# Package: R4RNA (via r-universe)

September 28, 2024

**Type** Package

**Title** An R package for RNA visualization and analysis

**Version** 1.33.0

**Date** 2015-11-02

**Author** Daniel Lai, Irmtraud Meyer <irmtraud.meyer@cantab.net>

**Maintainer** Daniel Lai <jujubix@cs.ubc.ca>

**Depends** R (>= 3.2.0), Biostrings (>= 2.38.0)

**Description** A package for RNA basepair analysis, including the visualization of basepairs as arc diagrams for easy comparison and annotation of sequence and structure. Arc diagrams can additionally be projected onto multiple sequence alignments to assess basepair conservation and covariation, with numerical methods for computing statistics for each.

**License** GPL-3

**biocViews** Alignment, MultipleSequenceAlignment, Preprocessing, Visualization, DataImport, DataRepresentation, MultipleComparison

**URL** http://www.e-rna.org/r-chie/

**NeedsCompilation** no

**Repository** https://bioc.r-universe.dev

**RemoteUrl** https://github.com/bioc/R4RNA

**RemoteRef** HEAD

**RemoteSha** 6cc90854e669612b01b29e31e13a7dbbc7949c78

## Contents

---

R4RNA-package *An R package for RNA visualization and analysis*

---

## Description

An R package for RNA visualization and analysis

## Examples

```
  # Read input data
  predicted <- readHelix(system.file("extdata", "helix.txt", package = "R4RNA"))
  known <- readVienna(system.file("extdata", "vienna.txt", package = "R4RNA"))
sequence <- as.character(readBStringSet(system.file("extdata", "fasta.txt", package = "R4RNA")))

  plotHelix(predicted)
  pval.coloured <- colourByValue(predicted, log = TRUE, get = TRUE)
  plotDoubleHelix(pval.coloured, known, scale = FALSE)
  plotOverlapHelix(pval.coloured, known)

  cov.coloured <- colourByCovariation(known, sequence, get = TRUE)
  plotCovariance(sequence, cov.coloured)

  plotDoubleCovariance(cov.coloured, pval.coloured, sequence,
      conflict.filter = "grey")
  plotOverlapCovariance(pval.coloured, known, sequence, grid = TRUE,
      conflict.filter = "grey", legend = FALSE, any = TRUE)

  # List of all functions
  ls("package:R4RNA")
  # use example() and help() for more details on each function
```

`Alignment Statistics` *Compute statistics for a multiple sequence alignments*

## Description

Functions to compute covariation, percent identity conservation, and percent canonical basepairs given a multiple sequence alignment and optionally a secondary structure. Statistics can be computed for a single base, basepair, helix or entire alignment.

## Usage

```
baseConservation(msa, pos)

basepairConservation(msa, pos.5p, pos.3p)
basepairCovariation(msa, pos.5p, pos.3p)
basepairCanonical(msa, pos.5p, pos.3p)

helixConservation(helix, msa)
helixCovariation(helix, msa)
helixCanonical(helix, msa)

alignmentConservation(msa)
alignmentCovariation(msa, helix)
alignmentCanonical(msa, helix)

alignmentPercentGaps(msa)
```

## Arguments

| | |
|---|---|
| `helix` | A helix data.frame |
| `msa` | A multiple sequence alignment. Can be either a `Biostrings` `XStringSet` object or a named array of strings like ones obtained from converting XStringSet with `as.character`. |
| `pos`, `pos.5p`, `pos.3p` | Positions of bases or basepairs for which statistics shall be calculated for. |

## Details

Conservation values have a range of [0, 1], where 0 is the absence of primary sequence conservation (all bases different), and 1 is full primary sequence conservation (all bases identical).

Canonical values have a range of [0, 1], where 0 is a complete lack of basepair potential, and 1 indicates that all basepairs are valid

Covariation values have a range of [-2, 2], where -2 is a complete lack of basepair potential and sequence conservation, 0 is complete sequence conservation regardless of basepairing potential, and 2 is a complete lack of sequence conservation but maintaining full basepair potential.

helix values are average of base/basepair values, and the alignment values are averages of helices or all columns depending on whether the helix argument is required.

alignmentPercentGaps simply returns the percentage of nucleotides that are gaps in a sequence for each sequence of the alignment.

## Value

baseConservation, basepairConservation, basepairCovariation, basepairCanonical, alignmentConservation, alignmentCovariation, and alignmentCanonical return a single decimal value.

helixConservation, helixCovariation, helixCanonical return a list of values whose length equals the number of rows in helix.

alignmentPercentGaps returns a list of values whose length equals the number of sequences in the multiple sequence alignment.

## Author(s)

Jeff Proctor, Daniel Lai

## Examples

```
data(helix)

baseConservation(fasta, 9)

basepairConservation(fasta, 9, 18)
basepairCovariation(fasta, 9, 18)
basepairCanonical(fasta, 9, 18)

helixConservation(helix, fasta)
helixCovariation(helix, fasta)
helixCanonical(helix, fasta)

alignmentConservation(fasta)
alignmentCovariation(fasta, helix)
alignmentCanonical(fasta, helix)

alignmentPercentGaps(fasta)
```

---

# Basepair Frequency *Calculates the frequency of each basepair*

---

## Description

Calculates the frequency of each basepair in a given helix structure. Internally, breaks helices into basepairs, and returns a structure of unique basepairs, where the values is its frequency, regardless of original value.

### Usage

```
basepairFrequency(helix)
```

### Arguments

| | |
|---|---|
| helix | A helix data.frame |

### Value

A helix data.frame of unique basepairs of length 1, with the frequency of appearance as its value, sorted by decreasing value.

### Author(s)

Daniel Lai

### See Also

colourByBasepairFrequency

### Examples

```
data(helix)
basepairFrequency(helix)
```

---

## Basepair/Helix Conversion

*Expand or collapse helices to and from basepairs*

---

### Description

Given a helix data frame, expands a helix of arbitrary length into helices of length 1 (i.e. basepairs). Also does the reverse operation of clustering consecutive basepairs (or helices), and merging/collapsing them into a single helix.

### Usage

```
expandHelix(helix)
collapseHelix(helix, number = FALSE)
```

### Arguments

| | |
|---|---|
| helix | A helix data frame. |
| number | Indicates presence of a column in the helix data frame titled exactly 'number', which will be used to unique identify basepairs belonging to the same helix. Only basepairs from the same helix as identified by the number will be collapsed together. |

### Details

During the expansion, basepairs expanded from a single helix will all be assigned the value of the originating helix (the same goes for all other columns besides i, j, and length). During collapsing, only helices/basepairs of equal value will be grouped together. The ordering of collapsed helices returned will be sorted by value (increasing order). For any other columns besides i, j, length and value, values will be obtained from the corresponding columns of the outer most basepair.

### Value

Returns a helix data frame.

### Author(s)

Daniel Lai

### Examples

```
# Create helix data frame
helix <- data.frame(2, 8, 3, 0.5)
helix[2, ] <- c(5, 15, 4, -0.5)
helix <- as.helix(helix)
helix$colour <- c("red", "blue")

# Before expansion
print(helix)
# After expansion
print(expanded <- expandHelix(helix))
# Collapse back (sorted by value)
print(collapseHelix(expanded))
```

---

## Coerce to Helix — *Coerce to a Helix Data Frame*

---

### Description

Functions to coerce a structure into a helix data frame, and to check whether a structure is a valid helix data frame. A helix data frame is a data frame, so any structure coercible into a data.frame can become a helix data frame.

### Usage

```
as.helix(x, length)
is.helix(x)
```

### Arguments

| | |
|---|---|
| x | Structure to coerce. Should be a structure coercible into a standard R data.frame structure for `as.helix`. Should be a string for `parseBracket`. May be anything for `is.helix`. |
| length | The length of the RNA sequence containing the helices. |

### Details

as.helix takes in a data.frame and coerces it into a helix data frame acceptable by other R4RNA functions. This mainly involves setting specific column names and casting to specific types.

### Value

is.helix returns a boolean.

as.helix returns helix data frame with valid input.

### Author(s)

Daniel Lai

### Examples

```
# Not a valid helix data frame
helix <- data.frame(c(1, 2, 3), seq(10, 20, length.out = 3), 5, runif(3))
is.helix(helix)
warnings()

# Formatted into a helix data frame
helix <- as.helix(helix)
is.helix(helix)
```

---

## Colour Helices *Assign colours to helices*

---

### Description

Functions to generate colours for helices by various rules, including integer counts, value ranges, percent identity covariation, conservation, percentage canonical basepair, basepair frequency, and non-pseudoknotted groups.

### Usage

```
colourByCount(helix, cols, counts, get = FALSE)
colourByValue(helix, cols, breaks, get = FALSE,
    log = FALSE, include.lowest = TRUE, ...)
colourByBasepairFrequency(helix, cols, get = TRUE)
colourByUnknottedGroups(helix, cols, get = TRUE)
colourByCovariation(helix, msa, cols, get = FALSE)
colourByConservation(helix, msa, cols, get = FALSE)
colourByCanonical(helix, msa, cols, get = FALSE)
defaultPalette()
```

### Arguments

| | |
|---|---|
| `helix` | A helix data frame to be coloured. |
| `cols` | An array of characters (or numbers) representing a set of colours to colour `helix` with. When missing, a default set of colours from defaultPalette() will be used. Valid input include hex codes, colour names from the `colours` function, and integer numbers. The colours will be interpreted as being from best to worst. |
| `counts` | An array of integers the same length as `cols`, dictating the number of times each corresponding colour should be used. When missing, the function will divide the number of helices evenly over each of the colours available. |
| `breaks` | An integer number of intervals to break the 'value' column of `helix` into, or a list of numbers defining the interval breaks. If missing, the range of 'helix$value' will automatically be split evenly into intervals for each colour available. |
| `get` | If TRUE, returns the input `helix` with a `col` column, else simply returns an array of colours the same length as the number of row in `helix`. The exceptions are `colourByBasepairFrequency` and `colourByUnknottedGroups` which will return a different helix if TRUE, and a list of colours that will not match the input helix if FALSE. |
| `log` | If TRUE, will breaks values into even log10 space intervals, useful when values are p-values. |
| `include.lowest` | Whether the lowest interval should include the lowest value, passed to `cut` |
| `...` | Additional arguments passed to `cut`, potentially useful ones include `right` (whether intervals should be inclusive on the right or left) and `dig.lab` (number of digits in interval labels). |
| `msa` | A multiple sequence alignment. Can be either a `Biostrings` `XStringSet` object or a named array of strings like ones obtained from converting XStringSet with `as.character`. |

### Details

`colourByCount` assigns colours indepenent of the helix input's value column, and instead operates over the number of helices (i.e. rows).

`colourByValue` uses `cut` to assign each of the helices to an interval based on its value.

`colourByCovariation`, `colourByConservation` , and `colourByCanonical`, colour helices according to compensatory mutations (or covariation), percentage identity conservation, and percentage canonical basepair repsectively, relative to the multiple sequence alignment provided.

`colourByBasepairFrequency` colours each basepair according to the number of times it appear in the input, regardless of its value.

`colourByUnknottedGroups` greedily partitions the basepairs into non- pseudoknotted groups, and assigns a colour to each.

### Value

All "colourBy" functions return a list of colours when `get = FALSE`, and a helix with a `col` column if `get = TRUE`. In both bases, the returned object has attributes "legend" and "fill", showing

the mapping between interval (in legend) and colour (in fill), which can as eponymous arguments legend.

defaultPalette returns the default list of colours.

## Author(s)

Daniel Lai

## See Also

plotHelix

logseq

basepairFrequency

unknottedGroups

## Examples

```
data(helix)

known$col <- colourByCount(known)
plotHelix(known)

plotHelix(colourByValue(helix, log = TRUE, get = TRUE))

cov <- colourByCovariation(known, fasta, get = TRUE)
plotCovariance(fasta, cov)
legend("topleft", legend = attr(cov, "legend"),
    fill = attr(cov, "fill"), title = "Covariation")
```

---

# Convert Helix Formats *Convert helix structures to and from other formats*

---

## Description

Converts dot bracket vienna format to and from helix format. It should be noted that the allows structures of vienna is a subset of those allowed in the helix format. Thus, conversion from vienna to helix will yield the identical structure, while conversion from helix to vienna may result in the loss of certain basepairs (mainly those that are conflicting). Pseudoknots are supported in both directions of conversion with limitations.

## Usage

```
viennaToHelix(vienna, value = NA, palette = NA)
helixToVienna(helix)
helixToConnect(helix)
helixToBpseq(helix)
```

### Arguments

| | |
|---|---|
| `vienna` | A string containing *only* a vienna dot bracket structure, with balanced brackets. Allowable brackets are (, <, [, {, A, B, C, and D (where upper-case alphabets are paired with lower-case alphabets). |
| `value` | A numerical value to assign to all helices. |
| `palette` | A list of colour names for up to 8 colours that will be used to colour brackets of type (, <, [, {, A, B, C, and D, respectively. |
| `helix` | A helix data.frame. |

### Details

`viennaToHelix` will ignore any non dot-bracket characters prior to parsing, so the resultant length will be shorter than expected if invalid characters are included.

If the colour palette is less than the number of supported brackets, it will simply cycle through the list. To explicitly prevent the colouring/ display of specific bracket type, colour it "NA".

For `helixToVienna`, pseudoknotted basepairs will be assigned different bracket types. As there are only 8 supported bracket types, any basepair pseudonotted deeper than 8 levels will be excluded from the output. Additionally, vienna format is unable to respresent conflicting basepairs, so conflicting basepairs will also be excluded. For both types of exclusion, those at the bottom of the helix data.frame will always be excluded in favour of keeping helices higher on the data.frame table.

`helixToConnect` and `helixToBpseq` will convert a *non-conflicting* helix data.frame into connect or bpseq format repsectively, provided the helix structure has a "sequence" attribute containing a single nucleotide sequence of the structure.

### Value

`viennaToHelix` returns a helix data.frame. `helixToVienna` returns a character string of basepairs in the Vienna helix format. `helixToConnect` and `HelixTpBpseq` return data.frames in the connect and bpseq formats, respectively.

### Author(s)

Daniel Lai

### Examples

```
# viennaToHelix demonstrating ALL valid bracket symbols
dot_bracket <- ".....(<[{.....ABCD.....}]>).....dcba....."
parsed <- viennaToHelix(dot_bracket, -31.5)
print(parsed)

vienna <- helixToVienna(parsed)
print(vienna)

# Colouring the brackets by bracket type
colour <- c("red", "orange", "yellow", "green", "lightblue", "blue", "purple", "black")
double.rainbow <- viennaToHelix(dot_bracket, 0, colour)
plotHelix(double.rainbow)
```

`Covariation Plots` *Plot nucleotide sequence coloured by covariance*

## Description

Given a multiple sequence alignment and a corresponding secondary structure, nucleotides in the sequence alignment will be coloured according to the basepairing and conservation status, where green is the most commonly observed valid basepair in the column, dark blue being valid covariation (i.e. mutation into another valid basepair), cyan is one-sided mutation that retains the basepair, and red is a mutation where the basepair has been lost.

## Usage

```
plotCovariance(msa, helix, arcs = TRUE, add = FALSE, grid = FALSE, text =
  FALSE, legend = TRUE, species = 0, base.colour = FALSE, palette = NA, flip =
  FALSE, grid.col = "white", grid.lwd = 0, text.cex = 0.5, text.col = "white",
 text.font = 2, text.family = "sans", species.cex = 0.5, species.col = "black",
 species.font = 2, species.family = "mono", shape = "circle", conflict.cutoff =
  0.01, conflict.lty = 2, conflict.col = NA, pad = c(0, 0, 0, 0), y = 0, x = 0,
    ...)
plotDoubleCovariance(top.helix, bot.helix, top.msa, bot.msa = top.msa,
    add = FALSE, grid = FALSE, species = 0, legend = TRUE,
    pad = c(0, 0, 0, 0), ...)
plotOverlapCovariance(predict.helix, known.helix, msa, bot.msa = TRUE,
   overlap.cutoff = 1, miss = "black", add = FALSE, grid = FALSE, species = 0,
    legend = TRUE, pad = c(0, 0, 0, 0), ...)
```

## Arguments

`msa`, `top.msa`, `bot.msa`

A multiple sequence alignment. Can be either a `Biostrings` `XStringSet` object or a named array of strings like ones obtained from converting XStringSet with `as.character`.

`top.msa` and `bot.msa` are specific to `top.helix` and `bot.helix` respectively, and may be set to NA to have no multiple sequence alignment at all.

`helix`, `top.helix`, `bot.helix`, `predict.helix`, `known.helix`

A helix data.frame with a structure corresponding to `msa`,

See `plotDoubleHelix` and `plotOverlapHelix` for detailed explanations of `top.helix`, `bot.helix`, `predict.helix`, and `known.helix`.

`arcs` TRUE if the structure should be plotted as arcs. Arcs may be styled with styling columns, see example and `plotHelix` for details.

`add` TRUE if graphical elements are to be added to an existing device, else a new plotting device is created with `blankPlot`.

`grid` TRUE if the multiple sequence alignment is to be drawn as a grid of bases, else the multiple sequence alignment is drawn as equidistant horizontal lines.

`text` Only applicable when grid is TRUE. TRUE if the grid is to be filled with nucleotide character.

`legend` TRUE if legend are to be shown.

`species` If a number greater than 0 is given, then species names for the multiple sequence alignment will be printed along the left side. This name is typically the entire header lines of FASTA entries. The number specifies the start position relative to the left edge of the multiple sequence alignment).

`base.colour` TRUE if bases are to be coloured by nucleotide instead of basepair conservation.

`palette` A list of colour names to override the default colour palette. When base.colour is TRUE, the first 6 colours will be used for colouring bases A, U, G, C, - (gap), and ? (everything else), respectively. When base.colour is FALSE, the first 7 colours will be used for colouring conserved basepairs, covarying basepairs, one-sided conserved basepairs, invalid basepairs, unpaired bases, gaps, and bases/pairs with ambiguous bases, resepctively. If the palette is shorter than the expected length, the palette will simply cycle. "NA" is a valid colour, that will effectively plot nothing.

`flip` If TRUE, the entire plot will be flipped upside down. Note that this is *not* a perfect mirror image about the horizon.

`grid.col`, `grid.lwd`
The colour and line width of the borders displayed when `grid` is TRUE.

`text.cex`, `text.col`, `text.font`, `text.family`
cex, col, family and font for the text displayed via the `text` option. Use `help("par")` for more information the paramters.

`species.cex`, `species.col`, `species.font`, `species.family`
cex, col, family and font for the species text displayed via the `species` option. Use `help("par")` for more information the paramters.

`shape` One of "circle", "triangle", or "square", specifying the shape of the arcs.

`conflict.lty`, `conflict.col`, `conflict.cutoff`
Determines the line type (style) and colour to be used for conflicting basepairs. By default, conflicting helices are drawn as dotted lines (`lty = 2`) and whatever colour was originally assigned to it (`col = NA`). Conflicting helices may be coloured by setting `conflict.col` to some R-compatible colour name. If both arguments are set to `NA`, then no attempt to exclude conflicting helices will be made when colouring covariance plot columns, which in most cases will render the plot nonsensical. When the input has helices with multiple basepairs, and only part of the helix is conflicting, the `conflict.cutoff` determines above what percentage of basepairs have to be conflicting before a helix is considered conflicting, with the default set at 1 conflicting).

`miss` The colour for unpredicted arcs in overlapping diagrams, see `plotOverlapHelix` for more information.

`overlap.cutoff` Decimal between 0 and 1 indicating the percentage of basepairs within a helix that have to be overlapping for the entire helix to count as overlapping. Default is 1, or 100

`pad` A four integer array passed to `blankPlot`, specifies the number of pixels to pad the bottom, left, top and right sides of the figure with, repsectively.

| | |
|---|---|
| x, y | Coordinates for the left bottom corner of the plot. Useful for manually positioning and overlapping figure elements. |
| ... | In `plotCovariance`, these are additional arguments passed to `blankPlot`, useful arguments include 'lwd', 'col', 'cex' for line width, line colour, and text size, respectively. `help('par')` for more.<br>For `plotDoubleCovariance` and `plotOverlapCovariance`, these are additional arguments passed to `plotCovariance` (and thus indirectly also to `blankPlot`). |

## Value

Not intended to return a value, will plot to GUI or file if specific.

## Author(s)

Daniel Lai

## See Also

`plotHelix`

`plotDoubleHelix`

`plotOverlapHelix`

`colourByCovariation`

`colourByConservation`

`colourByCanonical`

## Examples

```
data(helix)

# Basic covariance plot
plotCovariance(fasta, known, cex = 0.8, lwd = 1.5)

# Grid mode
plotCovariance(fasta, known, grid = TRUE, text = FALSE, cex = 0.8)

# Global style and nucleotide colouring
plotCovariance(fasta, known, grid = TRUE, text = FALSE, base.colour = TRUE)

# Styling indivual helices with styling columns
known$col <- c("red", "blue")
plotCovariance(fasta, known, lwd = 2, cex = 0.8)

# Use in combination with colourBy functions
cov <- colourByCovariation(known, fasta, get = TRUE)
plotCovariance(fasta, cov)
legend("topleft", legend = attr(cov, "legend"),
    fill = attr(cov, "fill"), title = "Covariation")
```

## Create Blank Plot *Create a blank plotting canvas*

### Description

Creates a blank plotting canvas with the given dimensions, along with functions to find best values for the canvas dimensions.

### Usage

```
blankPlot(width, top, bottom, pad = c(0, 0, 0, 0), scale = TRUE,
    scale.lwd = 1, scale.col = "#DDDDDD", scale.cex = 1, debug = FALSE,
    png = NA, pdf = NA, factor = ifelse(!is.na(png), 8, 1/9),
    no.par = FALSE, asp = 1,...)
maxHeight(helix)
```

### Arguments

| | |
|---|---|
| `width` | A number indicating the horizontal width of the blank plot. |
| `top`, `bottom` | The maximum and minimum values vertically to be displayed in the plot. |
| `pad` | An array of 4 integers, specifying the pixels of whitespace to pad beyond the dimensions given by top, bottom, and width. Four number corresponding to padding on the bottom, left, top and right, respectively. Default is c(0, 0, 0, 0). |
| `scale` | If TRUE, inserts a scale on the plot. |
| `scale.lwd`, `scale.col`, `scale.cex` | Allows manual modification of the scale's line width and colour, respectively. |
| `png`, `pdf` | If one or the other is set to a filename, a file in png or pdf format will be produced respectively. If both are set to non-NA values, `png` will have priority. |
| `factor` | The scaling factor used to produce plots of png or pdf format. Should be set so after multiplication of the `top`, `bot`, etc arguments, good document dimension in pixels with `png` and inches for `pdf` will be produced. |
| `debug` | If TRUE, frames the boundaries of the intended plotting space in red, used to determine if inputs produce expected output area. Also outputs to STDIN dimensions of the plot. |
| `no.par` | Suppresses the internal call to `par` in the function if set to TRUE, useful for using `par` arguments such as `mfrow`, etc. |
| `asp` | Controls and aspect ratio of the plot, defaultly set to 1, set to NA to disable completely. |
| `...` | Additional arguments passed to `par` when `no.par` is FALSE, common ones include 'lwd', 'col', 'cex' for line width, line colour, and text size, respectively. `help('par')` for more. When `no.par` is set to TRUE, this option does nothing, and manually calling `par` is required prior to the calling of this function. |
| `helix` | A helix data.frame |

### Details

blankPlot creates a blank plot with the given dimensions, with minimal margins around the plot and no axis or labels. If more control is required, using plot directly would be more efficient.

maxHeight returns the height that the highest helix would require, and can be used to determine top and bottom for blankPlot.

### Value

maxHeight returns a numeric integer.

### Author(s)

Daniel Lai

### See Also

plotHelix

### Examples

```
# Create helix and obtain height
helix <- as.helix(data.frame(1, 37, 12, 0.5))
height <- maxHeight(helix)
print(height)

# Use height to create properly sized plot
width <- attr(helix, "length")
blankPlot(width, height, 0)

# Add helix to plot
plotHelix(helix, add = TRUE)
```

---

## Example Data — *Helices predicted by TRANSAT with p-values*

---

### Description

This data set contains two sets of helices and a multiple sequence alignment. The two sets of helices are helices and known which are helices predicted to occur for RNA sequence RF00458 by the program TRANSAT, and experimentally proposed structure of the same sequence, respectively. fasta is the seed homologues for the multiple sequence alignment obtained from the RFAM database.

### Usage

```
data(helix)
```

### Format

`helix` and `known` are 4 column data frames, where columns **i** and **j** denote the left-most and right-most basepairs, the **length** is the number of *consecutive* basepairs the helix contains, and the **value** is assigned to each helix on a row.

`fasta` is an array of named characters of length 7.

### Value

`fasta` is an array of strings, `helix` and `known` are data.frames in "helix" format.

### References

Wiebe NJ, Meyer IM. (2010) *TRANSAT– method for detecting the conserved helices of functional RNA structures, including transient, pseudo-knotted and alternative structures*. PLoS Comput Biol. 6(6):e1000823.

Gardner PP, Daub J, Tate J, Moore BL, Osuch IH, Griffiths-Jones S, Finn RD, Nawrocki EP, Kolbe DL, Eddy SR, Bateman A. (2011) *Rfam: Wikipedia, clans and the "decimal" release*. Nucleic Acids Res. 39(Database issue):D141-5.

---

## `Find Unknotted Groups` *Partition basepairs into unknotted groups*

---

### Description

Breaks down input helices into basepairs, and assigns each basepair to a numbered group such that basepairs in each group are non-pseudoknotted relative to all other basepairs within the same group.

The algorithm is greedy and thus will *not* find the best combination of basepairs to minimize the number of groups.

### Usage

```
unknottedGroups(helix)
```

### Arguments

| | |
|---|---|
| `helix` | A helix data.frame. |

### Value

An array of integers dictating the groups of each helix. Will only correspond to the input helix structure if the input had helices of length 1 (e.g. output of `expandHelix`).

### Author(s)

Daniel Lai

### See Also

colourByUnknottedGroups

expandHelix

### Examples

```
data(helix)
known$group <- unknottedGroups(known)
print(known)
```

---

## Helix Type Filters *Logical filters of helix by type*

---

### Description

Given a helix data frame, checks if helices are conflicting, duplicating, or overlapping, and returns an array of numeric values, where 0 is FALSE and 1 is TRUE. Values in between 0 and 1 occur when a single helix has multiple basepairs with different values, the number observed in this case is the mean of the basepair values within the helix. See details for exact definition of the three types of events.

### Usage

```
isConflictingHelix(helix)
isDuplicatingHelix(helix)
isOverlappingHelix(helix, query)
```

### Arguments

| | |
|---|---|
| helix | A helix data frame |
| query | For isOverlappingHelix, a helix data structure against which helix will be checked for overlap against. |

### Details

Helices of length greater than 1 are internally expanded into basepairs of length 1, after which the following conditions are evaluated:

A **conflicting** basepair is one where at least one of its two positions is used by either end of another basepair.

A **duplicating** basepair is one where both of its positions are used by both ends of another basepair.

An **overlapping** basepair is one in helix where both of its positions are used by both ends of another basepair in the query structure.

In the case of *conflicting* and *duplicating* basepairs, for a set of basepairs that satisfies this condition, the basepair situation highest on the data frame will be exempt from the condition. i.e. Say 5 basepairs are all duplicates of each other, the top 1 will return FALSE, while the bottom 4 will

return TRUE. This assumes some significant meaning to the ordering of rows prior to using this function. This is to be used with `which` to filter out basepairs that satisfy these conditions, leaving a set of basepairs free of these events.

If the original input had helices greater than length 1, then after applying all of the above, TRUE is treated as 1, FALSE as 0, and the average of values from each basepair is taken as the value for the helix in question.

## Value

Returns an array of numerics corresponding to each row of `helix`, giving the average conditional status of the helix, where 0 signifying all basepairs are FALSE, and 1 where all basepairs are TRUE.

## Author(s)

Daniel Lai

## Examples

```
data(helix)

conflicting <- isConflictingHelix(helix)
duplicating <- isDuplicatingHelix(helix)

# Nonsensical covariation plot
plotCovariance(fasta, helix)

# Plot nonconflicting helices
plotCovariance(fasta, helix[(!conflicting & !duplicating), ])

# Similar result
plotCovariance(fasta, helix, conflict.col = "lightgrey")
```

---

`Log10 Space Operations`

*Log base 10 sequence, floor and ceiling*

---

## Description

Sequence, floor and ceiling operations in log 10 space.

## Usage

```
logseq(from, to, length.out)

logfloor(x)

logceiling(x)
```

### Arguments

| | |
|---|---|
| `from`, `to` | Positive non-zero values to start and end sequence, respectively. |
| `length.out` | The number of elements the resulting sequence should containg. If absent, function will attempt to generate numbers factors of 10 apart. |
| `x` | A value to round. |

### Value

`logseq` returns an array numbers evenly distanced in log10-space.

`logfloor` and `logceiling` return a value that is 10 raised to an integer number.

### Author(s)

Daniel Lai

### Examples

```
logseq(1e-10, 1e3)
logseq(1e-10, 1e3, length.out = 10)
logceiling(2.13e-6)
logfloor(2.13e-6)
```

---

## `Plot Helix Structures` *Plots helices in arc diagram*

---

### Description

Plots a helix data frame as an arc diagram, with styling possible with properly named additional columns on the data frame.

### Usage

```
plotHelix(helix, x = 0, y = 0, flip = FALSE, line = FALSE, arrow = FALSE,
    add = FALSE, shape = "circle", ...)
plotDoubleHelix(top, bot, line = TRUE, arrow = FALSE, add = FALSE, ...)
plotOverlapHelix(predict, known, miss = "black", line = TRUE,
    arrow = FALSE, add = FALSE, overlap.cutoff = 1, ...)
plotArcs(i, j, length, x = 0, y = 0, flip = FALSE, shape = "circle", ...)
plotArc(i, j, x = 0, y = 0, flip = FALSE, shape = "circle", ...)
```

### Arguments

`helix`, `top`, `bot`, `predict`, `known`
: Helix data.frames, with the four mandatory columns. Any other column will be considered a styling column, and will be used for styling the helix. See `example` for styling usage. See `Details` for exact usage of each helix.

`x`, `y`
: The coordinate of the left bottom corner of the plot, useful for manually positioning figure elements.

`flip`
: If TRUE, flips the arcs upside down about the y-axis.

`line`
: If TRUE, a horizontal line representing the sequence is plotted.

`arrow`
: If TRUE, an arrow is played on the right end of the line.

`add`
: If TRUE, graphical elements are added to the active plot device, else a new plot device is created for the plot.

`shape`
: One of "circle", "triangle", or "square", specifying the shape of the arcs.

`miss`
: The colour for unpredicted arcs in overlapping diagrams, see `details` for more information.

`overlap.cutoff`
: Decimal between 0 and 1 indicating the percentage of basepairs within a helix that have to be overlapping for the entire helix to count as overlapping. Default is 1, or 100

`i`, `j`
: The starting and ending position of the arc along the x-axis

`length`
: The total number of arcs to draw by incrementing *i* and decrementing *j*. Used to draw helices.

`...`
: Any additional parameters passed to `par`

### Details

`plotHelix` creates a arc diagram with all arcs on top, `plotDoubleHelix` creates a diagram with arcs on the top and bottom. `plotOverlapHelix` is slight trickier, and given two structures `predict` and `known`, plots the predicted helices that are known on top, predicted helices that are not known on the bottom, and finally plots unpredicted helices on top in the colour defined by `miss`.

`plotArc` and `plotArcs` are the core functions that make everything work, and may be used for extreme fine-tuning and customization.

### Value

Not intended to return a value, will plot to GUI or file if specific.

### Author(s)

Daniel Lai

### See Also

`colourByCount`

## Examples

```
data(helix)

# Plot helix plain
plotHelix(known)

# Apply global appearance options
plotHelix(known, line = TRUE, arrow = TRUE, col = "blue", lwd = 1.5)

# Add extra column with styling options
known$lty <- 1:4
known$lwd <- 1:2
known$col <- c(rgb(1, 0, 0), "orange", "yellow", "#00FF00", 4, "purple")
plotHelix(known)

# Manually colour helices according to value
helix$col <- "red"
helix$col[which(helix$value < 1e-3)] <- "orange"
helix$col[which(helix$value < 1e-4)] <- "green"
helix$col[which(helix$value < 1e-5)] <- "blue"
plotHelix(helix)

# Automatically creating a similar plot with legend
coloured <- colourByValue(helix, log = TRUE, get = TRUE)
plotHelix(coloured, line = TRUE, arrow = TRUE)
legend("topleft", legend = attr(coloured, "legend"),
    fill = attr(coloured, "fill"), title = "P-value", text.col = "black")

# Plot both helices with styles
plotDoubleHelix(helix, known)

# Overlap helix
plotOverlapHelix(helix, known)
```

---

# Read Structure File *Read secondary structure file*

---

## Description

Reads in secondary structure text files into a helix data frame.

## Usage

```
readHelix(file)
readConnect(file)
readVienna(file, palette = NA)
readBpseq(file)
```

### Arguments

| | |
|---|---|
| `file` | A text file in connect format, see details for format specifications. |
| `palette` | Used to colour basepairs by bracket type. See `viennaToHelix` for more details. |

### Details

**Helix**: Files start with a header line beginning with # followed by the sequence length, followed by a four-column tab-delimited table (with column names), where each row corresponds to a helix in the structure. The four columns are *i* and *j* for the left-most and right-most basepair positions respectively, the *length* of the helix (converging inwards from i and j, and finally an arbitrary *value* assigned to the helix.

**Vienna**: Dot-bracket notation from Vienna package programs, where each structure consists of matched brackets for basepairs and periods for unbased pairs. Valid brackets are (, , [, <, A, B, C, D matched with ), , ], >, a, b, c, d, respectively. An energy value can be appended to the end of any dot-bracket structure. The function will accept slight variations of the format, including those with FASTA-like headers (in which case line breaks are allows), and those without FASTA-like headers (in which case line breaks are NOT allowed), with both types allowing for a preceding (NOT following) nucleotide sequence for the structure. Multiple entries *of the same length* may be in a single file, which will be returned as a single helix structure, with respectively energy values (if specified).

**Connect**: Output from mfold and other programs, this format is expected to be a text file beginning with a header line that starts with the sequence length, with an optional Energy/dG value, followed by a six-column tab-delimited table where columns 1 and 5 denote the position that are basepaired (unpaired when column 5 is 0). Other columns are ignored, but for completeness, column 2 is the nucleotide, column 3 and 4 are the positions of the bases left and right of the base specified in column 1 respectively (with 0 denoting non-existance), and column 6 a copy of column 1. Multiple entries *of the same length* may be in a single file, which will be returned as a single helix structure. All helices will be assigned the energy value extracted from their respective structure header lines.

**Bpseq**: Format used by the Gutell Lab's Comparative RNA Website. The file may optionally begin several header lines (e.g. Filename, Organism, Accession, etc.), followed by a 3-column tab-delimited table for the structure, where column 1 is the base position, base 2 is the nucleotide base, and column 3 is the paired position (0 if unpaired). Certain pieces of header information will be parsed and returned as attributes of the output data frame. Multiple structures can be within a single file, returned as a single helix data frame, with attributes set to those of the first entry.

### Value

Returns a helix format data frame.

### Author(s)

Daniel Lai, Jeff Proctor

### Examples

```
file <- system.file("extdata", "helix.txt", package = "R4RNA")
helix <- readHelix(file)
head(helix)
```

```
file <- system.file("extdata", "connect.txt", package = "R4RNA")
connect <- readConnect(file)
head(connect)
message("Note connect data assigns structure energy level to all basepairs")

file <- system.file("extdata", "vienna.txt", package = "R4RNA")
vienna <- readVienna(file)
head(vienna)
message("Note vienna data assigns structure energy level to all basepairs")

file <- system.file("extdata", "bpseq.txt", package = "R4RNA")
bpseq <- readBpseq(file)
head(bpseq)
message("Note bpseq data has no value assigned to basepairs")
```

---

`Structure Mismatch Score`

*Scores how a basepair structure fits a sequence*

---

## Description

Calculates a score that indicates how badly a set of basepairs (i.e. a secondary structure) fits with a sequence. A perfect fit is a structure where all basepairs form valid basepairs (A:U, G:C, G:U, and equivalents) and has a score of 0. Each basepair that forms a non-canonical pairing or pairs to gaps increases the score by 1, and each base-pair with a single-sided gap increases the score by 2.

## Usage

```
structureMismatchScore(msa, helix, one.gap.penalty = 2, two.gap.penalty = 2,
              invalid.penalty = 1)
```

## Arguments

| | |
|---|---|
| `msa` | A multiple sequence alignment. Can be either a `Biostrings` `XStringSet` object or a named array of strings like ones obtained from converting XStringSet with `as.character`. |
| `helix` | A helix data.frame |
| `one.gap.penalty` | Penalty score for basepairs with one of the bases being a gap |
| `two.gap.penalty` | Penalty score for basepairs with both bases being a gaps |
| `invalid.penalty` | Penalty score for non-canonical basepairs |

## Value

Returns an array of mismatch scores.

### Author(s)

Jeff Proctor, Daniel Lai

### Examples

```
data(helix)
mismatch <- structureMismatchScore(fasta, known)

# Sort by increasing mismatch
sorted_fasta <- fasta[order(mismatch)]
```

---

## Write Helix — *Write out a helix data frame into a text file*

---

### Description

Write out a helix data frame into a text file into the four-column tab-delimited format with proper header and column names.

### Usage

```
writeHelix(helix, file = stdout())
```

### Arguments

| | |
|---|---|
| `helix` | A helix data frame. |
| `file` | A character string pointing to a file path, or a file connection. Defaults to the console. |

### Value

No value returned, will write to STDOUT or specific file location.

### Author(s)

Daniel Lai

### Examples

```
# Create helix data frame
helix <- data.frame(2, 8, 3, 0.5)
helix[2, ] <- c(5, 15, 4, -0.5)
helix <- as.helix(helix)

writeHelix(helix)
```

# Index