Testing for circularity

Contents

Prerequisites

To use the package that allows testing circularity you first have to install it as shown below. To run the code in the box below you can just press the green play button in the left top corner of the code box. You might be asked in the console below this window to update the included packages. Afterwards it might take a while to install all 3 packages.

```
install.packages("Rcpp", repos = "https://packages.othr.de/cran/" )
install.packages("devtools", repos = "https://packages.othr.de/cran/")
devtools::install_github("StarmanMartin/GCATR", ref = "Testing")
```

Analyse whether given code is present in sequence

Now that you have installed all prerequisistes you can try out the GCATR package.

You should define the sequence seq and the code for which the sequence is checked If you define the string as in the following example, you have to provide the tuple length. On top of that you can provide the frame in which the sequence is checked. If you provide no value for the frame, no frameshift will be done.

To execute the following code snippet you just have to press the little green arrow on the right side of the code box.

```
seq <- "ACGTCGCGACGTACGACGTCG"
code = "ACGTCG"
analysis <- GCATR::find_and_analysis_code_in_sequence(seq, "ACGTCG", tuple_length=3)
print(analysis)
```

```
## $word
## [1] "ACG" "TCG" "ACG" "ACG" "TCG"
##
## $idx_list
## [1]  0  3 12 15 18
##
## $rest
## [1] "CGACGT"
##
## $parts
## [1] ""          "ACGTCG"    "CGACGT"    "ACGACGTCG"
##
## $longest_match
## [1] 9
##
## $total_match_in_percent
## [1] 71.42857
```

Ways to provide input

Apart from the way seen above you can also provide the input as follows:

```
code = "ACG TCG"
code = c("ACG", "TCG")
```

Generate a circular code by min value

This function allows you to generate a circular code by providing a tuple length and a input alphabet. It uses the first codon of each individual equivalence class.

You can play around with the functionality with following code snippet.

```
alphabet <- c("A","C","G","U")
tuple_length <- 3
min_code <- GCATR::generate_code_by_min_value(alphabet, tuple_length)
print(min_code)
```

```
##  [1] "AAC" "AAG" "AAU" "ACC" "ACG" "ACU" "AGC" "AGG" "AGU" "AUC" "AUG" "AUU"
## [13] "CCG" "CCU" "CGG" "CGU" "CUG" "CUU" "GGU" "GUU"
```

Shift tuples by a specific value

This function allows you to apply a shift to given tuples. The code can be provided as seen in the section that states the way to provide the input.

The following snippet demonstrates how to apply a 1 or 2 shift.

```
shifted_code <- GCATR::shift_tuples(1, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGC" "GAG" "GAU"
```

```
shifted_code <- GCATR::shift_tuples(2, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGG" "AUG" "GCA"
```

Shift tuples by a specific value

This function allows you to apply a shift to given tuples. The code can be provided as seen in the section that states the way to provide the input.

The following snippet demonstrates how to apply a 1 or 2 shift.

```
shifted_code <- GCATR::shift_tuples(1, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGC" "GAG" "GAU"
```

```
shifted_code <- GCATR::shift_tuples(2, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGG" "AUG" "GCA"
```

Transform code

This function enables you to perform custom transformations on provided code. You may specify how letters are translated. In order to do so you have to create to strings as so.

```r
from = "AGCT"
to = "GCAT"
```

What this does is it determines how each individual letter is transformed In our example the A from the first string gets transformed to a G. The table below shows how the transformation is done.

```r
library(knitr)
values <- matrix(c("A", "G", "C", "T", "G", "C", "A", "T"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 1: Value Transformation

from	to
A	G
G	C
C	A
T	T

The function call is done like this. The code can be provided as seen in the section that states the way to provide the input.

```r
from = "AGCT"
to = "GCAT"
transformed_code <- GCATR::code_transform_tuples(from, to, "ACGTCT", tuple_length = 3)
print(transformed_code)
```

```
## [1] "GAC" "TAT"
```

Transform code with self-complementarity preserving permuations

There is eight transformations that preserve self-complementarity. The most simple one is the identity transformation, that does not permute anything The code can be provided as seen in the section that states the way to provide the input..

```r
transformation_type <- "Id"
code <- "ACGTCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code, tuple_length = 3)
print(transformed_code)
```

```
## [1] "ACG" "TCT"
```

Next up is the KM transformation. It transforms the code as seen in the table below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "C", "G", "A", "T"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 2: Value Transformation

from	to
A	C
T	G
C	A
G	T

Call it identically as the Identity transformation just changing up the transformation type.

```
transformation_type <- "KM"
code <- "ACG TCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CAT" "GAG"
```

The YR transformation transforms the code as seen below.

```
library(knitr)
values <- matrix(c("A", "T", "C", "G", "G", "C", "T", "A"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 3: Value Transformation

from	to
A	G
T	C
C	T
G	A

Call it identically as the other transformations just changing up the transformation type.

```
transformation_type <- "YR"
code <- c("ACG","TCT")
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CTC" "GTA"
```

The AT transformation transforms the code as seen below.

```
library(knitr)
values <- matrix(c("A", "T", "C", "G", "T", "A", "C", "G"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 4: Value Transformation

from	to
A	T
T	A
C	C
G	G

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "AT"
code <- "ACGTCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code, tuple_length = 3)
print(transformed_code)
```

```
## [1] "ACA" "TCG"
```

The CG transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "A", "T", "G", "C"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 5: Value Transformation

from	to
A	A
T	T
C	G
G	C

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "CG"
code <- c("ACG","TCT")
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "AGC" "TGT"
```

The ACTG transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "C", "G", "T", "A"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 6: Value Transformation

from	to
A	C
T	G
C	T
G	A

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "ACTG"
code <- "ACG TCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CTA" "GTG"
```

The AGTC transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "G", "C", "A", "T"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 7: Value Transformation

from	to
A	G
T	C
C	A
G	T

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "AGTC"
code = c("ACG", "TCT")
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CAC" "GAT"
```

Prerequisites

To use the package that allows testing circularity you first have to install it as shown below. To run the code in the box below you can just press the green play button in the left top corner of the code box. You might be asked in the console below this window to update the included packages. Afterwards it might take a while to install all 3 packages.

```r
install.packages("Rcpp", repos = "https://packages.othr.de/cran/" )
install.packages("devtools", repos = "https://packages.othr.de/cran/")
devtools::install_github("StarmanMartin/GCATR", ref = "Testing")
```

Analyse whether given code is present in sequence

Now that you have installed all prerequisistes you can try out the GCATR package.

You should define the sequence seq and the code for which the sequence is checked If you define the string as in the following example, you have to provide the tuple length. On top of that you can provide the frame in which the sequence is checked. If you provide no value for the frame, no frameshift will be done.

To execute the following code snippet you just have to press the little green arrow on the right side of the code box.

```r
seq <- "ACGTCGCGACGTACGACGTCG"
code = "ACGTCG"
analysis <- GCATR::find_and_analysis_code_in_sequence(seq, "ACGTCG", tuple_length=3)
print(analysis)
```

```
## $word
## [1] "ACG" "TCG" "ACG" "ACG" "TCG"
##
## $idx_list
## [1]  0  3 12 15 18
##
## $rest
## [1] "CGACGT"
##
## $parts
## [1] ""          "ACGTCG"    "CGACGT"    "ACGACGTCG"
##
## $longest_match
## [1] 9
##
## $total_match_in_percent
## [1] 71.42857
```

Ways to provide input

Apart from the way seen above you can also provide the input as follows:

```r
code = "ACG TCG"
code = c("ACG", "TCG")
```

Generate a circular code by min value

This function allows you to generate a circular code by providing a tuple length and a input alphabet. It uses the first codon of each individual equivalence class.

You can play around with the functionality with following code snippet.

```
alphabet <- c("A","C","G","U")
tuple_length <- 3
min_code <- GCATR::generate_code_by_min_value(alphabet, tuple_length)
print(min_code)
```

```
##  [1] "AAC" "AAG" "AAU" "ACC" "ACG" "ACU" "AGC" "AGG" "AGU" "AUC" "AUG" "AUU"
## [13] "CCG" "CCU" "CGG" "CGU" "CUG" "CUU" "GGU" "GUU"
```

Shift tuples by a specific value

This function allows you to apply a shift to given tuples. The code can be provided as seen in the section that states the way to provide the input.

The following snippet demonstrates how to apply a 1 or 2 shift.

```
shifted_code <- GCATR::shift_tuples(1, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGC" "GAG" "GAU"
```

```
shifted_code <- GCATR::shift_tuples(2, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGG" "AUG" "GCA"
```

Shift tuples by a specific value

This function allows you to apply a shift to given tuples. The code can be provided as seen in the section that states the way to provide the input.

The following snippet demonstrates how to apply a 1 or 2 shift.

```
shifted_code <- GCATR::shift_tuples(1, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGC" "GAG" "GAU"
```

```
shifted_code <- GCATR::shift_tuples(2, "CAGGGAUGA", tuple_length = 3)
print(shifted_code)
```

```
## [1] "AGG" "AUG" "GCA"
```

Transform code

This function enables you to perform custom transformations on provided code. You may specify how letters are translated. In order to do so you have to create to strings as so.

```
from = "AGCT"
to = "GCAT"
```

What this does is it determines how each individual letter is transformed In our example the A from the first string gets transformed to a G. The table below shows how the transformation is done.

```
library(knitr)
values <- matrix(c("A", "G", "C", "T", "G", "C", "A", "T"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 8: Value Transformation

from	to
A	G
G	C
C	A
T	T

The function call is done like this. The code can be provided as seen in the section that states the way to provide the input.

```
from = "AGCT"
to = "GCAT"
transformed_code <- GCATR::code_transform_tuples(from, to, "ACGTCT", tuple_length = 3)
print(transformed_code)
```

```
## [1] "GAC" "TAT"
```

Transform code with self-complementarity preserving permuations

There is eight transformations that preserve self-complementarity. The most simple one is the identity transformation, that does not permute anything The code can be provided as seen in the section that states the way to provide the input..

```
transformation_type <- "Id"
code <- "ACGTCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code, tuple_length = 3)
print(transformed_code)
```

```
## [1] "ACG" "TCT"
```

Next up is the KM transformation. It transforms the code as seen in the table below.

```
library(knitr)
values <- matrix(c("A", "T", "C", "G", "C", "G", "A", "T"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 9: Value Transformation

from	to
A	C
T	G
C	A
G	T

Call it identically as the Identity transformation just changing up the transformation type.

```r
transformation_type <- "KM"
code <- "ACG TCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CAT" "GAG"
```

The YR transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "G", "C", "T", "A"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 10: Value Transformation

from	to
A	G
T	C
C	T
G	A

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "YR"
code <- c("ACG","TCT")
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CTC" "GTA"
```

The AT transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "T", "A", "C", "G"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 11: Value Transformation

from	to
A	T
T	A
C	C
G	G

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "AT"
code <- "ACGTCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code, tuple_length = 3)
print(transformed_code)
```

```
## [1] "ACA" "TCG"
```

The CG transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "A", "T", "G", "C"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 12: Value Transformation

from	to
A	A
T	T
C	G
G	C

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "CG"
code <- c("ACG","TCT")
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "AGC" "TGT"
```

The ACTG transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "C", "G", "T", "A"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 13: Value Transformation

from	to
A	C
T	G
C	T
G	A

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "ACTG"
code <- "ACG TCT"
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CTA" "GTG"
```

The AGTC transformation transforms the code as seen below.

```r
library(knitr)
values <- matrix(c("A", "T", "C", "G", "G", "C", "A", "T"),byrow = FALSE, nrow = 4)
colnames(values) <- c("from", "to")
kable(values[1:4, ], caption = "Value Transformation")
```

Table 14: Value Transformation

from	to
A	G
T	C
C	A
G	T

Call it identically as the other transformations just changing up the transformation type.

```r
transformation_type <- "AGTC"
code = c("ACG", "TCT")
transformed_code <- GCATR::code_named_transform_tuples(transformation_type, code)
print(transformed_code)
```

```
## [1] "CAC" "GAT"
```