(drosophila2probe)
probes <- DNAStringSet(drosophila2probe$sequence)
mm <- complement(narrow(probes, start=13, end=13))
left <- narrow(probes, end=12)
right <- narrow(probes, start=14)
xscat(left, mm, right)
```

## Collapsing an XStringSet (or XStringViews) object with a small number of elements:
```r
probes1000 <- as.list(probes[1:1000])
y1 <- do.call(xscat, probes1000)
y2 <- do.call(c, probes1000) # slightly faster than the above
y1 == y2 # TRUE
```

## Note that this method won't be efficient when the number of elements to collapse is big (> 10000) so we need to provide a collapse() (or xscollapse()) function in Biostrings that will be efficient at doing this. Please complain on the Bioconductor mailing list (http://bioconductor.org/docs/mailList.html) if you need this.

---

## Description

The `BString` class is a general container for storing a big string (a long sequence of characters) and for making its manipulation easy and efficient.

The `DNAString`, `RNAString` and `AAString` classes are similar containers but with the more biology-oriented purpose of storing a DNA sequence (`DNAString`), an RNA sequence (`RNAString`), or a sequence of amino acids (`AAString`).

---

XString-class  

Author(s)

H. Pages

See Also

`XString-class`, `XStringSet-class`, `XStringViews-class`, `paste`
All those containers derive directly (and with no additional slots) from the XString virtual class.

Details

The 2 main differences between an XString object and a standard character vector are: (1) the data stored in an XString object are not copied on object duplication and (2) an XString object can only store a single string (see the XStringSet container for an efficient way to store a big collection of strings in a single object).

Unlike the DNAString, RNAString and AAString containers that accept only a predefined set of letters (the alphabet), a BString object can be used for storing any single string based on a single-byte character set.

Constructor-like functions and generics

In the code snippet below, \( x \) can be a single string (character vector of length 1) or an XString object.

\[
\text{BString}(x=\text{""}, \text{start}=1, \text{nchar}=\text{NA}): \text{Tries to convert } x \text{ into a BString object by reading } nchar \text{ letters starting at position } \text{start} \text{ in } x.
\]

Accessor methods

In the code snippets below, \( x \) is an XString object.

\[
\text{alphabet}(x): \text{NULL for a BString object. See the corresponding man pages when } x \text{ is a DNAString, RNAString or AAString object.}
\]

\[
\text{length}(x) \text{ or } \text{nchar}(x): \text{Get the length of an XString object, i.e., its number of letters.}
\]

Coercion

In the code snippets below, \( x \) is an XString object.

\[
\text{as.character}(x): \text{Converts } x \text{ to a character string.}
\]

\[
\text{toString}(x): \text{Equivalent to as.character}(x).
\]

Subsetting

In the code snippets below, \( x \) is an XString object.

\[
 x[i]: \text{Return a new XString object made of the selected letters (subscript } i \text{ must be an NA-free numeric vector specifying the positions of the letters to select). The returned object belongs to the same class as } x. \\
\text{Note that, unlike subseq, } x[i] \text{ does copy the sequence data and therefore will be very inefficient for extracting a big number of letters (e.g. when } i \text{ contains millions of positions).}
\]

Equality

In the code snippets below, \( e1 \) and \( e2 \) are XString objects.

\[
e1 == e2: \text{TRUE if } e1 \text{ is equal to } e2. \text{FALSE otherwise.}
\]

Comparison between two XString objects of different base types (e.g. a BString object and a DNAString object) is not supported with one exception: a DNAString object and an RNAString object can be compared (see RNAString-class for more details about this).

Comparison between a BString object and a character string is also supported (see examples below).
XStringPartialMatches-class

```r
  e1 != e2: Equivalent to !(e1 == e2).
```

**Author(s)**

H. Pages

**See Also**

subseq, letter, DNAString-class, RNAString-class, AAString-class, XStringSet-class, XStringViews-class, reverse, XString-method

**Examples**

```r
b <- BString("I am a BString object")
b
length(b)

## Extracting a linear subsequence
subseq(b)
subseq(b, start=3)
subseq(b, start=-3)
subseq(b, end=-3)
subseq(b, end=-3, width=5)

## Subsetting
b2 <- b[length(b):1]       # better done with reverse(b)
as.character(b2)

b2 == b                   # FALSE
b2 == as.character(b2)    # TRUE

## b[1:length(b)] is equal but not identical to b!
b == b[1:length(b)]        # TRUE
identical(b, 1:length(b))  # FALSE

## This is because subsetting an XString object with [ makes a copy
## of part or all its sequence data. Hence, for the resulting object,
## the internal slot containing the memory address of the sequence
## data differs from the original. This is enough for identical() to
## see the 2 objects as different.
```

---

**XStringPartialMatches-class**

**XStringPartialMatches objects**

**Description**

WARNING: This class is currently under development and might not work properly! Full documentation will come later.

Please DO NOT TRY TO USE it for now. Thanks for your comprehension!
Accessor methods

In the code snippets below, \( x \) is an XStringPartialMatches object.

\[
\text{subpatterns}(x): \text{Not ready yet.}
\]
\[
\text{pattern}(x): \text{Not ready yet.}
\]

Standard generic methods

In the code snippets below, \( x \) is an XStringPartialMatches objects, and \( i \) can be a numeric or logical vector.

\[
x[i]: \text{Return a new XStringPartialMatches object made of the selected views.} \ i \ \text{can be a numeric vector, a logical vector, NULL or missing. The returned object has the same subject as} \ x.
\]

Author(s)

H. Pages

See Also

XStringViews-class, XString-class, letter

---

**XStringQuality-class**

*PhredQuality and SolexaQuality objects*

Description

Objects for storing string quality measures.

Usage

```r
## Constructors:
PhredQuality(x)
SolexaQuality(x)
```

Arguments

\( x \)Either a character vector, BString, BStringSet, integer vector, or number vector of error probabilities.

Details

*PhredQuality objects store characters that are interpreted as \([0 - 99]\) quality measures by subtracting 33 from their ASCII decimal representation (e.g. ! = 0, " = 1, # = 2, ...).*

*SolexaQuality objects store characters are interpreted as \([-5 - 99]\) quality measures by subtracting 64 from their ASCII decimal representation (e.g. ; = -5, < = -4, = = -3, ...).*

Author(s)

P. Aboyoun
XStringSet-class

See Also

pairwiseAlignment, PairwiseAlignedXStringSet-class, DNAString-class, BStringSet-class

Examples

PhredQuality(0:40)
SolexaQuality(0:40)

PhredQuality(seq(1e-4,0.5,length=10))
SolexaQuality(seq(1e-4,0.5,length=10))

XStringSet-class  BStringSet, DNAStringSet, RNAStringSet and AAStringSet objects

Description

The BStringSet class is a container for storing a set of BString objects and for making its manipulation easy and efficient.

Similarly, the DNAStringSet (or RNAStringSet, or AAStringSet) class is a container for storing a set of DNAString (or RNAString, or AAString) objects.

All those containers derive directly (and with no additional slots) from the XStringSet virtual class.

Usage

## Constructors:
BStringSet(x=character(), start=NA, end=NA, width=NA, use.names=TRUE)
DNAStringSet(x=character(), start=NA, end=NA, width=NA, use.names=TRUE)
RNAStringSet(x=character(), start=NA, end=NA, width=NA, use.names=TRUE)
AAStringSet(x=character(), start=NA, end=NA, width=NA, use.names=TRUE)

## Accessor-like methods:
## S4 method for signature 'XStringSet':
length(x)
## S4 method for signature 'character':
width(x)
## S4 method for signature 'XStringSet':
width(x)
## S4 method for signature 'XStringSet':
names(x)
## S4 method for signature 'XStringSet':
nchar(x, type="chars", allowNA=FALSE)

## Efficient subsequence extraction:
## S4 method for signature 'character':
subseq(x, start=NA, end=NA, width=NA)
## S4 method for signature 'XStringSet':
subseq(x, start=NA, end=NA, width=NA)

## ... and more (see below)
Arguments

- `x`: Either a character vector (with no NAs), or an `XString`, `XStringSet` or `XStringViews` object.
- `start, end, width`: Either NA, a single integer, or an integer vector of the same length as `x` specifying how `x` should be "narrowed" (see `?narrow` for the details).
- `use.names`: TRUE or FALSE. Should names be preserved?
- `type, allowNA`: Ignored.

Details

The `BStringSet`, `DNAStringSet`, `RNAStringSet` and `AAStringSet` functions are constructors that can be used to "naturally" turn `x` into an `XStringSet` object of the desired base type. They also allow the user to "narrow" the sequences contained in `x` via proper use of the `start, end` and/or `width` arguments. In this context, "narrowing" means dropping a prefix or/and a suffix of each sequence in `x`. The "narrowing" capabilities of these constructors can be illustrated by the following property: if `x` is a character vector (with no NAs), or an `XStringSet` (or `XStringViews`) object, then the 3 following transformations are equivalent:

```r
BStringSet(x, start=mystart, end=myend, width=mywidth)
subseq(BStringSet(x), start=mystart, end=myend, width=mywidth)
BStringSet(subseq(x, start=mystart, end=myend, width=mywidth))
```

Note that, besides being more convenient, the first form is also more efficient on character vectors.

Accessor-like methods

In the code snippets below, `x` is an `XStringSet` object.

- `length(x)`: The number of sequences in `x`.
- `width(x)`: A vector of non-negative integers containing the number of letters for each element in `x`. Note that `width(x)` is also defined for a character vector with no NAs and is equivalent to `nchar(x, type="bytes")`.
- `names(x)`: NULL or a character vector of the same length as `x` containing a short user-provided description or comment for each element in `x`. These are the only data in an `XStringSet` object that can safely be changed by the user. All the other data are immutable! As a general recommendation, the user should never try to modify an object by accessing its slots directly.
- `alphabet(x)`: Return NULL, `DNA_ALPHABET`, `RNA_ALPHABET` or `AA_ALPHABET` depending on whether `x` is a `BStringSet`, `DNAStringSet`, `RNAStringSet` or `AAStringSet` object.
- `nchar(x)`: The same as `width(x)`.

Subsequence extraction and related transformations

In the code snippets below, `x` is a character vector (with no NAs), or an `XStringSet` (or `XStringViews`) object.

- `subseq(x, start=NA, end=NA, width=NA)`: Applies `subseq` on each element in `x`. See `?subseq` for the details.

Note that this is similar to what `substr` does on a character vector. However there are some noticeable differences: (1) the arguments are `start` and `stop` for `substr`; (2) the SEW interface (start/end/width) interface of `subseq` is richer (e.g. support for negative start or end
values); and (3) `subseq` checks that the specified start/end/width values are valid i.e., unlike `substr`, it throws an error if they define "out of limits" subsequences or subsequences with a negative width.

```r
narrow(x, start=NA, end=NA, width=NA, use.names=TRUE): Same as `subseq`. The only differences are: (1) `narrow` has a `use.names` argument; and (2) all the things `narrow` and `subseq` work on (IRanges, XStringSet or XStringViews objects for `narrow`, XSequence or XStringSet objects for `subseq`). But they both work and do the same thing on an XStringSet object.
```

```r
dthreebands(x, start=NA, end=NA, width=NA): Like the method for IRanges objects, the threebands methods for character vectors and XStringSet objects extend the capability of `narrow` by returning the 3 set of subsequences (the left, middle and right subsequences) associated to the narrowing operation. See ?threebands in the IRanges package for the details.
```

```r
subseq<- (x, start=NA, end=NA, width=NA) <- value: A vectorized version of the `subseq<-` method for XSequence objects. See ?subseq<- for the details.
```

### Subsetting and appending

In the code snippets below, `x` and `values` are XStringSet objects, and `i` should be an index specifying the elements to extract.

```r
x[i]: Return a new XStringSet object made of the selected elements.
x[[i]]: Extract the i-th XString object from x.
append(x, values, after=length(x)): Add sequences in `values` to `x`.
```

### Ordering and related methods

In the code snippets below, `x` is an XStringSet object.

```r
order(x): Return a permutation which rearranges `x` into ascending or descending order.
sort(x): Sort `x` into ascending order (equivalent to `x[order(x)]`).
rank(x): Rank `x` in ascending order.
```

### Duplicated and unique methods

In the code snippets below, `x` is an XStringSet object.

```r
duplicated(x): Return a logical vector whose elements denotes duplicates in `x`.
unique(x): Return an XStringSet containing the unique values in `x`.
```

### Set operations

In the code snippets below, `x` and `y` are XStringSet objects

```r
union(x, y): Union of `x` and `y`.
intersect(x, y): Intersection of `x` and `y`.
setdiff(x, y): Asymmetric set difference of `x` and `y`.
setequal(x, y): Set equality of `x` to `y`.
```
Identical value matching

In the code snippets below, \( x \) is a character vector, XString, or XStringSet object and \( \text{table} \) is an XStringSet object.

- \( x \; \in \% \in \% \; \text{table} \): Returns a logical vector indicating which elements in \( x \) match identically with an element in \( \text{table} \).

- \( \text{match}(x, \; \text{table}, \; \text{nomatch} = \text{NA\_integer\_}, \; \text{incomparables} = \text{NULL}) \): Returns an integer vector containing the first positions of an identical match in \( \text{table} \) for the elements in \( x \).

Other methods

In the code snippets below, \( x \) is an XStringSet object.

- \( \text{unlist}(x) \): Turns \( x \) into an XString object by combining the sequences in \( x \) together. Fast equivalent to \( \text{do.call(c, as.list}(x)) \).
- \( \text{as.character}(x, \; \text{use.names}) \): Convert \( x \) to a character vector of the same length as \( x \). \text{use.names} controls whether or not \( \text{names}(x) \) should be used to set the names of the returned vector (default is TRUE).
- \( \text{as.matrix}(x, \; \text{use.names}) \): Return a character matrix containing the "exploded" representation of the strings. This can only be used on an XStringSet object with equal-width strings. \text{use.names} controls whether or not \( \text{names}(x) \) should be used to set the row names of the returned matrix (default is TRUE).
- \( \text{toString}(x) \): Equivalent to \( \text{toString}(\text{as.character}(x)) \).

Author(s)

H. Pages

See Also

BString-class, DNAString-class, RNAString-class, AAString-class, XStringViews-class, \texttt{substr}, \texttt{subseq}, \texttt{narrow}

Examples

```r
data <- c("#CTC-NACCAGTAT", 
"#TTGA", 
"TACCTAGAG")
width(data)
x1 <- BStringSet(data)
x1

# 3 equivalent ways to obtain the same BStringSet object:
BStringSet(x1, start=4, end=-3)
subseq(x1, start=4, end=-3)
BStringSet(subseq(x0, start=4, end=-3))
dna0 <- DNAStringSet(x0, start=4, end=-3)
```
### B. USING THE XStringSet CONSTRUCTORS ON AN XStringSet OBJECT

```r
library(drosophila2probe)
probes <- DNAStringSet(drosophila2probe$sequence)
probes
```

```r
RNAStringSet(probes, start=2, end=-5) # does NOT copy the sequence data!
```

### C. USING subseq() ON AN XStringSet OBJECT

```r
subseq(probes, start=2, end=-5)
```

```r
subseq(probes, start=13, end=13) <- "N"
```

```r
subseq(probes, start=1, end=0) <- "--"
```

```r
subseq(probes, end=2) <- ""
```

```r
subseq(probes, start=-2:-5)
```

```r
## Add/remove a prefix:
subseq(probes, start=1, end=0) <- "--"
subseq(probes, end=2) <- ""
```

```r
## Do more complicated things:
subseq(probes, start=4:7, end=7) <- c("YYYY", "YYY", "YY", "Y")
subseq(probes, start=4, end=6) <- subseq(probes, start=-2:-5)
```

### D. UNLISTING AN XStringSet OBJECT

```r
library(drosophila2probe)
probes <- DNAStringSet(drosophila2probe$sequence)
unlist(probes)
```

---

**XStringSet-io**  
Read/write an XStringSet or XStringViews object from/to a file

### Description

Functions to read/write an XStringSet or XStringViews object from/to a file.

### Usage

```r
## XStringSet object:
read.BStringSet(file, format)
read.DNAStringSet(file, format)
read.RNAStringSet(file, format)
read.AAStringSet(file, format)
```
write.XStringSet(x, file="", append=FALSE, format, width=80)

## XStringViews object:
read.XStringViews(file, format, subjectClass, collapse="")
write.XStringViews(x, file="", append=FALSE, format, width=80)

## FASTQ utilities:
fastq.geometry(file)

## Some related helper functions:
FASTArecordsToCharacter(FASTArecs, use.names=TRUE)
CharacterToFASTArecords(x)
FASTArecordsToXStringViews(FASTArecs, subjectClass, collapse="")
XStringSetToFASTArecords(x)

Arguments

- **file**: A character vector with no NAs. If "" (the default for write.XStringSet and write.XStringViews), then the functions write to the standard output connection (the console) unless redirected by sink.
- **format**: Either "fasta" or "fastq". Note that write.XStringSet and write.XStringViews only support "fasta" for now.
- **x**: For write.XStringSet and write.XStringViews, the object to write to file. For CharacterToFASTArecords, the (possibly named) character vector to be converted to a list of FASTA records as one returned by readFASTA. For XStringSetToFASTArecords, the XStringSet object to be converted to a list of FASTA records as one returned by readFASTA.
- **append**: TRUE or FALSE. If TRUE output will be appended to file; otherwise, it will overwrite the contents of file. See ?cat for the details.
- **width**: Only relevant if format is "fasta". The maximum number of letters per line of sequence.
- **subjectClass**: The class to be given to the subject of the XStringViews object created and returned by the function. Must be the name of one of the direct XString subclasses i.e. "BString", "DNAString", "RNAString" or "AAString".
- **collapse**: An optional character string to be inserted between the views of the XStringViews object created and returned by the function.
- **FASTArecs**: A list of FASTA records as one returned by readFASTA.
- **use.names**: Whether or not the description line preceding each FASTA records should be used to set the names of the returned object.

Details

Only FASTA and FASTQ files are supported for now. The identifiers and qualities stored in the FASTQ records are ignored (only the sequences are returned).

Reading functions read.BStringSet, read.DNAStringSet, read.RNAStringSet, read.AAStringSet and read.XStringViews load sequences from a file into an XStringSet or XStringViews object. Only one FASTA file, but more than one FASTQ file, can be read at a time (by passing a character vector of length > 1). In that case, all the FASTQ files must have the same "width" (i.e. all their sequences must have the same length) and the sequences from all the files are stored in the returned object in the order they were read.
The `fastq.geometry` utility returns an integer vector describing the "geometry" of the FASTQ files i.e. a vector of length 2 where the first element is the total number of sequences contained in the FASTQ files and the second element the "width" of these files (this width is NA if the files have different "widths").

Writing functions `write.XStringSet` and `write.XStringViews` write an `XStringSet` or `XStringViews` object to a file or connection. They only support the FASTA format for now.

`FASTArecordsToCharacter`, `CharacterToFASTArecords`, `FASTArecordsToXStringViews` and `XStringSetToFASTArecords` are helper functions used internally by `write.XStringSet` and `read.XStringViews` for switching between different representations of the same object.

See Also

`fasta.info`, `readFASTA`, `writeFASTA`, `XStringSet-class`, `XStringViews-class`, `BString-class`, `DNAString-class`, `RNAString-class`, `AAString-class`

Examples

```r
## A. READ/WRITE FASTA FILES
file <- system.file("extdata", "someORF.fa", package="Biostrings")
x <- read.DNAStringSet(file, "fasta")
x
write.XStringSet(x, format="fasta") # writes to the console

## B. READ FASTQ FILES
file <- system.file("extdata", "s_1_sequence.txt", package="Biostrings")
fastq.geometry(file)
read.DNAStringSet(file, "fastq") # only the FASTQ sequences are returned # (identifiers and qualities are dropped)

## C. SOME RELATED HELPER FUNCTIONS
# Converting 'x'...
# ... to a list of FASTA records (as one returned by the "readFASTA" function)
x1 <- XStringSetToFASTArecords(x)
# ... to a named character vector
x2 <- FASTArecordsToCharacter(x1) # same as 'as.character(x)'
```

The `XStringViews-class` **The `XStringViews` class**

**Description**

The `XStringViews` class is the basic container for storing a set of views (start/end locations) on the same sequence (an `XString` object).
Details

An XStringViews object contains a set of views (start/end locations) on the same XString object called "the subject string" or "the subject sequence" or simply "the subject". Each view is defined by its start and end locations: both are integers such that start <= end. An XStringViews object is in fact a particular case of an Views object (the XStringViews class contains the Views class) so it can be manipulated in a similar manner: see ?Views for more information. Note that two views can overlap and that a view can be "out of limits" i.e. it can start before the first letter of the subject or/and end after its last letter.

Constructor

Views(subject, start=NULL, end=NULL, width=NULL, names=NULL): See ?Views in the IRanges package for the details.

Accessor-like methods

All the accessor-like methods defined for Views objects work on XStringViews objects. In addition, the following accessors are defined for XStringViews objects:

nchar(x): A vector of non-negative integers containing the number of letters in each view. Values in nchar(x) coincide with values in width(x) except for "out of limits" views where they are lower.

Other methods

In the code snippets below, x, object, e1 and e2 are XStringViews objects, and i can be a numeric or logical vector.

e1 == e2: A vector of logicals indicating the result of the view by view comparison. The views in the shorter of the two XStringViews object being compared are recycled as necessary. Like for comparison between XString objects, comparison between two XStringViews objects with subjects of different classes is not supported with one exception: when the subjects are DNAString and RNAString instances.

Also, like with XString objects, comparison between an XStringViews object with a BString subject and a character vector is supported (see examples below).

e1 != e2: Equivalent to !(e1 == e2).

as.character(x, use.names, check.limits): Convert x to a character vector of the same length as x. use.names controls whether or not names(x) should be used to set the names of the returned vector (default is TRUE), check.limits controls whether or not an error should be raised if x contains "out of limit" views (default is TRUE). With check.limits=FALSE then "out of limit" views are padded with spaces.

as.matrix(x, mode, use.names, check.limits): Depending on what mode is chosen ("integer" or "character"), return either a 2-column integer matrix containing start(x) and end(x) or a character matrix containing the "exploded" representation of the views. mode="character" can only be used on an XStringViews object with equal-width views. Arguments use.names and check.limits are ignored with mode="integer". With mode="character", use.names controls whether or not names(x) should be used to set the row names of the returned matrix (default is TRUE), and check.limits controls whether or not an error should be raised if x contains "out of limit" views (default is TRUE). With check.limits=FALSE then "out of limit" views are padded with spaces.

toString(x): Equivalent to toString(as.character(x)).
Author(s)

H. Pages

See Also

Views-class, gaps, XStringViews-constructors, XString-class, XStringSet-class, letter, MIndex-class

Examples

## One standard way to create an XStringViews object is to use
## the Views() constructor.

## Views on a DNAString object:
s <- DNAString("-CTC-N")
v4 <- Views(s, start=3:0, end=5:8)
v4
subject(v4)
length(v4)
start(v4)
end(v4)
width(v4)

## Attach a comment to views #3 and #4:
names(v4)[3:4] <- "out of limits"
names(v4)

## A more programatical way to "tag" the "out of limits" views:
names(v4)[start(v4) < 1 | nchar(subject(v4)) < end(v4)] <- "out of limits"
## or just:
names(v4)[nchar(v4) < width(v4)] <- "out of limits"

## Two equivalent ways to extract a view as an XString object:
s2a <- v4[[2]]
s2b <- subseq(subject(v4), start=start(v4)[2], end=end(v4)[2])
identical(s2a, s2b) # TRUE

## It is an error to try to extract an "out of limits" view:
##v4[[3]] # Error!

v12 <- Views(DNAString("TAATAATG"), start=-2:9, end=0:11)
v12 <- DNAString("TAA")
v12[v12 == v12[4]]
v12[v12 == v12[1]]
v12[3] <- Views(RNAString("AU"), start=0, end=2)

## Here the first view doesn't even overlap with the subject:
Views(BString("aaa--b"), start=-3:4, end=-3:4 + c(3:6, 6:3))

## 'start' and 'end' are recycled:
subject <- "abcdefg hij"
Views(subject, start=2:1, end=4)
Views(subject, start=5:7, end=nchar(subject))
Views(subject, start=1, end=5:7)

## Applying gaps() to an XStringViews object:
v2 <- Views("abCDefgHIJK", start=c(8, 3), end=c(14, 4))
gaps(v2)

## Coercion:
as(v2, "XStringSet") # same as 'as(v12, "DNAStringSet")'
as(v2, "RNAStringSet")

---

**XStringViews-constructors**

*Basic functions for creating or modifying XStringViews objects*

**Description**

A set of basic functions for creating or modifying XStringViews objects.

**Usage**

```r
adjacentViews(subject, width, gapwidth=0)
XStringViews(x, subjectClass, collapse="")
```

**Arguments**

- `subject`: An `XString` object or a single string.
- `width`: An integer vector containing the widths of the views.
- `gapwidth`: An integer vector containing the widths of the gaps between the views.
- `x`: An `XString` object or a character vector for `XStringViews`.
- `subjectClass`: The class to be given to the subject of the `XStringViews` object created and returned by the function. Must be the name of one of the direct `XString` subclasses i.e. "BString", "DNAString", "RNAString" or "AAString".
- `collapse`: An optional character string to be inserted between the views of the `XStringViews` object created and returned by the function.

**Details**

The `adjacentViews` function returns an `XStringViews` object containing views on `subject` with widths given in the `width` vector and separated by gaps of width `gapwidth`. The first view starts at position 1.

The `XStringViews` constructor will try to create an `XStringViews` object from the value passed to its `x` argument. If `x` itself is an `XStringViews` object, the returned object is obtained by coercing its subject to the class specified by `subjectClass`. If `x` is an `XString` object, the returned object is made of a single view that starts at the first letter and ends at the last letter of `x` (in addition `x` itself is coerced to the class specified by `subjectClass` when specified). If `x` is a character vector, the returned object has one view per character string in `x` (and its subject is an instance of the class specified by `subjectClass`).

**Value**

These functions return an `XStringViews` object `y`. `length(y)` (the number of views in `y`) is `length(width)` for the `adjacentViews` function. For the `XStringViews` constructor, `length(y)` is 1 when `x` is an `XString` object and `length(x)` otherwise.
yeastSEQCHR1

 Assumes

See Also

XStringViews-class, XString-class

Examples

```r
callViews("abcdefghij", 4:2, gapwidth=1)

v12 <- Views(DNAString("TAATAATG"), start=-2:9, end=0:11)
XStringViews(v12, subjectClass="RNAString")
XStringViews(AAString("MARKSLEMSIR*"))
XStringViews("abcdefghij", subjectClass="BString")
```

Description

This is a single character string containing DNA sequence of yeast chromosome number 1. The data were obtained from the Saccharomyces Genome Database (ftp://genome-ftp.stanford.edu/pub/yeast/data_download/sequence/genomic_sequence/chromosomes.fasta).

Details

Annotation based on data provided by Yeast Genome project.

Source data built: Yeast Genome data are built at various time intervals. Sources used were downloaded Fri Nov 21 14:00:47 2003 Package built: Fri Nov 21 14:00:47 2003

References

http://www.yeastgenome.org/DownloadContents.shtml

Examples

data(yeastSEQCHR1)
nchar(yeastSEQCHR1)
Index

!=, BString, character-method  
   (XString-class), 86
!=, XString, XString-method  
   (XString-class), 86
!=, XString, XStringViews-method  
   (XStringViews-class), 96
!=, XStringViews, XString-method  
   (XStringViews-class), 96
!=, XStringViews, XStringViews-method  
   (XStringViews-class), 96
!=, character, BString-method  
   (XString-class), 86
!=, character, XStringViews-method  
   (XStringViews-class), 96

*Topic character
   stringDist, 76

*Topic classes
   AString-class, 1
   AlignedXStringSet-class, 2
   Biostrings internals, 7
   BOC_ScienceString-class, 7
   DNAString-class, 10
   InDel-class, 16
   MaskedXString-class, 24
   MIndex-class, 49
   PairwiseAlignedXStringSet-class, 54
   PDict-class, 60
   QualityScaledXStringSet-class, 67
   RNAString-class, 75
   XString-class, 86
   XStringPartialMatches-class, 88
   XStringQuality-class, 89
   XStringSet-class, 90
   XStringViews-class, 96

*Topic cluster
   stringDist, 76

*Topic datasets
   phiX174Phage, 64
   substitution.matrices, 77
   yeastSEQCHR1, 100

*Topic data
   AMINO_ACID_CODE, 5
   GENETIC_CODE, 13
   IUPAC_CODE_MAP, 18
   substitution.matrices, 77

*Topic internal
   Biostrings internals, 7

*Topic manip
   basecontent, 6
   chartr, 8
   complementSeq, 9
   gregexpr2, 15
   injectHardMask, 16
   letterFrequency, 19
   longestConsecutive, 23
   maskMotif, 26
   matchprobes, 42
   matchPWM, 43
   nucleotideFrequency, 51
   readFASTA, 68
   replaceLetterAt, 69
   reverseComplement, 71
   reverseSeq, 73
   subXString, 80
   translate, 82
   xscat, 85
   XStringSet-io, 94

*Topic methods
   AString-class, 1
   align-utils, 4
   AlignedXStringSet-class, 2
   Biostrings internals, 7
   BOC_ScienceString-class, 7
   chartr, 8
   DNAString-class, 10
   findPalindromes, 11
   InDel-class, 16
   letter, 22
   letterFrequency, 19
   MaskedXString-class, 24
   maskMotif, 26
INDEX

match-utils, 45
matchLRPatterns, 28
matchPattern, 30
matchPDict, 33
matchPDict-inexact, 38
matchProbePair, 41
matchPWM, 43
MIndex-class, 49
needwunsQS, 50
nucleotideFrequency, 51
PairwiseAlignedXStringSet-class, 54
pairwiseAlignment, 58
PDict-class, 60
pid, 65
pmatchPattern, 66
QualityScaledXStringSet-class, 67
reverseComplement, 71
RNAString-class, 75
subXString, 80
toComplex, 81
translate, 82
trimLRPatterns, 83
xscat, 85
XString-class, 86
XStringPartialMatches-class, 88
XStringQuality-class, 89
XStringSet-class, 90
XStringViews-class, 96
XStringViews-constructors, 99

*Topic models
needwunsQS, 50
pairwiseAlignment, 58

*Topic multivariate
stringDist, 76

*Topic utilities
AMINO_ACID_CODE, 5
GENETIC_CODE, 13
injectHardMask, 16
IUPAC_CODE_MAP, 18
matchPWM, 43
readFASTA, 68
replaceLetterAt, 69
substitution.matrices, 77
XStringSet-io, 94
.inplaceReplaceLetterAt
(replaceLetterAt), 69
==, BString, character-method
(XString-class), 86
==, XString, XString-method
(XString-class), 86
==, XString, XStringViews-method
(XStringViews-class), 96
==, XStringViews, XString-method
(XStringViews-class), 96
==, XStringViews, XStringViews-method
(XStringViews-class), 96
==, XStringViews, character-method
(XString-class), 96
==, character, BString-method
(XString-class), 86
==, character, XStringViews-method
(XStringViews-class), 96
[, ACTree-method(PDict-class), 60
[, AlignedXStringSet0-method
(AssociatedXStringSet-class). 2
[, PairwiseAlignedXStringSet-method
(PairwiseAlignedXStringSet-class). 54
[, QualityScaledXStringSet-method
(QualityScaledXStringSet-class). 67
[, XString-method(XString-class). 86
[, XStringPartialMatches-method
(XStringPartialMatches-class). 88
[, XStringSet-method
(XStringSet-class). 90
[<-, AlignedXStringSet0-method
(AssociatedXStringSet-class). 2
[<-, PairwiseAlignedXStringSet-method
(PairwiseAlignedXStringSet-class). 54
[, ByPos_MIndex-method
(MIndex-class). 49
[, PDict-method(PDict-class), 60
[, SparseList-method(Biostrings
internals), 7
[, XStringSet-method
(XStringSet-class). 90
[<-, XStringSet-method
(XStringSet-class). 90
%in%, XString, XStringSet-method
(XStringSet-class). 90
%in%, XStringSet, XStringSet-method
(XStringSet-class). 90
%in%, character, XStringSet-method
(XStringSet-class). 90
AA_ALPHABET, 14, 91
INDEX
AA_ALPHABET (AAString-class), 1
AAString, 1, 5, 11, 14, 75, 82, 86, 87, 90
AAString (AAString-class), 1
AAString-class, 1, 79, 83, 88, 93, 96
AAStringSet, 67
AAStringSet (XStringSet-class), 90
AAStringSet-class, 68
AAStringSet-class (XStringSet-class), 90
ACtree (PDict-class), 60
ACtree-class (PDict-class), 60
ACtree2 (PDict-class), 60
ACtree2-class (PDict-class), 60
adjacentViews (XStringViews-constructors), 99
agrep, 77
align-utils, 57
align-utils, 4, 47
aligned
(AlignedXStringSet-class), 2
aligned, AlignedXStringSet0-method
(AlignedXStringSet-class), 2
aligned, PairwiseAlignedFixedSubject-method
(PairwiseAlignedXStringSet-class), 54
AlignedXStringSet
(AlignedXStringSet-class), 2
AlignedXStringSet-class, 57
AlignedXStringSet-class, 2, 5
AlignedXStringSet0
(AlignedXStringSet-class), 2
AlignedXStringSet0-class
(AlignedXStringSet-class), 2
alignScore (needwunqS), 50
alphabet, 4, 21, 24, 53, 56
alphabet (XString-class), 86
alphabet, ANY-method
(XString-class), 86
alphabetFrequency, 2, 6, 8, 9, 11, 25, 31,
35, 53, 74, 75
alphabetFrequency (letterFrequency), 19
alphabetFrequency, DNAString-method
(letterFrequency), 19
alphabetFrequency, DNAStringSet-method
(letterFrequency), 19
alphabetFrequency, MaskedXString-method
(letterFrequency), 19
alphabetFrequency, RNAString-method
(letterFrequency), 19
alphabetFrequency, RNAStringSet-method
(letterFrequency), 19
alphabetFrequency, XString-method
(letterFrequency), 19
alphabetFrequency, XStringSet-method
(letterFrequency), 19
alphabetFrequency, XStringViews-method
(letterFrequency), 19
AMINO_ACID_CODE, 1, 2, 5, 14, 53
append, QualityScaledXStringSet, QualityScaledXStringSet-class
(QualityScaledXStringSet-class), 67
append, XStringSet, XStringSet-method
(XStringSet-class), 90
as.character, AlignedXStringSet0-method
(AlignedXStringSet-class), 2
as.character, MaskedXString-method
(MaskedXString-class), 24
as.character, PairwiseAlignedFixedSubject-method
(PairwiseAlignedXStringSet-class), 54
as.character, XString-method
(XString-class), 86
as.character, XStringSet-method
(XStringSet-class), 90
as.character, XStringViews-method
(XStringViews-class), 96
as.complex, DNAString-method
toComplex, 81
as.integer, PhredQuality-method
(XStringQuality-class), 89
as.integer, SolexaQuality-method
(XStringQuality-class), 89
as.list, MTB_PDict-method
(PDict-class), 60
as.list, SparseList-method
(Biostrings internals), 7
as.matrix, ACtree-method
(PDict-class), 60
as.matrix, PairwiseAlignedFixedSubject-method
(PairwiseAlignedXStringSet-class), 54
as.matrix, XStringSet-method
(XStringSet-class), 90
as.matrix, XStringViews-method
(XStringViews-class), 96
as.numeric, PhredQuality-method
INDEX

(XStringQuality-class), 89
as.numeric, SolexaQuality-method
(XStringQuality-class), 89
basecontent, 6, 23
Biostrings internals, 7
BLOSUM100
(substitution.matrices), 77
BLOSUM45 (substitution.matrices),
77
BLOSUM50 (substitution.matrices),
77
BLOSUM62 (substitution.matrices),
77
BLOSUM80 (substitution.matrices),
77
BOC2_SubjectString
(BOC_SubjectString-class),
7
BOC2_SubjectString-class
(BOC_SubjectString-class),
7
BOC_SubjectString
(BOC_SubjectString-class),
7
BOC_SubjectString-class, 7
BSgenome, 70, 71
BString, 1, 4, 10, 11, 20, 75, 89, 90
BString (XString-class), 86
BString-class, 93, 96
BString-class (XString-class), 86
BStringSet, 4, 67, 89
BStringSet (XStringSet-class), 90
BStringSet-class, 68, 90
BStringSet-class
(XStringSet-class), 90
BStringViews
(XStringViews-constructors),
99
BStringViews, ANY-method
(XStringViews-constructors),
99
BStringViews, file-method
(XStringViews-constructors),
99
BStringViews, XString-method
(XStringViews-constructors),
99
BStringViews, XStringViews-method
(XStringViews-constructors),
99
ByPos_MIndex-class
(MIndex-class), 49
cat, 68, 95
cDNA (translate), 82
CharacterToFASTAreCORDs
(XStringSet-class), 94
chartr, 8, 8, 17, 71, 72
chartr, ANY, ANY, MaskedXString-method
(chartr), 8
chartr, ANY, ANY, XString-method
(chartr), 8
chartr, ANY, ANY, XStringSet-method
(chartr), 8
chartr, ANY, ANY, XStringViews-method
(chartr), 8
class: AAString (AAString-class), 1
class: AAStringSet
(XStringSet-class), 90
class: ACtree (PDict-class), 60
class: ACtree2 (PDict-class), 60
class: AlignedXStringSet
(AlignedXStringSet-class),
2
class: AlignedXStringSet0
(AlignedXStringSet-class),
2
class: BOC2_SubjectString
(BOC_SubjectString-class),
7
class: BOC_SubjectString
(BOC_SubjectString-class),
7
class: BString (XString-class), 86
class: BStringSet
(XStringSet-class), 90
class: BStringSet-class
(XStringSet-class), 90
class: DNAString
(DNAIndex-class), 49
class: DNAStringSet
(XStringSet-class), 90
class: Dups (Biostrings
internals), 7
class: Indel (Indel-class), 16
class: MaskedAAString
(MaskedXString-class), 24
class: MaskedBString
(MaskedXString-class), 24
class: MaskedDNAString
(MaskedXString-class), 24
class: MaskedRNAString
(MaskedXString-class), 24
class: MaskedXString
(MaskedXString-class), 24
class:MIndex (MIndex-class), 49
class:MTB_PDict (PDict-class), 60
class:PairwiseAlignedFixedSubject (PairwiseAlignedXStringSet-class), 54
class:PairwiseAlignedFixedSubjectSummary (PairwiseAlignedXStringSet-class), 54
class:PairwiseAlignedXStringSet (PairwiseAlignedXStringSet-class), 54
class:PDict (PDict-class), 60
class:PDict3Parts (PDict-class), 60
class:PhredQuality (XStringQuality-class), 89
class:PreprocessedTB (PDict-class), 60
class:QualityAlignedXStringSet (AlignedXStringSet-class), 2
class:QualityScaledAAStringSet (QualityScaledXStringSet-class), 67
class:QualityScaledBStringSet (QualityScaledXStringSet-class), 67
class:QualityScaledDNAStringSet (QualityScaledXStringSet-class), 67
class:QualityScaledRNAStringSet (QualityScaledXStringSet-class), 67
class:RNAString (RNAString-class), 75
class:RNAStringSet (XStringSet-class), 90
class:SolexaQuality (XStringQuality-class), 89
class:SparseList (Biostrings internals), 7
class:TB_PDict (PDict-class), 60
class:Twobit (PDict-class), 60
class:XString (XString-class), 86
class:XStringCodec (Biostrings internals), 7
class:XStringPartialMatches (XStringPartialMatches-class), 88
class:XStringQuality (XStringQuality-class), 89
class:XStringSet (XStringSet-class), 90
class:XStringViews (XStringViews-class), 96
class:AAString (AAString-class), 24
class:AAStringSet (AAStringSet-class), 90
class:AAStringSetSeq (AAStringSet-codons), 90
class:AString (AString-class), 86
class:AStringSet (AStringSet-class), 90
class:Base (Base-class), 86
class:BaseSet (BaseSet-class), 90
class:Biostrings (Biostrings-class), 7
class:Bioprot (Bioprot-class), 10
class:BioprotSet (BioprotSet-class), 17
class:BString (BString-class), 86
class:BStringSet (BStringSet-class), 90
class:character (character-class), 86
class:characterSet (characterSet-class), 90
class:codons (codons-class), 82
class:codons, DNAString-method (translate), 82
class:codons, MaskedDNAString-method (translate), 82
class:codons, MaskedRNAString-method (translate), 82
class:codons, RNAString-method (translate), 82
class:coerce, AAString, MaskedAAString-method (translate), 82
class:coerce, BString, MaskedBString-method (translate), 82
class:coerce, BString, PhredQuality-method (translate), 82
class:coerce, BStringSet, PhredQuality-method (translate), 82
class:coerce, character, AAString-method (translate), 82
class:coerce, character, AAStringSet-method (translate), 82
class:coerce, character, BString-method (translate), 82
class:coerce, character, BStringSet-method (translate), 82
class:coerce, character, DNAString-method (translate), 82
class:coerce, character, DNAStringSet-method (translate), 82
class:coerce, character, PhredQuality-method (translate), 82
class:coerce, character, RNAString-method (translate), 82
class:coerce, character, RNAStringSet-method (translate), 82
class:coerce, character, XString-method (translate), 82
class:coerce, character, XStringSet-method (translate), 82
class:coerce, DNAString, MaskedDNAString-method (translate), 82
INDEX

107

coerce, integer, PhredQuality-method
(XStringQuality-class), 89

coerce, integer, SolexaQuality-method
(XStringQuality-class), 89

coerce, MaskedAAString, AAString-method
(MaskedXString-class), 24

coerce, MaskedBString, BString-method
(MaskedXString-class), 24

coerce, MaskedDNAString, DNAString-method
(MaskedXString-class), 24

coerce, MaskedRNAString, RNAString-method
(MaskedXString-class), 24

coerce, MaskedXString, MaskedIRanges-method
(MaskedXString-class), 24

coerce, MaskedXString, MaskedSet-method
(MaskedXString-class), 24

compareStrings
(align-utils), 4

complement
(reverseComplement), 71

complement, DNAString-method
(reverseComplement), 71

complement, DNAStringSet-method
(reverseComplement), 71

complement, MaskedDNAString-method
(reverseComplement), 71

complement, MaskedRNAString-method
(reverseComplement), 71

complement, RNAString-method
(reverseComplement), 71

coerce, XString, DNAString-method

coerce, XString, DNAStringSet-method

coerce, XString, DNAStringSet-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, RNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method

coerce, XString, DNAString-method
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>complement</code></td>
<td>Reverse complement</td>
<td>71</td>
</tr>
<tr>
<td><code>complementedPalindromeArmLength</code></td>
<td>Length of the arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeArmLength, DNAString-method</code></td>
<td>Length of the arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeArmLength, XStringViews-method</code></td>
<td>Length of the arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeLeftArm</code></td>
<td>Left arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeLeftArm, DNAString-method</code></td>
<td>Left arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeLeftArm, XStringViews-method</code></td>
<td>Left arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeRightArm</code></td>
<td>Right arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeRightArm, DNAString-method</code></td>
<td>Right arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>complementedPalindromeRightArm, XStringViews-method</code></td>
<td>Right arm of a palindrome</td>
<td>11</td>
</tr>
<tr>
<td><code>consmat</code></td>
<td>Consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusMatrix, ANY-method</code></td>
<td>ANY consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusMatrix, character-method</code></td>
<td>Character consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusMatrix, list-method</code></td>
<td>List consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusMatrix, matrix-method</code></td>
<td>Matrix consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusMatrix, PairwiseAlignedFixedSubject-method</code></td>
<td>Pairwise aligned fixed subject consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusMatrix, XStringSet-method</code></td>
<td>XStringSet consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusMatrix, XStringViews-method</code></td>
<td>XStringViews consensus matrix</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusString</code></td>
<td>Consensus string</td>
<td>56</td>
</tr>
<tr>
<td><code>consensusString, ANY-method</code></td>
<td>ANY consensus string</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusString, character-method</code></td>
<td>Character consensus string</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusString, list-method</code></td>
<td>List consensus string</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusString, matrix-method</code></td>
<td>Matrix consensus string</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusString, XStringSet-method</code></td>
<td>XStringSet consensus string</td>
<td>19</td>
</tr>
<tr>
<td><code>consensusString, XStringViews-method</code></td>
<td>XStringViews consensus string</td>
<td>19</td>
</tr>
<tr>
<td><code>countbases</code></td>
<td>Count bases</td>
<td>6</td>
</tr>
<tr>
<td><code>countIndex, MIndex-class</code></td>
<td>Count index</td>
<td>49</td>
</tr>
<tr>
<td><code>countIndex, ByPos_MIndex-method</code></td>
<td>Count index by position</td>
<td>49</td>
</tr>
<tr>
<td><code>countIndex, MaskedXString-method</code></td>
<td>Count index, masked XString</td>
<td>33</td>
</tr>
<tr>
<td><code>countIndex, MaskedXString-method</code></td>
<td>Count index, masked XString</td>
<td>33</td>
</tr>
<tr>
<td><code>countPDict, MaskedXString-method</code></td>
<td>Count PDict, masked XString</td>
<td>33</td>
</tr>
<tr>
<td><code>countPWM</code></td>
<td>Count PWM</td>
<td>43</td>
</tr>
<tr>
<td><code>coverage</code></td>
<td>Coverage</td>
<td>4, 21, 46, 47</td>
</tr>
<tr>
<td><code>deletion</code></td>
<td>Deletion</td>
<td>16</td>
</tr>
<tr>
<td><code>dinucleotideFrequency</code></td>
<td>Dinucleotide frequency</td>
<td>51</td>
</tr>
<tr>
<td><code>dist</code></td>
<td>Distance</td>
<td>77</td>
</tr>
<tr>
<td><code>dna2rna</code></td>
<td>DNA to RNA</td>
<td>82</td>
</tr>
<tr>
<td><code>DNA_ALPHABET</code></td>
<td>DNA alphabet</td>
<td>63, 91</td>
</tr>
<tr>
<td><code>DNA_ALPHABET, DNAString-class</code></td>
<td>DNA alphabet, DNAString class</td>
<td>10</td>
</tr>
</tbody>
</table>
DNA_BASES (DNAString-class), 10
DNAString, 1, 12, 14, 19, 29, 34, 41, 44, 46, 61, 62, 70–72, 75, 81, 82, 84, 86, 87, 90, 97
DNAString (DNAString-class), 10
DNAString-class, 10, 13, 35, 44, 72, 79, 83, 88, 90, 93, 96
DNAStringSet, 61, 62, 67, 70, 71, 72, 75, 81, 82, 84, 86, 87, 90, 97
DNAStringSet-class, 61, 62, 67, 70, 71, 82
DNAStringSet-class (XStringSet-class), 90
DNAStringSet-class (XStringSet-class), 90
DNAStringSet-class (XStringSet-class), 90
duplicated, Dups-method (Biostrings internals), 7
duplicated, Dups-method (PDict-class), 60
duplicated, PreprocessedTB-method (PDict-class), 60
duplicated, XStringSet-method (XStringSet-class), 90
Dups (Biostrings internals), 7
Dups-class (Biostrings internals), 7
end, AlignedXStringSet0-method (AlignedXStringSet-class), 2
e endIndex (MIndex-class), 49
endIndex, ByPos_MIndex-method (MIndex-class), 49
errorSubstitutionMatrices (substitution.matrices), 77
extractAllMatches (matchPDict), 33
fasta.info, 96
fasta.info (readFASTA), 68
FASTArecordsToBStringViews (XStringSet-ion), 94
FASTArecordsToCharacter (XStringSet-ion), 94
FASTArecordsToXStringViews (XStringSet-ion), 94
fastq.geometry (XStringSet-ion), 94
findComplementedPalindrome (findPalindromes), 11
findComplementedPalindrome (findPalindromes), 11
findComplementedPalindrome (findPalindromes), 11
findComplementedPalindrome (findPalindromes), 11
findComplementedPalindrome (findPalindromes), 11
findComplementedPalindrome (findPalindromes), 11
findPalindromes, MaskedXString-method (findPalindromes), 11
findPalindromes, XString-method (findPalindromes), 11
findPalindromes, XStringViews-method (findPalindromes), 11
gaps, 98
gaps, MaskedXString-method (MaskedXString-class), 24
GENETIC_CODE, 5, 13, 53, 82, 83
gregexpr, 15
gregexpr2, 15
hasLetterAt, 53
hasLetterAt (match-utils), 45
hasOnlyBaseLetters (letterFrequency), 19
hasOnlyBaseLetters, DNAString-method (letterFrequency), 19
hasOnlyBaseLetters, DNAStringSet-method (letterFrequency), 19
hasOnlyBaseLetters, MaskedDNAString-method (letterFrequency), 19
hasOnlyBaseLetters, MaskedRNAString-method (letterFrequency), 19
hasOnlyBaseLetters, RNAString-method (letterFrequency), 19
hasOnlyBaseLetters, RNAStringSet-method (letterFrequency), 19
hasOnlyBaseLetters, XStringViews-method (letterFrequency), 19
head, PDict3Parts-method (PDict-class), 60
head, TB_PDict-method (PDict-class), 60
InDel (InDel-class), 16
indel (AlignedXStringSet-class), 2
indel, AlignedXStringSet0-method (AlignedXStringSet-class), 2
InDel-class, 16
initialize, ACtree-method (PDict-class), 60
initialize, ACtree2-method
initialize, BOC2_SubjectString-method (BOC_SubjectString-class), 7
initialize, BOC2_SubjectString-method (BOC_SubjectString-class), 7
initialize, BOC2_SubjectString-method (BOC_SubjectString-class), 7
initialize, PreprocessedTB-method (PDict-class), 60
initialize, Twobit-method (PDict-class), 60
initialize, XStringCodec-method (Biostrings internals), 7
injectHardMask, 16, 25, 71
injectHardMask, MaskedXString-method (injectHardMask), 16
injectHardMask, XStringViews-method (injectHardMask), 16
injectSNPs, 70, 71
isMatching, 34, 35
isMatching (match-utils), 45
isMatchingAt, 30, 31
isMatchingEndingAt (match-utils), 45
isMatchingEndingAt, character-method (match-utils), 45
isMatchingEndingAt, XString-method (match-utils), 45
isMatchingEndingAt, XStringSet-method (match-utils), 45
isMatchingStartingAt (match-utils), 45
isMatchingStartingAt, character-method (match-utils), 45
isMatchingStartingAt, XString-method (match-utils), 45
isMatchingStartingAt, XStringSet-method (match-utils), 45
IUPAC_CODE_MAP, II, 18, 29, 46, 47, 71, 75, 84
lcprefix (pmatchPattern), 66
lcprefix, character, character-method (pmatchPattern), 66
lcprefix, character, XString-method (pmatchPattern), 66
lcprefix, XString, character-method (pmatchPattern), 66
lcprefix, XString, XString-method (pmatchPattern), 66
lcsubstr (pmatchPattern), 66
lcsubstr, character, character-method (pmatchPattern), 66
lcsubstr, character, XString-method (pmatchPattern), 66
lcsubstr, XString, character-method (pmatchPattern), 66
lcsubstr, XString, XString-method (pmatchPattern), 66
lcsubstr, XString, XString-method (pmatchPattern), 66
lcsubstr (pmatchPattern), 66
lcsubstr, character, character-method (pmatchPattern), 66
lcsubstr, character, XString-method (pmatchPattern), 66
lcsubstr, XString, character-method (pmatchPattern), 66
lcsubstr, XString, XString-method (pmatchPattern), 66
lcsubstr (pmatchPattern), 66
length, AlignedXStringSet0-method (AlignedXStringSet-class), 2
length, Dups-method (Biostrings internals), 7
length, MaskedXString-method (MaskedXString-class), 24
length, MIndex-method (MIndex-class), 49
length, PairwiseAlignedFixedSubjectSummary-method (PairwiseAlignedXStringSet-class), 54
length, PairwiseAlignedXStringSet-method (PairwiseAlignedXStringSet-class), 54
length, PDict-method (PDict-class), 60
length, PDict3Parts-method (PDict-class), 60
length, PreprocessedTB-method (PDict-class), 60
length, SparseList-method (Biostrings internals), 7
length, XString-method, 24
length, XString-method (XString-class), 86
length, XStringSet-method (XStringSet-class), 90
letter, 2, 11, 22, 75, 81, 88, 89, 98
letter, character-method (letter), 22
letter, MaskedXString-method (letter), 22
letter, XString-method (letter), 22
letter, XString-method (letter), 22
letter, XStringViews-method
(letter), 22
letterFrequency, 19
longestConsecutive, 23
ls, SparseList-method (Biostrings internals), 7
mask (maskMotif), 26
MaskCollection, 24
MaskCollection-class, 25, 26, 47
MaskedAAString, 17
MaskedAAString
(MaskedXString-class), 24
MaskedAAString-class
(MaskedXString-class), 24
MaskedBString, 17
MaskedBString
(MaskedXString-class), 24
MaskedBString-class
(MaskedXString-class), 24
MaskedDNAString, 17, 71, 82
MaskedDNAString
(MaskedXString-class), 24
MaskedDNAString-class
(MaskedXString-class), 24
maskedRatio, MaskedXString-method (MaskedXString-class), 24
MaskedDNAString, 17, 71, 82
MaskedRNAString
(MaskedXString-class), 24
MaskedRNAString-class
(MaskedXString-class), 24
MaskedXString, 8, 16, 17, 19, 20, 22, 24, 26, 30, 34, 51, 52, 71, 80
MaskedXString
(MaskedXString-class), 24
MaskedXString-class
(MaskedXString-class), 24
maskMotif, 8, 16, 17, 21, 22, 24, 26, 29, 47, 53, 72, 83
maskMotif, 13, 17, 25, 26, 31
maskMotif, MaskedXString, character-method (maskMotif), 26
maskMotif, MaskedXString, XString-method (maskMotif), 26
maskMotif, XString, ANY-method (maskMotif), 26
masks (MaskedXString-class), 24
masks, MaskedXString-method (MaskedXString-class), 24
masks, XString-method (MaskedXString-class), 24
masks<-, (MaskedXString-class), 24
masks<-, MaskedXString, MaskCollection-method (MaskedXString-class), 24
masks<-, MaskedXString, NULL-method (MaskedXString-class), 24
masks<-, XString, ANY-method (MaskedXString-class), 24
masks<-, XString, NULL-method (MaskedXString-class), 24
match, character, XStringSet-method (XStringSet-class), 90
match, XString, XStringSet-method (XStringSet-class), 90
match, XStringSet, XStringSet-method (XStringSet-class), 90
match-utils, 5, 45, 65, 84
matchDNAPattern (matchPattern), 30
matchLRPatterns, 13, 28, 31, 42, 47, 84
matchLRPatterns, MaskedXString-method (matchLRPatterns), 28
matchLRPatterns, XString-method (matchLRPatterns), 28
matchLRPatterns, XStringViews-method (matchLRPatterns), 28
matchPattern, 8, 13, 15, 29, 30, 35, 41–44, 47, 59, 66, 84
matchPattern, BOC2_SubjectString-method (BOC_SubjectString-class), 7
matchPattern, BOC_SubjectString-method (BOC_SubjectString-class), 7
matchPattern, character-method (matchPattern), 30
matchPattern, MaskedXString-method (matchPattern), 30
matchPattern, XString-method (matchPattern), 30
matchPattern, XStringSet-method (matchPattern), 30
matchPattern, XStringViews-method (matchPattern), 30
matchPDict, 31, 33, 38, 39, 42, 43, 47, 49, 50, 59, 60, 63
matchPDict, MaskedXString-method (matchPDict), 33
matchPDict, XString-method (matchPDict), 33
matchPDict, XStringSet-method (matchPDict), 33
matchPDict, XStringViews-method (matchPDict), 33
matchPDict-inexact, 33, 34
<table>
<thead>
<tr>
<th>Function/Subroutine</th>
<th>Line(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>matchPDict-exact (matchPDict)</td>
<td>33</td>
</tr>
<tr>
<td>matchPDict-inexact</td>
<td>35, 38</td>
</tr>
<tr>
<td>matchProbePair, 13, 29, 31, 41</td>
<td></td>
</tr>
<tr>
<td>matchProbePair,DNAString-method (matchProbePair)</td>
<td>41</td>
</tr>
<tr>
<td>matchProbePair,MaskedDNAString-method (matchProbePair)</td>
<td>41</td>
</tr>
<tr>
<td>matchProbePair,XStringViews-method (matchProbePair)</td>
<td>41</td>
</tr>
<tr>
<td>matchprobes</td>
<td>42</td>
</tr>
<tr>
<td>matchPWM</td>
<td>43</td>
</tr>
<tr>
<td>maxScore (matchPWM)</td>
<td>43</td>
</tr>
<tr>
<td>maxWeights (matchPWM)</td>
<td></td>
</tr>
<tr>
<td>mergeIUPACLetters (IUPAC_CODE_MAP)</td>
<td>18</td>
</tr>
<tr>
<td>MIndex, 31, 34, 36, 47</td>
<td></td>
</tr>
<tr>
<td>MIndex (MIndex-class)</td>
<td>49</td>
</tr>
<tr>
<td>MIndex-class, 31, 35, 39, 47, 49, 98</td>
<td></td>
</tr>
<tr>
<td>mismatch</td>
<td>31</td>
</tr>
<tr>
<td>mismatch (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>mismatch,AlignedXStringSet0,missing-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatch,ANY,XStringViews-method (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>mismatchSummary (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchSummary,AlignedXStringSet0-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchSummary,PairwiseAlignedFixedSubjectSummary-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchSummary,PairwiseAlignedFixedSubjectSummary-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchSummary,QualityAlignedXStringSet-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchTable (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchTable,AlignedXStringSet0-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchTable,PairwiseAlignedXStringSet-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mismatchTable,QualityAlignedXStringSet-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>mkAllStrings</td>
<td>51</td>
</tr>
<tr>
<td>(nucleotideFrequency)</td>
<td></td>
</tr>
<tr>
<td>MTB_PDict (PDict-class)</td>
<td>60</td>
</tr>
<tr>
<td>MTB_PDict-class (PDict-class)</td>
<td>60</td>
</tr>
<tr>
<td>names,MIndex-method</td>
<td>60</td>
</tr>
<tr>
<td>names,PDict-method (PDict-class)</td>
<td>49</td>
</tr>
<tr>
<td>names,XStringSet-method</td>
<td>90</td>
</tr>
<tr>
<td>names&lt;-,MIndex-method</td>
<td>49</td>
</tr>
<tr>
<td>names&lt;-,PDict-method (PDict-class)</td>
<td>60</td>
</tr>
<tr>
<td>names&lt;-,XStringSet-method (XStringSet-class)</td>
<td>90</td>
</tr>
<tr>
<td>narrow, 91, 93</td>
<td></td>
</tr>
<tr>
<td>narrow,Character-method</td>
<td></td>
</tr>
<tr>
<td>narrow,MaskedXString-method</td>
<td>24</td>
</tr>
<tr>
<td>nchar,AlignedXStringSet0-method (AlignedXStringSet-class)</td>
<td>54</td>
</tr>
<tr>
<td>nchar,MaskedXString-method</td>
<td></td>
</tr>
<tr>
<td>nchar,PairwiseAlignedFixedSubjectSummary-method (PairwiseAlignedXStringSet-class)</td>
<td>54</td>
</tr>
<tr>
<td>nchar,PairwiseAlignedXString-set-method (PairwiseAlignedXStringSet-class)</td>
<td>54</td>
</tr>
<tr>
<td>nchar,XString-method</td>
<td>86</td>
</tr>
<tr>
<td>nchar,XString-set-method</td>
<td></td>
</tr>
<tr>
<td>nchar,QualityScaledXStringSet-method (QualityScaledXStringSet-class)</td>
<td>67</td>
</tr>
<tr>
<td>nedit (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>nedit,AlignedXStringSet0-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>nedit,PairwiseAlignedFixedSubjectSummary-method (align-utils)</td>
<td>4</td>
</tr>
<tr>
<td>neditAt (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>neditEndingAt (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>neditEndingAt,character-method (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>neditEndingAt,XString-method (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>neditStartingAt,character-method (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>neditStartingAt,XString-method (match-utils)</td>
<td>45</td>
</tr>
<tr>
<td>needwunsQS, 50</td>
<td></td>
</tr>
<tr>
<td>needwunsQS,Character-method</td>
<td>50</td>
</tr>
<tr>
<td>needwunsQS,Character-method (needwunsQS)</td>
<td>50</td>
</tr>
</tbody>
</table>
needwunsQS, character, XString-method (needwunsQS), 50

needwunsQS, XString, character-method (needwunsQS), 50

needwunsQS, XString, XString-method (needwunsQS), 50

nindel (AlignedXStringSet-class), 2

nindel, AlignedXStringSet0-method (AlignedXStringSet-class), 2

nindel, PairwiseAlignedFixedSubjectSummary-method (PairwiseAlignedFixedSubjectSummary-class), 54

nindel, PairwiseAlignedFixedSubject-method (PairwiseAlignedFixedSubject-class), 54

nmatch (match-utils), 45

nmatch, ANY, XStringViews-method (match-utils), 45

nmatch, PairwiseAlignedFixedSubjectSummary, missing-method (align-utils), 4

nmismatch (match-utils), 45

nmismatch, AlignedXStringSet0, missing-method (align-utils), 4

nmismatch, ANY, XStringViews-method (match-utils), 45

nmismatch, PairwiseAlignedFixedSubjectSummary, missing-method (align-utils), 4

nmismatch, PairwiseAlignedXStringSet, missing-method (align-utils), 4

nmismatchEndingAt (match-utils), 45

nucleotideFrequencyAt (nucleotideFrequency), 51

nucleotideFrequencyAt (nucleotideFrequency), 47

nucleotideFrequencyAt (nucleotideFrequency), 51

nucleotideFrequencyAt, XString-method (nucleotideFrequency), 51

nucleotideFrequencyAt, XString-views-method (nucleotideFrequency), 51

nucleotideSubstitutionMatrix (substitution.matrices), 77

oligonucleotideFrequency, XString-method (nucleotideFrequency), 51

oligonucleotideFrequency, XString-set-method (nucleotideFrequency), 51

oligonucleotideFrequency, XStringViews-method (nucleotideFrequency), 51

oligonucleotideTransitions (nucleotideFrequency), 51

order, XString-set-method (XStringSet-class), 90

PairwiseAlignedFixedSubject, 59

PairwiseAlignedFixedSubject (PairwiseAlignedFixedSubject-class), 54

PairwiseAlignedFixedSubject, character, character-method (PairwiseAlignedFixedSubject-class), 54

PairwiseAlignedFixedSubject, character, missing-method (PairwiseAlignedFixedSubject-class), 54

PairwiseAlignedFixedSubject, XString, XString-method (PairwiseAlignedFixedSubject-class), 54

PairwiseAlignedFixedSubject-class (PairwiseAlignedFixedSubject-class), 54

PairwiseAlignedFixedSubjectSummary (PairwiseAlignedFixedSubjectSummary-class), 54

PairwiseAlignedFixedSubjectSummary, character, character-method (PairwiseAlignedFixedSubjectSummary-class), 54

PairwiseAlignedFixedSubjectSummary, character, missing-method (PairwiseAlignedFixedSubjectSummary-class), 54

PairwiseAlignedFixedSubjectSummary-class (PairwiseAlignedFixedSubjectSummary-class), 54

PairwiseAlignedXStringSet, 21

PairwiseAlignedXStringSet (PairwiseAlignedXStringSet-class), 54

PairwiseAlignedXStringSet, character, character-method (PairwiseAlignedXStringSet-class), 54

PairwiseAlignedXStringSet, character, missing-method (PairwiseAlignedXStringSet-class), 54

PairwiseAlignedXStringSet-class (PairwiseAlignedXStringSet-class), 54

PairwiseAlignedXStringSet, XString, XString-method (PairwiseAlignedXStringSet-class), 54

PairwiseAlignedXStringSet, XStringSet, missing-method (PairwiseAlignedXStringSet-class), 54
INDEX

PairwiseAlignedXStringSet-class, palindromeRightArm, XString-method
  3, 16
PairwiseAlignedXStringSet-class, palindromeRightArm, XStringViews-method
  5, 50, 54, 59, 65, 79, 90
pairwiseAlignment, 3, 5, 16, 31, 50, 57, 58, 65, 79, 90
pairwiseAlignment, character, character-method (pairwiseAlignment), 58
pairwiseAlignment, character, QualityScaledXString-set-method (pairwiseAlignment), 58
pairwiseAlignment, character, XString-method (pairwiseAlignment), 58
pairwiseAlignment, XString-set-method (pairwiseAlignment), 58
pairwiseAlignment, XString, character-method (pairwiseAlignment), 58
pairwiseAlignment, XString, QualityScaledXString-set-method (pairwiseAlignment), 58
pairwiseAlignment, XString, XString-set-method (pairwiseAlignment), 58
pairwiseAlignment, XString, XStringViews-method (pairwiseAlignment), 58
pairwiseAlignment, XStringSet, character-method (pairwiseAlignment), 58
pairwiseAlignment, XStringSet, QualityScaledXString-set-method (pairwiseAlignment), 58
pairwiseAlignment, XStringSet, XString-set-method (pairwiseAlignment), 58
pairwiseAlignment, XStringSet, XStringViews-set-method (pairwiseAlignment), 58
palindromeArmLength
  (findPalindromes), 11
palindromeArmLength, XString-method (findPalindromes), 11
palindromeArmLength, XStringViews-method (findPalindromes), 11
palindromeLeftArm
  (findPalindromes), 11
palindromeLeftArm, XString-method (findPalindromes), 11
palindromeLeftArm, XStringViews-method (findPalindromes), 11
palindromeRightArm
  (findPalindromes), 11
palindromeRightArm, XString-method (findPalindromes), 11
palindromeRightArm, XStringViews-method (findPalindromes), 11
pairwiseAlignment, character, character-method (pairwiseAlignment), 58
pairwiseAlignment, character, QualityScaledXString-set-method (pairwiseAlignment), 58
pairwiseAlignment, character, XString-method (pairwiseAlignment), 58
PAM120
  (substitution.matrices), 77
PAM250
  (substitution.matrices), 77
PAM30
  (substitution.matrices), 77
PAM40
  (substitution.matrices), 77
PAM70
  (substitution.matrices), 77
paste, 86
pattern, PairwiseAlignedXStringSet-method (pairwiseAlignment), 58
pattern, QualityScaledXStringSet-method (pairwiseAlignment), 58
pattern, XString-method (pairwiseAlignment), 58
patternFrequency (PDict-class), 60
patternFrequency, PDict-method (PDict-class), 60
PDict, 34, 38, 39
PDict (PDict-class), 60
PDict, character-method (PDict-class), 60
PDict, DNAString-set-method (PDict-class), 60
PDict, DNAStringViews-method (PDict-class), 60
PDict, XString-set-method (PDict-class), 60
PDict, XStringViews-set-method (PDict-class), 60
PDict, DNAString-set-method (PDict-class), 60
PDict, DNAStringViews-method (PDict-class), 60
PDict, XString-set-method (PDict-class), 60
PDict, XStringViews-set-method (PDict-class), 60
PhredQuality
  (XStringQuality-class), 89
PhredQuality-class, 79
PhredQuality-class (XStringQuality-class), 89
pid, 57, 65
pid, PairwiseAlignedXStringSet-method (pid), 65
pmatchPattern, 66
pmatchPattern, character-method (pmatchPattern), 66
pmatchPattern, XString-method (pmatchPattern), 66
pmatchPattern, XStringViews-method (pmatchPattern), 66
INDEX

PreprocessedTB (PDict-class), 60
PreprocessedTB-class (PDict-class), 60
print.needwunsQS (needwunsQS), 50
PWMscore (matchPWM), 43
PWMscoreStartingAt (matchPWM), 43
quality (QualityScaledXStringSet-class), 67
quality, QualityScaledXStringSet-method (QualityScaledXStringSet-class), 67
QualityAlignedXStringSet (AlignedXStringSet-class), 2
QualityAlignedXStringSet-class (AlignedXStringSet-class), 2
QualityScaledAAStringSet (QualityScaledXStringSet-class), 67
QualityScaledAAStringSet-class (QualityScaledXStringSet-class), 67
QualityScaledBStringSet (QualityScaledXStringSet-class), 67
QualityScaledBStringSet-class (QualityScaledXStringSet-class), 67
QualityScaledDNAStringSet (QualityScaledXStringSet-class), 67
QualityScaledDNAStringSet-class (QualityScaledXStringSet-class), 67
QualityScaledRNAStringSet (QualityScaledXStringSet-class), 67
QualityScaledRNAStringSet-class (QualityScaledXStringSet-class), 67
QualityScaledXStringSet, 58
QualityScaledXStringSet (QualityScaledXStringSet-class), 67
QualityScaledXStringSet-class, 67
qualitySubstitutionMatrices (substitution.matrices), 77
quPhiX174 (phiX174Phage), 64
rank, XStringSet-method

(XStringSet-class), 90
read.AAStringSet, 69
read.AAStringSet (XStringSet-io), 94
read.BStringSet, 69
read.BStringSet (XStringSet-io), 94
read.BStringViews (XStringSet-io), 94
read.DNAStringSet, 69
read.DNAStringSet (XStringSet-io), 94
read.Mask, 26
read.RNAStringSet, 69
read.RNAStringSet (XStringSet-io), 94
read.table, 69
read.XStringViews (XStringSet-io), 94
readFasta, 68, 95, 96
reduce, MaskedXString-method (MaskedXString-class), 24
rep, AlignedXStringSet0-method (AlignedXStringSet-class), 2
rep, PairwiseAlignedXStringSet-method (PairwiseAlignedXStringSet-class), 54
rep, XStringSet-method (XStringSet-class), 90
replaceLetterAt, 8, 17, 69
replaceLetterAt, DNAString-method (replaceLetterAt), 69
replaceLetterAt, DNAStringSet-method (replaceLetterAt), 69
replaceLetterAtLoc (replaceLetterAt), 69
rev, 53
revcompDNA (reverseSeq), 73
revcompRNA (reverseSeq), 73
reverse, character-method (reverseComplement), 71
reverse, MaskedXString-method, 25
reverse, MaskedXString-method (reverseComplement), 71
reverse, XString-method, 53, 73, 88
reverse, XString-method (reverseComplement), 71
reverse, XStringSet-method (reverseComplement), 71
reverse, XStringViews-method (reverseComplement), 71
reverseComplement, 6, 8, 9, 11, 29, 42, 44, 71, 73–75, 83
reverseComplement, DNAString-method (reverseComplement), 71
reverseComplement, DNAStringSet-method (reverseComplement), 71
reverseComplement, MaskedDNAString-method (reverseComplement), 71
reverseComplement, MaskedRNAString-method (reverseComplement), 71
reverseComplement, matrix-method (matchPWM), 43
reverseComplement, RNAString-method (reverseComplement), 71
reverseComplement, RNAStringSet-method (reverseComplement), 71
reverseComplement, XStringViews-method (reverseComplement), 71
reverseSeq, 23, 73
Rle, 47, 70
rna2dna (translate), 82
RNA_ALPHABET, 91
RNA_ALPHABET (RNAString-class), 75
RNA_BASES (RNAString-class), 75
RNA_GENETIC_CODE (GENETIC_CODE), 13
RNAString, 1, 11, 14, 19, 29, 46, 71, 72, 82, 84, 86, 87, 90, 97
RNAString (RNAString-class), 75
RNAString-class, 11, 72, 75, 83, 87, 88, 93, 96
RNAStringSet, 67, 71, 82
RNAStringSet (XStringSet-class), 90
RNAStringSet-class, 68, 72
RNAStringSet-class (XStringSet-class), 90
scan, 69
score, PairwiseAlignedFixedSubjectSummary-method (PairwiseAlignedXStringSet-class), 54
score, PairwiseAlignedXStringSet-method (PairwiseAlignedXStringSet-class), 54
setdiff, XStringSet, XStringSet-method (XStringSet-class), 90
setequal, XStringSet, XStringSet-method (XStringSet-class), 90
show, ACTree-method (PDict-class), 60
show, ACTree2-method (PDict-class), 60
show, AlignedXStringSet0-method (AlignedXStringSet-class), 2
show, ByPos_MIndex-method (MIndex-class), 49
show, Dups-method (Biostrings internals), 7
show, MaskedXString-method (MaskedXString-class), 24
show, MTB_PDict-method (PDict-class), 60
show, PairwiseAlignedFixedSubjectSummary-method (PairwiseAlignedXStringSet-class), 54
show, PairwiseAlignedXStringSet-method (PairwiseAlignedXStringSet-class), 54
show, QualityScaledXStringSet-method (QualityScaledXStringSet-class), 67
show, TB_PDict-method (PDict-class), 60
show, Twobit-method (PDict-class), 60
show, XString-method (XString-class), 86
show, XStringPartialMatches-method (XStringPartialMatches-class), 88
show, XStringSet-method (XStringSet-class), 90
show, XStringViews-method (XStringViews-class), 96
SolexaQuality (XStringQuality-class), 89
SolexaQuality-class, 79
SolexaQuality-class (XStringQuality-class), 89
sort, XStringSet-method (XStringSet-class), 90
SparseList (Biostrings internals), 7
SparseList-class (Biostrings internals), 7
srPhiX174 (phiX174Phage), 64
start, AlignedXStringSet0-method (AlignedXStringSet-class), 2
startIndex (MIndex-class), 49
startIndex, ByPos_MIndex-method (MIndex-class), 49
stringDist, 59, 76
INDEX

stringDist, character-method (stringDist), 76
stringDist, QualityScaledXStringSet-method (stringDist), 76
stringDist, XStringSet-method (stringDist), 76
strrev (reverseComplement), 71
subBString (subXString), 80
subject, PairwiseAlignedXStringSet-method (PairwiseAlignedXStringSet-class), 54
subpatterns (XStringPartialMatches-class), 88
subpatterns, XStringPartialMatches-method (XStringSet-class), 90
subseq, character-method (XStringSet-class), 90
subseq, MaskedXString-method (MaskedXString-class), 24
subseq, XStringSet-method (XStringSet-class), 90
subseq<-, character-method (XStringSet-class), 90
subseq<-, XStringSet-method (XStringSet-class), 90
substitution.matrices, 50, 59, 77, 77
substr, 91–93
substr, MaskedXString-method (subXString), 80
substr, XString-method (subXString), 80
substring, MaskedXString-method (subXString), 80
substring, XString-method (subXString), 80
subXString, 80
summary, PairwiseAlignedFixedSubject-method (PairwiseAlignedXStringSet-class), 54
tail, PDict3Parts-method (PDict-class), 60
tail, TB_PDict-method (PDict-class), 60
tb (PDict-class), 60
tb, PreprocessedTB-method (PDict-class), 60
tb_PDict (PDict-class), 60
TB_PDict-class (PDict-class), 60
toComplex, 81
toComplex, DNAString-method (toComplex), 81
toString, AlignedXStringSet-method (AlignedXStringSet-class), 2
toString, MaskedXString-method (MaskedXString-class), 24
toString, PairwiseAlignedFixedSubject-method (PairwiseAlignedXStringSet-class), 54
toString, XString-method (XString-set-class), 86
toString, XStringSet-method (XStringSet-class), 90
toString, XStringViews-method (XStringViews-class), 96
toupper, 43
transcribe (translate), 82
translate, 14, 82
translate, DNAString-method (translate), 82
translate, DNAStringSet-method (translate), 82
translate, MaskedDNAString-method (translate), 82
translate, MaskedRNAString-method (translate), 82
translate, RNAString-method (translate), 82
translate, RNAStringSet-method (translate), 82
trimLRPatterns, 29, 47, 83
width,MIndex-method
(MIndex-class), 49
width,PDict-method
(PDict-class),
60
width,PDict3Parts-method
(PDict-class),
60
width,PreprocessedTB-method
(PDict-class),
60
width,XStringSet-method
(XStringSet-class),
90
write.BStringViews
(XStringSet-io), 94
write.table,
69
write.XStringSet,
69
write.XStringSet
(XStringSet-io),
94
write.XStringViews
(XStringSet-io), 94
writeFASTA,
96
writeFASTA
(readFASTA), 68
wtPhiX174 (phiX174Phage),
64
xsbasetype (Biostrings internals), 7
xsbasetype,AAString-method
(XString-class), 86
xsbasetype,AlignedXStringSet0-method
(AlignedXStringSet-class),
2
xsbasetype,BString-method
(XString-class), 86
xsbasetype,DNAString-method
(XString-class), 86
xsbasetype,MaskedXString-method
(MaskedXString-class), 24
xsbasetype,PairwiseAlignedXStringSet-method
(PairwiseAlignedXStringSet-class),
54
xsbasetype,RNAString-method
(XString-class), 86
xsbasetype,XStringSet-method
(XStringSet-class),
90
xsbasetype,XStringViews-method
(XStringViews-class),
96
xsbasetype<-,(Biostrings internals), 7
xsbasetype<-,MaskedXString-method
(MaskedXString-class), 24
xsbasetype<-,XString-method
(XString-class), 86
xsbasetype<-,XStringSet-method
(XStringSet-class),
90
xsbasetype<-,XStringViews-method
(XStringViews-class), 96
xscat, 85
XSequence, 92
XString, 1, 8, 10, 12, 1, 17, 19, 20, 22, 24, 28,
30, 34, 46, 47, 49–52, 55, 58, 66, 67,
70–72, 75, 80, 81, 83–85, 91–93, 96,
97, 99
XString (XString-class), 86
XString-class, 37
XString-class, 2, 5, 8, 11, 21, 22, 25, 26,
29, 47, 53, 66, 75, 81, 84, 86, 86, 89,
98, 100
XStringCodec (Biostrings internals), 7
XStringCodec-class (Biostrings internals), 7
XStringPartialMatches-class, 88
XStringQuality, 58, 67, 76
XStringQuality
(XStringQuality-class), 89
XStringQuality-class, 59, 68, 89
XStringSet, 8, 19, 20, 30, 34, 46, 47, 51,
52, 55, 58, 67, 71, 76, 83–85, 87,
94–96
XStringSet (XStringSet-class), 90
XStringSet-class, 3
XStringSet-class, 5, 8, 21, 53, 83, 84,
86, 88, 90, 96, 98
XStringSet-io, 94
XStringSetToFASTArecords
(XStringSet-io), 94
XStringViews, 4, 8, 12, 13, 16, 17, 19, 20,
22, 25, 26, 28–31, 41, 42, 44, 46, 49,
51, 52, 61, 67, 71, 80, 82, 85, 91, 92,
94–96, 99
xsbasetype<-,XStringViews-method
(XStringViews-class), 99
XStringViews, ANY-method
(XStringViews-constructors),
99
XStringViews, XString-method
(XStringViews-constructors),
99
XStringViews, XStringViews-method
(XStringViews-constructors),
99
XStringViews-class, 57
XStringViews-class, 5, 8, 13, 17, 21,
22, 25, 26, 29, 31, 35, 44, 47, 50, 53,
63, 66, 72, 81, 83, 86, 88, 89, 93, 96,
96, 100
XStringViews-constructors, 98, 99

yeastSEQCHR1, 100