

# Package ‘bioseq’

May 7, 2026

**Type** Package

**Title** A Toolbox for Manipulating Biological Sequences

**Version** 0.1.5

## Description

Classes and functions to work with biological sequences (DNA, RNA and amino acid sequences). Implements S3 infrastructure to work with biological sequences as described in Keck (2020) <[doi:10.1111/2041-210X.13490](https://doi.org/10.1111/2041-210X.13490)>. Provides a collection of functions to perform biological conversion among classes (transcription, translation) and basic operations on sequences (detection, selection and replacement based on positions or patterns). The package also provides functions to import and export sequences from and to other package formats.

**License** GPL-3

**URL** <https://fkeck.github.io/bioseq/>

**BugReports** <https://github.com/fkeck/bioseq/issues>

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 3.1.0)

**Imports** methods, vctrs, tibble, ape, crayon, dplyr, pillar, stringi, stringr, stringdist, readr, rlang

**Suggests** knitr, rmarkdown, testthat (>= 2.1.0), covr

**VignetteBuilder** knitr

**RoxygenNote** 7.2.1

**NeedsCompilation** no

**Author** Francois Keck [aut, cre, cph] (ORCID: <<https://orcid.org/0000-0002-3323-4167>>)

**Maintainer** Francois Keck <[francois.keck@gmail.com](mailto:francois.keck@gmail.com)>

**Repository** CRAN

**Date/Publication** 2025-09-29 16:20:08 UTC

## Contents

aa	3
aliview	4
alphabets	4
as-tibble-ape	5
as-tibble-bioseq	6
as_aa	7
as_AAbin	7
as_AAbin.tbl_df	8
as_dna	8
as_DNAbin	9
as_DNAbin.tbl_df	9
as_rna	10
as_seqinr_alignment	10
dic_genetic_codes	11
dna	11
fragilaria	12
genetic-codes	12
is_aa	13
is_dna	14
is_rna	15
new_aa	15
new_dna	16
new_rna	16
read_fasta	16
rev_complement	17
rna	18
seaview	18
seq-replace	19
seq_cluster	20
seq_combine	21
seq_consensus	22
seq_count_pattern	23
seq_crop_pattern	25
seq_crop_position	26
seq_detect_pattern	27
seq_disambiguate_IUPAC	29
seq_extract_pattern	29
seq_extract_position	31
seq_nchar	31
seq_nseq	32
seq_remove_pattern	33
seq_remove_position	34
seq_replace_position	35
seq_rev_translate	35
seq_spellout	36
seq_split_kmer	37

seq_split_pattern . . . . .	38
seq_stat_gc . . . . .	39
seq_stat_prop . . . . .	40
seq_translate . . . . .	40
transcription . . . . .	41
write_fasta . . . . .	42
<b>Index</b>	<b>43</b>

---

aa *Build an amino acid (AA) vector*

---

## Description

aa() build a AA vector from a character vector.

## Usage

```
aa(...)
```

## Arguments

... character to turn into AA. Can be a set of name-value pairs.

## Value

vector of class bioseq\_aa

## See Also

Other classes: [dna\(\)](#), [rna\(\)](#)

## Examples

```
aa("AGGTGC", "TTCGA")  
  
aa(Seq_1 = "AGGTGC", Seq_2 = "TTCGA")  
  
x <- c("AGGTGC", "TTCGA")  
aa(x)
```

---

aliview	<i>AliView: DNA sequences viewer</i>
---------	--------------------------------------

---

### Description

This function uses AliView (Larsson, 2014) to visualize DNA sequences. The software must be installed on the computer.

### Usage

```
aliview(  
  x,  
  aliview_exec = getOption("bioseq.aliview.exec", default = "aliview")  
)
```

### Arguments

`x` a DNA, RNA or AA vector. Alternatively a DNABin or AABin object.  
`aliview_exec` a character string giving the path of the program.

### Details

By default, the function assumes that the executable is installed in a directory located on the PATH. Alternatively the user can provide an absolute path to the executable (i.e. the location where the software was installed/uncompressed). This information can be stored in the global options settings using `options(bioseq.aliview.exec = "my_path_to_aliview")`.

### References

Larsson, A. (2014). AliView: a fast and lightweight alignment viewer and editor for large data sets. *Bioinformatics* 30(22): 3276-3278.

### See Also

Other GUI wrappers: [seaview\(\)](#)

---

alphabets	<i>Biological alphabets</i>
-----------	-----------------------------

---

### Description

List of the allowed characters for each type of sequences.

### DNA

A C G T W S M K R Y B D H V N -

**RNA**

A C G U W S M K R Y B D H V N -

**AA**

A C D E F G H I K L M N P Q R S T V W Y B X Z J U O \* -

**References**

Nomenclature Committee of the International Union of Biochemistry (NC-IUB) (1986). Proc. Natl. Acad. Sci. USA. 83 (1): 4–8.

Nomenclature and Symbolism for Amino Acids and Peptides. IUPAC-IUB Joint Commission on Biochemical Nomenclature. 1983.

---

as-tibble-ape	<i>Convert DNABin/AABin to tibble</i>
---------------	---------------------------------------

---

**Description**

These methods convert sequences from **ape** formats DNABin and AABin to tibbles.

**Usage**

```
as_tibble.DNABin(x, label = "label", sequence = "sequence", ...)
```

```
as_tibble.AABin(x, label = "label", sequence = "sequence", ...)
```

**Arguments**

x	a DNABin or AABin object.
label	Name of the column that stores the sequence labels in the returned tibble.
sequence	Name of the column that stores the sequences in the returned tibble.
...	Not used.

**Value**

A tibble with two columns (if name is not NULL, the default) or one column (otherwise).

**See Also**

Other conversions: [as-tibble-bioseq](#), [as\\_AABin\(\)](#), [as\\_DNABin\(\)](#), [as\\_aa\(\)](#), [as\\_dna\(\)](#), [as\\_rna\(\)](#), [as\\_seqinr\\_alignment\(\)](#)

## Examples

```
require(ape)
require(tibble)
x <- rDNABin(nrow = 10, ncol = 25)
as_tibble.DNABin(x)
```

---

as-tibble-bioseq      *Convert bioseq DNA, RNA and AA to tibble*

---

## Description

Convert bioseq DNA, RNA and AA to tibble

## Usage

```
as_tibble.bioseq_dna(x, label = "label", sequence = "sequence", ...)
as_tibble.bioseq_rna(x, label = "label", sequence = "sequence", ...)
as_tibble.bioseq_aa(x, label = "label", sequence = "sequence", ...)
```

## Arguments

x	a DNA, RNA or AA vector.
label	Name of the column that stores the sequence labels in the returned tibble.
sequence	Name of the column that stores the sequences in the returned tibble.
...	Not used.

## Value

A tibble with two columns (if name is not NULL, the default) or one column (otherwise).

## See Also

Other conversions: [as-tibble-ape](#), [as\\_AABin\(\)](#), [as\\_DNABin\(\)](#), [as\\_aa\(\)](#), [as\\_dna\(\)](#), [as\\_rna\(\)](#), [as\\_seqinr\\_alignment\(\)](#)

## Examples

```
require(tibble)
x <- dna(A = "ACGTTAGTGTAGCCGT", B = "CTCGAAATGA", C = NA)
as_tibble(x)
```

---

as_aa	<i>Coercion to an amino acid (AA) vector</i>
-------	--

---

**Description**

Coercion to an amino acid (AA) vector

**Usage**

```
as_aa(x)
```

**Arguments**

x                    An object to coerce.

**Value**

An amino acid vector of class `bioseq_aa`

**See Also**

Other conversions: [as-tibble-ape](#), [as-tibble-bioseq](#), [as\\_AAbin\(\)](#), [as\\_DNAbin\(\)](#), [as\\_dna\(\)](#), [as\\_rna\(\)](#), [as\\_seqinr\\_alignment\(\)](#)

---

as_AAbin	<i>Coerce to AAbin</i>
----------	------------------------

---

**Description**

Coerce to AAbin

**Usage**

```
as_AAbin(x, ...)
```

**Arguments**

x                    An object.  
...                  Other parameters.

**Value**

An AAbin object.

**See Also**

Other conversions: [as-tibble-ape](#), [as-tibble-bioseq](#), [as\\_DNAbin\(\)](#), [as\\_aa\(\)](#), [as\\_dna\(\)](#), [as\\_rna\(\)](#), [as\\_seqinr\\_alignment\(\)](#)

---

as_AAbin.tbl_df	<i>Coerce tibble to AAbin</i>
-----------------	-------------------------------

---

**Description**

Coerce tibble to AAbin

**Usage**

```
## S3 method for class 'tbl_df'
as_AAbin(x, sequences, labels = NULL, ...)
```

**Arguments**

x	a tibble.
sequences	Name of the tibble column that stores the sequences.
labels	Name of the tibble column that stores the sequence labels.
...	Other params.

**Value**

An AAbin object.

---

as_dna	<i>Coercion to DNA vector</i>
--------	-------------------------------

---

**Description**

Coercion to DNA vector

**Usage**

```
as_dna(x)
```

**Arguments**

x	An object to coerce.
---	----------------------

**Value**

A DNA vector of class bioseq\_dna

**See Also**

Other conversions: [as-tibble-ape](#), [as-tibble-bioseq](#), [as\\_AAbin\(\)](#), [as\\_DNAbin\(\)](#), [as\\_aa\(\)](#), [as\\_rna\(\)](#), [as\\_seqinr\\_alignment\(\)](#)

---

as_DNABin	<i>Coerce to DNABin</i>
-----------	-------------------------

---

**Description**

Coerce to DNABin

**Usage**

```
as_DNABin(x, ...)
```

**Arguments**

x	An object.
...	Other parameters.

**Value**

A DNABin object.

**See Also**

Other conversions: [as-tibble-ape](#), [as-tibble-bioseq](#), [as\\_AABin\(\)](#), [as\\_aa\(\)](#), [as\\_dna\(\)](#), [as\\_rna\(\)](#), [as\\_seqnr\\_alignment\(\)](#)

---

as_DNABin.tbl_df	<i>Coerce tibble to DNABin</i>
------------------	--------------------------------

---

**Description**

Coerce tibble to DNABin

**Usage**

```
## S3 method for class 'tbl_df'  
as_DNABin(x, sequences, labels = NULL, ...)
```

**Arguments**

x	a tibble.
sequences	Name of the tibble column that stores the sequences.
labels	Name of the tibble column that stores the sequence labels.
...	Other params.

**Value**

A DNABin object.

---

as\_rna                      *Coercion to RNA vector*

---

**Description**

Coercion to RNA vector

**Usage**

```
as_rna(x)
```

**Arguments**

x                      An object to coerce.

**Value**

A RNA vector of class `bioseq_rna`

**See Also**

Other conversions: [as-tibble-ape](#), [as-tibble-bioseq](#), [as\\_AAbin\(\)](#), [as\\_DNAbin\(\)](#), [as\\_aa\(\)](#), [as\\_dna\(\)](#), [as\\_seqinr\\_alignment\(\)](#)

---

as\_seqinr\_alignment      *Coerce to seqinr alignment*

---

**Description**

Coerce to seqinr alignment

**Usage**

```
as_seqinr_alignment(x, ...)
```

**Arguments**

x                      An object.  
...                    Other parameters.

**Value**

An alignment object.

**See Also**

Other conversions: [as-tibble-ape](#), [as-tibble-bioseq](#), [as\\_AAbin\(\)](#), [as\\_DNAbin\(\)](#), [as\\_aa\(\)](#), [as\\_dna\(\)](#), [as\\_rna\(\)](#)

---

dic_genetic_codes	<i>Genetic code tables</i>
-------------------	----------------------------

---

**Description**

The function returns a list of named vectors with Start, Stop and Full\_name attributes.

**Usage**

```
dic_genetic_codes()
```

**Value**

A list of genetic code tables for DNA/RNA translation.

---

dna	<i>Build a DNA vector</i>
-----	---------------------------

---

**Description**

dna() build a DNA vector from a character vector.

**Usage**

```
dna(...)
```

**Arguments**

... characters to turn into DNA. Can be a set of name-value pairs.

**Value**

a vector of class bioseq\_dna

**See Also**

Other classes: [aa\(\)](#), [rna\(\)](#)

**Examples**

```
dna("AGGTGC", "TTCGA")  
  
dna(Seq_1 = "AGGTGC", Seq_2 = "TTCGA")  
  
x <- c("AGGTGC", "TTCGA")  
dna(x)
```

---

fragilaria

*DNA sequences (rbcL) for various Fragilaria*

---

### Description

An unparsed FASTA of DNA sequences (rbcL) for various strains of *Fragilaria* retrieved from NCBI.

### Usage

fragilaria

### Format

A long character vector (unparsed FASTA).

### Source

GenBank <https://www.ncbi.nlm.nih.gov/genbank/> using the following search term: "(rbcL) AND *Fragilaria*"

### See Also

[read\\_fasta](#) to parse these data.

---

genetic-codes

*Genetic code tables*

---

### Description

List of all genetic code tables available in `bioseq`. The number in bold can be used to select a table in appropriate functions.

### Available genetic codes

1. Standard
2. Vertebrate Mitochondrial
3. Yeast Mitochondrial
4. Mold Mitochondrial; Protozoan Mitochondrial; Coelenterate Mitochondrial; Mycoplasma; Spiroplasma
5. Invertebrate Mitochondrial
6. Ciliate Nuclear; Dasycladacean Nuclear; Hexamita Nuclear
9. Echinoderm Mitochondrial; Flatworm Mitochondrial
10. Euplotid Nuclear

11. Bacterial, Archaeal and Plant Plastid
12. Alternative Yeast Nuclear
13. Ascidian Mitochondrial
14. Alternative Flatworm Mitochondrial
15. Blepharisma Macronuclear
16. Chlorophycean Mitochondrial
21. Trematode Mitochondrial
22. Scenedesmus obliquus Mitochondrial
23. Thraustochytrium Mitochondrial
24. Pterobranchia Mitochondrial
25. Candidate Division SR1 and Gracilibacteria
26. Pachysolen tannophilus Nuclear
27. Karyorelict Nuclear
28. Condylostoma Nuclear
29. Mesodinium Nuclear
30. Peritrich Nuclear
31. Blastocrithidia Nuclear
32. Balanophoraceae Plastid
33. Cephalodiscidae Mitochondrial

## References

Andrzej (Anjay) Elzanowski and Jim Ostell at National Center for Biotechnology Information (NCBI), Bethesda, Maryland, U.S.A. <https://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/index.cgi?chapter=tgencodes>

---

is\_aa

*Test if the object is an amino acid vector*

---

## Description

This function returns TRUE for objects of class bioseq\_aa

## Usage

```
is_aa(x)
```

## Arguments

x                    An object.

**Value**

Logical.

**Examples**

```
x <- c("AGGTGC", "TTCGA")
is_aa(x)
y <- aa(x)
is_aa(x)
```

---

is\_dna

*Test if the object is a DNA vector*

---

**Description**

This function returns TRUE for objects of class bioseq\_dna

**Usage**

```
is_dna(x)
```

**Arguments**

x                    An object.

**Value**

Logical.

**Examples**

```
x <- c("AGGTGC", "TTCGA")
is_dna(x)
y <- dna(x)
is_dna(y)
```

---

is_rna	<i>Test if the object is a RNA vector</i>
--------	---

---

**Description**

This function returns TRUE for objects of class bioseq\_rna

**Usage**

```
is_rna(x)
```

**Arguments**

x                    An object.

**Value**

Logical.

**Examples**

```
x <- c("AGGTGC", "TTCGA")
is_rna(x)
y <- rna(x)
is_rna(x)
```

---

new_aa	<i>Amino acid (AA) vector constructor</i>
--------	---

---

**Description**

Amino acid (AA) vector constructor

**Usage**

```
new_aa(x = character())
```

**Arguments**

x                    a character vector.

---

new_dna	<i>DNA vector constructor</i>
---------	-------------------------------

---

**Description**

DNA vector constructor

**Usage**

```
new_dna(x = character())
```

**Arguments**

x	a character vector.
---	---------------------

---

new_rna	<i>RNA vector constructor</i>
---------	-------------------------------

---

**Description**

RNA vector constructor

**Usage**

```
new_rna(x = character())
```

**Arguments**

x	a character vector.
---	---------------------

---

read_fasta	<i>Read sequences in FASTA format</i>
------------	---------------------------------------

---

**Description**

Read sequences in FASTA format

**Usage**

```
read_fasta(file, type = "DNA")
```

**Arguments**

file	A path to a file, a connection or a character string.
type	Type of data. Can be "DNA" (the default), "RNA" or "AA".

**Value**

A DNA, RNA or AA vector (depending on type argument).

**See Also**

Other input/output operations: [write\\_fasta\(\)](#)

---

rev_complement	<i>Reverse and complement sequences</i>
----------------	---

---

**Description**

Reverse and complement sequences

**Usage**

```
seq_complement(x)
```

```
seq_reverse(x)
```

**Arguments**

x a DNA or RNA vector. Function `seq_reverse` also accepts AA vectors.

**Value**

A reverse or complement sequence (same class as the input).

**See Also**

Other biological operations: [seq\\_rev\\_translate\(\)](#), [seq\\_translate\(\)](#), [transcription](#)

**Examples**

```
x <- dna("ACTTTGGCTAAG")
seq_reverse(x)
seq_complement(x)
```

rna

*Build a RNA vector*

---

**Description**

rna() build a RNA vector from a character vector.

**Usage**

```
rna(...)
```

**Arguments**

... characters to turn into RNA. Can be a set of name-value pairs.

**Value**

a vector of class bioseq\_rna

**See Also**

Other classes: [aa\(\)](#), [dna\(\)](#)

**Examples**

```
rna("AGGUGC", "UUCGA")  
  
rna(Seq_1 = "AGGUGC", Seq_2 = "UUCGA")  
  
x <- c("AGGTGC", "TTCGA")  
rna(x)
```

---

seaview*SeaView: DNA sequences and phylogenetic tree viewer*

---

**Description**

This function opens SeaView (Gouy, Guindon & Gascuel, 2010) to visualize biological sequences and phylogenetic trees. The software must be installed on the computer.

**Usage**

```
seaview(  
  x,  
  seaview_exec = getOption("bioseq.seaview.exec", default = "seaview")  
)
```

**Arguments**

- `x` a DNA, RNA or AA vector. Alternatively a DNABin or AABin object or a phylogenetic tree (class phylo).
- `seaview_exec` a character string giving the path of the program.

**Details**

By default, the function assumes that the executable is installed in a directory located on the PATH. Alternatively the user can provide an absolute path to the executable (i.e. the location where the software was installed/uncompressed). This can be stored in the global options settings using `options(bioseq.seaview.exec = "my_path_to_seaview")`.

**References**

Gouy M., Guindon S. & Gascuel O. (2010) SeaView version 4 : a multiplatform graphical user interface for sequence alignment and phylogenetic tree building. *Molecular Biology and Evolution* 27(2):221-224.

**See Also**

Other GUI wrappers: [aliview\(\)](#)

---

seq-replace	<i>Replace matched patterns in sequences</i>
-------------	--

---

**Description**

Replace matched patterns in sequences

**Usage**

```
seq_replace_pattern(x, pattern, replacement)
```

**Arguments**

- `x` a DNA, RNA or AA vector.
- `pattern` a DNA, RNA or AA vectors (but same as `x`) or a character vector of regular expressions, or a list. See section Patterns.
- `replacement` a vector of replacements.

**Value**

A vector of same class as `x`.

## Patterns

It is important to understand how patterns are treated in **bioseq**.

Patterns are recycled along the sequences (usually the `x` argument). This means that if a pattern (vector or list) is of length  $> 1$ , it will be replicated until it is the same length as `x`. The reverse is not true and a vector of patterns longer than a vector of sequences will raise a warning.

Patterns can be DNA, RNA or AA vectors (but they must be from the same class as the sequences they are matched against). If patterns are DNA, RNA or AA vectors, they are disambiguated prior to matching. For example `pattern_dna("ARG")` will match AAG or AGG.

Alternatively, patterns can be a simple character vector containing regular expressions.

Vectors of patterns (DNA, RNA, AA or regex) can also be provided in a list. In that case, each vector of the list will be collapsed prior matching, which means that each vector element will be used as an alternative pattern. For example `pattern_list(c("AAA", "CCC"), "GG")` will match AAA or CCC in the first sequence, GG in the second sequence, AAA or CCC in the third, and so on following the recycling rule.

@section Fuzzy matching: When `max_error` is greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

## See Also

[stri\\_replace](#) from **stringi** and [str\\_replace](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

## Examples

```
x <- dna("ACGTTAGTGTAGCCGT", "CTCGAAATGA")
seq_replace_pattern(x, dna("AAA"), dna("GGGGGG"))
seq_replace_pattern(x, "^A.{2}T", "TTTTTT")
```

---

seq\_cluster

*Cluster sequences by similarity*

---

## Description

Cluster sequences by similarity

## Usage

```
seq_cluster(x, threshold = 0.05, method = "complete")
```

## Arguments

<code>x</code>	a DNA, RNA or AA vector of sequences to clustered.
<code>threshold</code>	Threshold value (range in [0, 1]).
<code>method</code>	the clustering method (see details).

## Details

The function uses **ape** [dist.dna](#) and [dist.aa](#) functions to compute pairwise distances among sequences and [hclust](#) for clustering.

Computing a full pairwise distance matrix can be computationally expensive. It is recommended to use this function for moderate size dataset.

Supported methods are:

- "single" (= Nearest Neighbour Clustering)
- "complete" (= Farthest Neighbour Clustering)
- "average" (= UPGMA)
- "mcquitty" (= WPGMA)

## Value

An integer vector with group memberships.

## See Also

Function [seq\\_consensus](#) to compute consensus and representative sequences for clusters.

Other aggregation operations: [seq\\_consensus\(\)](#)

## Examples

```
x <- c("-----TACGCAGTAAAAGCTACTGATG",
      "CGTCATACGCAGTAAAAACTACTGATG",
      "CTTCATACGCAGTAAAAACTACTGATG",
      "CTTCATATGCAGTAAAAACTACTGATG",
      "CTTCATACGCAGTAAAAACTACTGATG",
      "CGTCATACGCAGTAAAAGCTACTGATG",
      "CTTCATATGCAGTAAAAGCTACTGACG")
x <- dna(x)
seq_cluster(x)
```

---

seq\_combine

*Combine multiple sequences*

---

## Description

Combine multiple sequences

## Usage

```
seq_combine(..., sep = "", collapse = NULL)
```

**Arguments**

...	One or more vectors of sequences (DNA, RNA, AA). They must all be of the same type. Short vectors are recycled.
sep	String to insert between input vectors.
collapse	If not NULL, combine everything with this string as separator.

**Details**

The strings `sep` and `collapse` will be coerced to the type of input vectors with a warning if some character have to be replaced.

**Value**

A vector of sequences (if `collapse` is NULL). A vector with a single sequence, otherwise.

**See Also**

[stri\\_join](#) from **stringi** and [str\\_c](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

**Examples**

```
x <- dna("ACGTTAGTGTAGCCGT", "CTCGAAATGA")
y <- dna("TTTTTTTTT", "AAAAAAAAA")
seq_combine(x, y)
seq_combine(y, x, sep = "CCCCC")
seq_combine(y, x, sep = "CCCCC", collapse = "GGGGG")
```

---

seq\_consensus

*Find a consensus sequence for a set of sequences.*

---

**Description**

Find a consensus sequence for a set of sequences.

**Usage**

```
seq_consensus(x, method = "chr_majority", weights = NULL, gaps = TRUE)
```

**Arguments**

x	a DNA, RNA or AA vector.
method	the consensus method (see Details).
weights	an optional numeric vector of same length as x giving a weight for each input sequence.
gaps	logical. Should the gaps ("-") taken into account.

**Details**

"chr\_majority", "chr\_ambiguity", "seq\_centrality", "seq\_majority"

For chr\_ambiguity gap character always override other characters. Use gaps = FALSE to ignore gaps.

**Value**

A consensus sequence

**See Also**

Other aggregation operations: [seq\\_cluster\(\)](#)

**Examples**

```
x <- c("-----TACGCAGTAAAACTACTGATG",
      "CGTCATACGCAGTAAAACTACTGATG",
      "CTTCATACGCAGTAAAACTACTGATG",
      "CTTCATATGCAGTAAAACTACTGATG",
      "CTTCATACGCAGTAAAACTACTGATG",
      "CGTCATACGCAGTAAAACTACTGATG",
      "CTTCATATGCAGTAAAACTACTGACG")
x <- dna(x)
seq_consensus(x)
```

---

seq\_count\_pattern      *Count the number of matches in sequences*

---

**Description**

Count the number of matches in sequences

**Usage**

```
seq_count_pattern(x, pattern)
```

**Arguments**

x	a DNA, RNA or AA vector.
pattern	a DNA, RNA or AA vectors (but same as x) or a character vector of regular expressions, or a list. See section Patterns.

**Value**

An integer vector.

**Patterns**

It is important to understand how patterns are treated in **bioseq**.

Patterns are recycled along the sequences (usually the x argument). This means that if a pattern (vector or list) is of length > 1, it will be replicated until it is the same length as x. The reverse is not true and a vector of patterns longer than a vector of sequences will raise a warning.

Patterns can be DNA, RNA or AA vectors (but they must be from the same class as the sequences they are matched against). If patterns are DNA, RNA or AA vectors, they are disambiguated prior to matching. For example pattern dna("ARG") will match AAG or AGG.

Alternatively, patterns can be a simple character vector containing regular expressions.

Vectors of patterns (DNA, RNA, AA or regex) can also be provided in a list. In that case, each vector of the list will be collapsed prior matching, which means that each vector element will be used as an alternative pattern. For example pattern list(c("AAA", "CCC"), "GG") will match AAA or CCC in the first sequence, GG in the second sequence, AAA or CCC in the third, and so on following the recycling rule.

@section Fuzzy matching: When max\_error is greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

**See Also**

[stri\\_count](#) from **stringi** and [str\\_count](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

**Examples**

```
x <- dna("ACGTTAGTGTAGCCGT", "CTCGAAATGA")
seq_count_pattern(x, dna("AAA"))
seq_count_pattern(x, "T.G")
```

---

seq\_crop\_pattern      *Crop sequences using delimiting patterns*

---

## Description

Crop sequences using delimiting patterns

## Usage

```
seq_crop_pattern(  
  x,  
  pattern_in,  
  pattern_out,  
  max_error_in = 0,  
  max_error_out = 0,  
  include_patterns = TRUE  
)
```

## Arguments

**x**                    a DNA, RNA or AA vector to be cropped.

**pattern\_in**        patterns defining the beginning (left-side).

**pattern\_out**       patterns defining the end (right-side).

**max\_error\_in, max\_error\_out**  
                     numeric values ranging from 0 to 1 and giving the maximum error rate allowed between the target sequence and pattern\_in/pattern\_out. Error rate is relative to the length of the pattern.

**include\_patterns**  
                     logical. Should the matched pattern sequence included in the returned sequences?

## Value

A cropped DNA, RNA or AA vector. Sequences where patterns are not detected returns NA.

## Fuzzy matching

When `max_error_in` or `max_error_out` are greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

## Patterns

It is important to understand how patterns are treated in **bioseq**.

Patterns are recycled along the sequences (usually the `x` argument). This means that if a pattern (vector or list) is of length  $> 1$ , it will be replicated until it is the same length as `x`. The reverse is not true and a vector of patterns longer than a vector of sequences will raise a warning.

Patterns can be DNA, RNA or AA vectors (but they must be from the same class as the sequences they are matched against). If patterns are DNA, RNA or AA vectors, they are disambiguated prior to matching. For example `pattern dna("ARG")` will match AAG or AGG.

Alternatively, patterns can be a simple character vector containing regular expressions.

Vectors of patterns (DNA, RNA, AA or regex) can also be provided in a list. In that case, each vector of the list will be collapsed prior matching, which means that each vector element will be used as an alternative pattern. For example `pattern list(c("AAA", "CCC"), "GG")` will match AAA or CCC in the first sequence, GG in the second sequence, AAA or CCC in the third, and so on following the recycling rule.

@section Fuzzy matching: When `max_error` is greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

### See Also

`stri_extract` from **stringi**, `str_extract` from **stringr** and `afind` from **stringdist** for the underlying implementation.

Other string operations: `seq_replace`, `seq_combine()`, `seq_count_pattern()`, `seq_crop_position()`, `seq_detect_pattern()`, `seq_extract_pattern()`, `seq_extract_position()`, `seq_remove_pattern()`, `seq_remove_position()`, `seq_replace_position()`, `seq_split_kmer()`, `seq_split_pattern()`

### Examples

```
x <- dna("ACGTAAAAAGTGTAGCCCCCGT", "CTCGAAATGA")
seq_crop_pattern(x, pattern_in = "AAAA", pattern_out = "CCCC")
```

---

<code>seq_crop_position</code>	<i>Crop sequences between two positions</i>
--------------------------------	---

---

### Description

Crop sequences between two positions

### Usage

```
seq_crop_position(x, position_in = 1, position_out = -1)
```

### Arguments

<code>x</code>	a DNA, RNA or AA vector.
<code>position_in</code>	an integer giving the position where to start cropping.
<code>position_out</code>	an integer giving the position where to stop cropping.

### Value

A cropped DNA, RNA or AA vector.

**See Also**

`stri_sub` from **stringi** and `str_sub` from **stringr** for the underlying implementation.

Other string operations: `seq_replace`, `seq_combine()`, `seq_count_pattern()`, `seq_crop_pattern()`, `seq_detect_pattern()`, `seq_extract_pattern()`, `seq_extract_position()`, `seq_remove_pattern()`, `seq_remove_position()`, `seq_replace_position()`, `seq_split_kmer()`, `seq_split_pattern()`

**Examples**

```
x <- dna("ACGTTAGTG TAGCCGT")

# Drop the first 3 nucleotides (ACG)
seq_crop_position(x, position_in = 4)

# Crop codon between position 4 and 6
seq_crop_position(x, position_in = 4, position_out = 6)
```

---

seq\_detect\_pattern      *Detect the presence of patterns in sequences*

---

**Description**

Detect the presence of patterns in sequences

**Usage**

```
seq_detect_pattern(x, pattern, max_error = 0)
```

**Arguments**

x	a DNA, RNA or AA vector.
pattern	a DNA, RNA or AA vectors (but same as x) or a character vector of regular expressions, or a list. See section Patterns.
max_error	numeric value ranging from 0 to 1 and giving the maximum error rate allowed between the target sequence and the pattern. Error rate is relative to the length of the pattern.

**Value**

A logical vector.

## Patterns

It is important to understand how patterns are treated in **bioseq**.

Patterns are recycled along the sequences (usually the `x` argument). This means that if a pattern (vector or list) is of length  $> 1$ , it will be replicated until it is the same length as `x`. The reverse is not true and a vector of patterns longer than a vector of sequences will raise a warning.

Patterns can be DNA, RNA or AA vectors (but they must be from the same class as the sequences they are matched against). If patterns are DNA, RNA or AA vectors, they are disambiguated prior to matching. For example `pattern_dna("ARG")` will match AAG or AGG.

Alternatively, patterns can be a simple character vector containing regular expressions.

Vectors of patterns (DNA, RNA, AA or regex) can also be provided in a list. In that case, each vector of the list will be collapsed prior matching, which means that each vector element will be used as an alternative pattern. For example `pattern_list(c("AAA", "CCC"), "GG")` will match AAA or CCC in the first sequence, GG in the second sequence, AAA or CCC in the third, and so on following the recycling rule.

@section Fuzzy matching: When `max_error` is greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

## See Also

[stri\\_detect](#) from **stringi**, [str\\_detect](#) from **stringr** and [afind](#) from **stringdist** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

## Examples

```
x <- dna(c("ACGTTAGTGTAGCCGT", "CTCGAAATGA"))
seq_detect_pattern(x, dna(c("CCG", "AAA")))

# Regular expression
seq_detect_pattern(x, "^A.{2}T")

# Fuzzy matching
seq_detect_pattern(x, dna("AGG"), max_error = 0.2)
# No match. The pattern has three character, the max_error
# has to be > 1/3 to allow one character difference.

seq_detect_pattern(x, dna("AGG"), max_error = 0.4)
# Match
```

---

`seq_disambiguate_IUPAC`*Disambiguate biological sequences*

---

**Description**

This function finds all the combinations of sequences corresponding to a given vector of sequences with ambiguities (IUPAC codes).

**Usage**

```
seq_disambiguate_IUPAC(x)
```

**Arguments**

`x` a DNA, RNA or AA vector

**Value**

A list of DNA, RNA or AA vectors (depending on the input) giving all possible combinations.

**See Also**

Other op-misc: [seq\\_nchar\(\)](#), [seq\\_nseq\(\)](#), [seq\\_spellout\(\)](#), [seq\\_stat\\_gc\(\)](#), [seq\\_stat\\_prop\(\)](#)

**Examples**

```
x <- dna(c("AYCTGW", "CTTN"))
seq_disambiguate_IUPAC(x)

y <- seq_transcribe(x)
seq_disambiguate_IUPAC(y)

z <- aa("YJSNAALNX")
z <- seq_translate(y)
seq_disambiguate_IUPAC(z)
```

---

`seq_extract_pattern`*Extract matching patterns from sequences*

---

**Description**

Extract matching patterns from sequences

**Usage**

```
seq_extract_pattern(x, pattern)
```

**Arguments**

x	a DNA, RNA or AA vector.
pattern	a DNA, RNA or AA vectors (but same as x) or a character vector of regular expressions, or a list. See section Patterns.

**Value**

A list of vectors of same class as x.

**Patterns**

It is important to understand how patterns are treated in **bioseq**.

Patterns are recycled along the sequences (usually the x argument). This means that if a pattern (vector or list) is of length > 1, it will be replicated until it is the same length as x. The reverse is not true and a vector of patterns longer than a vector of sequences will raise a warning.

Patterns can be DNA, RNA or AA vectors (but they must be from the same class as the sequences they are matched against). If patterns are DNA, RNA or AA vectors, they are disambiguated prior to matching. For example pattern dna("ARG") will match AAG or AGG.

Alternatively, patterns can be a simple character vector containing regular expressions.

Vectors of patterns (DNA, RNA, AA or regex) can also be provided in a list. In that case, each vector of the list will be collapsed prior matching, which means that each vector element will be used as an alternative pattern. For example pattern list(c("AAA", "CCC"), "GG") will match AAA or CCC in the first sequence, GG in the second sequence, AAA or CCC in the third, and so on following the recycling rule.

@section Fuzzy matching: When max\_error is greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

**See Also**

[stri\\_extract](#) from **stringi** and [str\\_extract](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

**Examples**

```
x <- dna("ACGTTAGTGTAGCCGT", "CTCGAAATGA")
seq_extract_pattern(x, dna("AAA"))
seq_extract_pattern(x, "T.G")
```

---

seq\_extract\_position *Extract a region between two positions in sequences*

---

### Description

Extract a region between two positions in sequences

### Usage

```
seq_extract_position(x, position_in, position_out)
```

### Arguments

x                    a DNA, RNA or AA vector.  
position\_in        an integer giving the position where to start to extract.  
position\_out       an integer giving the position where to stop to extract.

### Value

A vector of same class as x.

### See Also

[stri\\_extract](#) from **stringi** and [str\\_extract](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

### Examples

```
x <- dna("ACGTTAGTGTAGCCGT", "CTCGAAATGA")  
seq_extract_position(x, 3, 8)
```

---

seq\_nchar *Count the number of character in sequences*

---

### Description

Count the number of character in sequences

### Usage

```
seq_nchar(x, gaps = TRUE)
```

**Arguments**

x                    a DNA, RNA or AA vector.  
gaps                if FALSE gaps are ignored.

**Value**

An integer vector giving the size of each sequence of x.

**See Also**

Other op-misc: [seq\\_disambiguate\\_IUPAC\(\)](#), [seq\\_nseq\(\)](#), [seq\\_spellout\(\)](#), [seq\\_stat\\_gc\(\)](#), [seq\\_stat\\_prop\(\)](#)

**Examples**

```
x <- dna(c("ATGCAGA", "GGR-----", "TTGCCTAGKTGAACC"))
seq_nchar(x)
seq_nchar(x, gaps = FALSE)
```

---

seq\_nseq

*Number of sequences in a vector*

---

**Description**

This is an alias for length.

**Usage**

```
seq_nseq(x)
```

**Arguments**

x                    a DNA, RNA or AA vector.

**Value**

an integer.

**See Also**

Other op-misc: [seq\\_disambiguate\\_IUPAC\(\)](#), [seq\\_nchar\(\)](#), [seq\\_spellout\(\)](#), [seq\\_stat\\_gc\(\)](#), [seq\\_stat\\_prop\(\)](#)

---

seq_remove_pattern	<i>Remove matched patterns in sequences</i>
--------------------	---

---

### Description

Remove matched patterns in sequences

### Usage

```
seq_remove_pattern(x, pattern)
```

### Arguments

x	a DNA, RNA or AA vector.
pattern	a DNA, RNA or AA vectors (but same as x) or a character vector of regular expressions, or a list. See section Patterns.

### Value

A vector of same class as x.

### Patterns

It is important to understand how patterns are treated in **bioseq**.

Patterns are recycled along the sequences (usually the x argument). This means that if a pattern (vector or list) is of length > 1, it will be replicated until it is the same length as x. The reverse is not true and a vector of patterns longer than a vector of sequences will raise a warning.

Patterns can be DNA, RNA or AA vectors (but they must be from the same class as the sequences they are matched against). If patterns are DNA, RNA or AA vectors, they are disambiguated prior to matching. For example pattern dna("ARG") will match AAG or AGG.

Alternatively, patterns can be a simple character vector containing regular expressions.

Vectors of patterns (DNA, RNA, AA or regex) can also be provided in a list. In that case, each vector of the list will be collapsed prior matching, which means that each vector element will be used as an alternative pattern. For example pattern list(c("AAA", "CCC"), "GG") will match AAA or CCC in the first sequence, GG in the second sequence, AAA or CCC in the third, and so on following the recycling rule.

@section Fuzzy matching: When max\_error is greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

### See Also

[str\\_remove](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

## Examples

```
x <- dna("ACGTTAGTG TAGCCGT", "CTCGAAATGA")
seq_remove_pattern(x, dna("AAA"))
seq_remove_pattern(x, "^A.{2}T")
```

---

seq\_remove\_position    *Remove a region between two positions in sequences.*

---

## Description

Remove a region between two positions in sequences.

## Usage

```
seq_remove_position(x, position_in, position_out)
```

## Arguments

**x**                    a DNA, RNA or AA vector.  
**position\_in**        an integer giving the position where to start to remove.  
**position\_out**       an integer giving the position where to stop to remove.

## Value

A vector of same class as x.

## See Also

[str\\_remove](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

## Examples

```
x <- dna("ACGTTAGTG TAGCCGT", "CTCGAAATGA")
seq_remove_position(x, 2, 6)
seq_remove_position(x, 1:2, 3:4)
```

---

seq\_replace\_position     *Replace a region between two positions in sequences*

---

### Description

Replace a region between two positions in sequences

### Usage

```
seq_replace_position(x, position_in, position_out, replacement)
```

### Arguments

x                     a DNA, RNA or AA vector.  
position\_in         an integer giving the position where to start to replace.  
position\_out        an integer giving the position where to stop to replace.  
replacement        a vector of replacements.

### Value

A vector of same class as x.

### See Also

[stri\\_replace](#) from **stringi** and [str\\_replace](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#), [seq\\_split\\_pattern\(\)](#)

### Examples

```
x <- dna("ACGTTAGTG TAGCCGT", "CTCGAAATGA")  
seq_replace_position(x, c(5, 2), 6, "-----")
```

---

seq\_rev\_translate         *Reverse translate amino acid sequences*

---

### Description

The function perform reverse translation of amino acid sequences. Such operation does not exist in nature but is provided for completeness. Because of codon degeneracy it is expected to produce many ambiguous nucleotides.

### Usage

```
seq_rev_translate(x, code = 1)
```

**Arguments**

x	an amino acid sequence (bioseq_aa)
code	an integer indicating the genetic code to use for reverse translation (default 1 uses the Standard genetic code). See Details.

**Details**

Gaps (-) are interpreted as unknown amino acids (X) but can be removed prior to the translation with the function `seq_remove_gap`.

**Value**

a vector of DNA sequences.

**See Also**

Other biological operations: [rev\\_complement](#), [seq\\_translate\(\)](#), [transcription](#)

**Examples**

```
x <- dna("ACTTTGGCTAAG")
y <- seq_translate(x)
z <- seq_rev_translate(y)
z
# There is a loss of information during the reverse translation
all.equal(x, z)
```

---

seq\_spellout

*Spell out sequences*

---

**Description**

This function spells out nucleotides and amino acids in sequences.

**Usage**

```
seq_spellout(x, short = FALSE, collapse = " - ")
```

**Arguments**

x	a DNA, RNA or AA vector
short	logical. If TRUE, the function will return 3-letters short names for amino acids (ignored for DNA and RNA).
collapse	a character vector to separate the results. Set to NULL to avoid collapsing the results.

**Value**

A character vector if collapse is not NULL. A list of character vectors otherwise.

**See Also**

Other op-misc: [seq\\_disambiguate\\_IUPAC\(\)](#), [seq\\_nchar\(\)](#), [seq\\_nseq\(\)](#), [seq\\_stat\\_gc\(\)](#), [seq\\_stat\\_prop\(\)](#)

**Examples**

```
x <- dna("ACGT")
seq_spellout(x)

x <- rna("ACGU")
seq_spellout(x)

x <- aa("ACGBTX")
seq_spellout(x)
```

---

seq\_split\_kmer

*Split sequences into k-mers*

---

**Description**

Split sequences into k-mers

**Usage**

```
seq_split_kmer(x, k)
```

**Arguments**

x	A DNA, RNA or AA vector.
k	an integer giving the size of the k-mer.

**Value**

a list of k-mer vectors of same class as x.

**See Also**

[seq\\_split\\_pattern](#).

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_pattern\(\)](#)

**Examples**

```
x <- dna(a = "ACGTTAGTGTAGCCGT", b = "CTCGAAATGA")
seq_split_kmer(x, k = 5)
```

---

seq\_split\_pattern      *Split sequences*

---

**Description**

Split sequences

**Usage**

```
seq_split_pattern(x, pattern)
```

**Arguments**

x	a DNA, RNA or AA vector.
pattern	a DNA, RNA or AA vectors (but same as x) or a character vector of regular expressions, or a list. See section Patterns.

**Value**

A list of vectors of same class as x.

**Patterns**

It is important to understand how patterns are treated in **bioseq**.

Patterns are recycled along the sequences (usually the x argument). This means that if a pattern (vector or list) is of length > 1, it will be replicated until it is the same length as x. The reverse is not true and a vector of patterns longer than a vector of sequences will raise a warning.

Patterns can be DNA, RNA or AA vectors (but they must be from the same class as the sequences they are matched against). If patterns are DNA, RNA or AA vectors, they are disambiguated prior to matching. For example pattern dna("ARG") will match AAG or AGG.

Alternatively, patterns can be a simple character vector containing regular expressions.

Vectors of patterns (DNA, RNA, AA or regex) can also be provided in a list. In that case, each vector of the list will be collapsed prior matching, which means that each vector element will be used as an alternative pattern. For example pattern list(c("AAA", "CCC"), "GG") will match AAA or CCC in the first sequence, GG in the second sequence, AAA or CCC in the third, and so on following the recycling rule.

@section Fuzzy matching: When max\_error is greater than zero, the function perform fuzzy matching. Fuzzy matching does not support regular expression.

**See Also**

[stri\\_split](#) from **stringi** and [str\\_split](#) from **stringr** for the underlying implementation.

Other string operations: [seq\\_replace](#), [seq\\_combine\(\)](#), [seq\\_count\\_pattern\(\)](#), [seq\\_crop\\_pattern\(\)](#), [seq\\_crop\\_position\(\)](#), [seq\\_detect\\_pattern\(\)](#), [seq\\_extract\\_pattern\(\)](#), [seq\\_extract\\_position\(\)](#), [seq\\_remove\\_pattern\(\)](#), [seq\\_remove\\_position\(\)](#), [seq\\_replace\\_position\(\)](#), [seq\\_split\\_kmer\(\)](#)

**Examples**

```
x <- dna(a = "ACGTTAGTG TAGCCGT", b = "CTCGAAATGA")
seq_split_pattern(x, dna("AAA"))
seq_split_pattern(x, "T.G")
```

---

seq\_stat\_gc

*Compute G+C content*


---

**Description**

Compute G+C content

**Usage**

```
seq_stat_gc(x)
```

**Arguments**

x                    a DNA or RNA

**Details**

Ambiguous characters (other than S and W) are ignored.

**Value**

A numeric vector of G+C proportions.

**See Also**

Other op-misc: [seq\\_disambiguate\\_IUPAC\(\)](#), [seq\\_nchar\(\)](#), [seq\\_nseq\(\)](#), [seq\\_spellout\(\)](#), [seq\\_stat\\_prop\(\)](#)

**Examples**

```
x <- dna(c("ATGCAGA", "GGR-----", "TTGCCTAGKTGAACC"))
seq_stat_gc(x)
```

---

seq\_stat\_prop                    *Compute proportions for characters*

---

**Description**

Compute proportions for characters

**Usage**

```
seq_stat_prop(x, gaps = FALSE)
```

**Arguments**

x                    a DNA, RNA or AA vector.  
gaps                if FALSE gaps are ignored.

**Value**

A list of vectors indicating the proportion of characters in each sequence.

**See Also**

Other op-misc: [seq\\_disambiguate\\_IUPAC\(\)](#), [seq\\_nchar\(\)](#), [seq\\_nseq\(\)](#), [seq\\_spellout\(\)](#), [seq\\_stat\\_gc\(\)](#)

**Examples**

```
x <- dna(c("ATGCAGA", "GGR-----", "TTGCCTAGKTGAACC"))  
seq_stat_prop(x)  
seq_stat_prop(x, gaps = TRUE)
```

---

seq\_translate                    *Translate DNA/RNA sequences into amino acids*

---

**Description**

Translate DNA/RNA sequences into amino acids

**Usage**

```
seq_translate(x, code = 1, codon_frame = 1, codon_init = FALSE)
```

**Arguments**

<code>x</code>	a vector of DNA ( <code>bioseq_dna</code> ) or RNA ( <code>bioseq_rna</code> ).
<code>code</code>	an integer indicating the genetic code to use for translation (default 1 uses the Standard genetic code). See Details.
<code>codon_frame</code>	an integer giving the nucleotide position where to start translation.
<code>codon_init</code>	a logical indicating whether the first codon is evaluated as a possible codon start and translated to methionine.

**Details**

Several genetic codes can be used for translation. See [genetic-codes](#) to get the list of available genetic codes and their ID number.

Gaps (-) are interpreted as unknown nucleotides (N) but can be removed prior to the translation with the function `seq_remove_gap`.

The function deals with ambiguities on both sides. This means that if ambiguous codons cannot be translated to amino acid, they are translated to the most specific ambiguous amino acids (X in the most extreme case).

**Value**

An amino acid vector (`bioseq_aa`).

**See Also**

Other biological operations: [rev\\_complement](#), [seq\\_rev\\_translate\(\)](#), [transcription](#)

**Examples**

```
x <- dna(c("ATGCAGA", "GGR", "TTGCCTAGKTGAACC", "AGNGC", "NNN"))
seq_translate(x)
```

---

<code>transcription</code>	<i>Transcribe DNA, reverse-transcribe RNA</i>
----------------------------	---

---

**Description**

Transcribe DNA, reverse-transcribe RNA

**Usage**

```
seq_transcribe(x)
```

```
seq_rev_transcribe(x)
```

**Arguments**

x                    A vector of DNA for seq\_transcribe, a vector of RNA for seq\_rev\_transcribe

**Value**

A vector of RNA for seq\_transcribe, a vector of DNA for seq\_rev\_transcribe

**See Also**

Other biological operations: [rev\\_complement](#), [seq\\_rev\\_translate\(\)](#), [seq\\_translate\(\)](#)

---

<code>write_fasta</code>	<i>Write sequences in FASTA format</i>
--------------------------	--

---

**Description**

Write sequences in FASTA format

**Usage**

```
write_fasta(x, file, append = FALSE, line_length = 80, block_length = 10)
```

**Arguments**

<code>x</code>	a DNA, RNA or AA vector.
<code>file</code>	a path to a file or a connection.
<code>append</code>	a logical. If TRUE append the data to the file. If FALSE (default), overwrite the file.
<code>line_length</code>	length (in number of character) of one line (excluding spaces separating blocks). Use Inf to avoid line breaks.
<code>block_length</code>	length (in number of character) of one block. Use the same value as <code>line_length</code> or Inf to avoid block separation.

**See Also**

Other input/output operations: [read\\_fasta\(\)](#)

# Index

- \* **GUI wrappers**
    - aliview, 4
    - seaview, 18
  - \* **aggregation operations**
    - seq\_cluster, 20
    - seq\_consensus, 22
  - \* **biological operations**
    - rev\_complement, 17
    - seq\_rev\_translate, 35
    - seq\_translate, 40
    - transcription, 41
  - \* **classes**
    - aa, 3
    - dna, 11
    - rna, 18
  - \* **conversions**
    - as-tibble-ape, 5
    - as-tibble-bioseq, 6
    - as\_aa, 7
    - as\_AAabin, 7
    - as\_dna, 8
    - as\_DNAbin, 9
    - as\_rna, 10
    - as\_seqinr\_alignment, 10
  - \* **datasets**
    - fragilaria, 12
  - \* **input/output operations**
    - read\_fasta, 16
    - write\_fasta, 42
  - \* **op-misc**
    - seq\_disambiguate\_IUPAC, 29
    - seq\_nchar, 31
    - seq\_nseq, 32
    - seq\_spellout, 36
    - seq\_stat\_gc, 39
    - seq\_stat\_prop, 40
  - \* **string operations**
    - seq\_replace, 19
    - seq\_combine, 21
    - seq\_count\_pattern, 23
    - seq\_crop\_pattern, 25
    - seq\_crop\_position, 26
    - seq\_detect\_pattern, 27
    - seq\_extract\_pattern, 29
    - seq\_extract\_position, 31
    - seq\_remove\_pattern, 33
    - seq\_remove\_position, 34
    - seq\_replace\_position, 35
    - seq\_split\_kmer, 37
    - seq\_split\_pattern, 38
- aa, 3, 11, 18
- afind, 26, 28
- aliview, 4, 19
- alphabets, 4
- as-tibble-ape, 5
- as-tibble-bioseq, 6
- as\_aa, 5–7, 7, 8–10
- as\_AAabin, 5–7, 7, 8–10
- as\_AAabin.tbl\_df, 8
- as\_dna, 5–7, 8, 9, 10
- as\_DNAbin, 5–8, 9, 10
- as\_DNAbin.tbl\_df, 9
- as\_rna, 5–10, 10
- as\_seqinr\_alignment, 5–10, 10
- as\_tibble.AAbin (as-tibble-ape), 5
- as\_tibble.bioseq\_aa (as-tibble-bioseq), 6
- as\_tibble.bioseq\_dna (as-tibble-bioseq), 6
- as\_tibble.bioseq\_rna (as-tibble-bioseq), 6
- as\_tibble.DNAbin (as-tibble-ape), 5
- dic\_genetic\_codes, 11
- dist.aa, 21
- dist.dna, 21
- dna, 3, 11, 18

- fragilaria, 12
- genetic-codes, 12, 41
- hclust, 21
- is\_aa, 13
- is\_dna, 14
- is\_rna, 15
- new\_aa, 15
- new\_dna, 16
- new\_rna, 16
- read\_fasta, 12, 16, 42
- rev\_complement, 17, 36, 41, 42
- rna, 3, 11, 18
- seaview, 4, 18
- seq-replace, 19
- seq\_cluster, 20, 23
- seq\_combine, 20, 21, 24, 26–28, 30, 31, 33–35, 37, 39
- seq\_complement (rev\_complement), 17
- seq\_consensus, 21, 22
- seq\_count\_pattern, 20, 22, 23, 26–28, 30, 31, 33–35, 37, 39
- seq\_crop\_pattern, 20, 22, 24, 25, 27, 28, 30, 31, 33–35, 37, 39
- seq\_crop\_position, 20, 22, 24, 26, 26, 28, 30, 31, 33–35, 37, 39
- seq\_detect\_pattern, 20, 22, 24, 26, 27, 27, 30, 31, 33–35, 37, 39
- seq\_disambiguate\_IUPAC, 29, 32, 37, 39, 40
- seq\_extract\_pattern, 20, 22, 24, 26–28, 29, 31, 33–35, 37, 39
- seq\_extract\_position, 20, 22, 24, 26–28, 30, 31, 33–35, 37, 39
- seq\_nchar, 29, 31, 32, 37, 39, 40
- seq\_nseq, 29, 32, 32, 37, 39, 40
- seq\_remove\_pattern, 20, 22, 24, 26–28, 30, 31, 33, 34, 35, 37, 39
- seq\_remove\_position, 20, 22, 24, 26–28, 30, 31, 33, 34, 35, 37, 39
- seq\_replace\_pattern (seq-replace), 19
- seq\_replace\_position, 20, 22, 24, 26–28, 30, 31, 33, 34, 35, 37, 39
- seq\_rev\_transcribe (transcription), 41
- seq\_rev\_translate, 17, 35, 41, 42
- seq\_reverse (rev\_complement), 17
- seq\_spellout, 29, 32, 36, 39, 40
- seq\_split\_kmer, 20, 22, 24, 26–28, 30, 31, 33–35, 37, 39
- seq\_split\_pattern, 20, 22, 24, 26–28, 30, 31, 33–35, 37, 38
- seq\_stat\_gc, 29, 32, 37, 39, 40
- seq\_stat\_prop, 29, 32, 37, 39, 40
- seq\_transcribe (transcription), 41
- seq\_translate, 17, 36, 40, 42
- str\_c, 22
- str\_count, 24
- str\_detect, 28
- str\_extract, 26, 30, 31
- str\_remove, 33, 34
- str\_replace, 20, 35
- str\_split, 39
- str\_sub, 27
- stri\_count, 24
- stri\_detect, 28
- stri\_extract, 26, 30, 31
- stri\_join, 22
- stri\_replace, 20, 35
- stri\_split, 39
- stri\_sub, 27
- transcription, 17, 36, 41, 41
- write\_fasta, 17, 42