Find Recombinations Among Genomes

Release 0.1.4

Nicolas Maillet

Dec 19, 2020

CONTENTS:

Find Recombinations Among Genomes (FRAGS) is a software dedicated to analyze recombinations in viral genomes.

note FRAGS is tested with Gitlab Ci for the following Python version: 3.7 to 3.8

issues Please use https://gitlab.pasteur.fr/nmaillet/frags

CHAPTER ONE

OVERVIEW

Find Recombinations Among Genomes (FRAGS), is a standalone software dedicated to identify reads coming from recombination events.

FRAGS is a python tool taking fasta/fastq files of reads as input and one or two reference genomes. It then identifies chimeric reads (reads composed of non-adjacent fragsments, coming either from on one or the two references genomes) and potential breakpoints (insert between fragsments). Optionally, breakpoints can then be Blasted again the host genome, if provided.

Main results are in CSV files. Three CSV files contain respectively results for reads that have fragsments coming from the first reference only, the second reference only or both references. Another CSV file contains headers of reads that did not match anywhere. Finally, three files are produced when using Blast: `breakpoints.fasta`, containing the breakpoint portions to Blast, `res_blast.csv` containing the result of Blast on `breakpoints.csv` and `compressed.fasta`, a compressed version of `res_blast.csv` keeping only results of the best e-value/bit-score for each input sequences and produce a fasta-like file.

FRAGS follows the standards for software development with continuous integration on Gitlab (https://gitlab.pasteur.fr/nmaillet/frags) and automatic on-line documentation (https://frags.readthedocs.io/en/latest/).

CHAPTER TWO

INSTALLATION

In order to install FRAGS, you can use **pip**:

```
pip3 install frags
```

This command installs FRAGS and its Python dependencies.

CHAPTER

THREE

USAGE

From the command line:

```
frags --help
```

CHAPTER

FOUR

USER AND DEVELOPER GUIDES

4.1 User Guide

4.1.1 Overview

You can run **Find Recombinations Among Genomes** using the standalone version called:

```
frags
```

You can obtain help by using:

```
frags --help
```

4.1.2 Installation

4.1.2.1 From pip

The suggested way of installing the latest **FRAGS** version is through **pip**:

```
pip3 install frags
```

Then you can use:

```
frags --help
```

4.1.2.2 From source code

FRAGS is coded in Python. To manually install it from source, get the source and install **FRAGS** using:

```
git clone https://gitlab.pasteur.fr/nmaillet/frags/
cd frags
python setup.py install
```

4.1.2.3 Using without installation

You can download the source code from Pasteur's **Gitlab**: https://gitlab.pasteur.fr/nmaillet/frags/.

In order to directly run **FRAGS** from source, you need to copy file `tests/context.py` into `frags` folder.

Then, uncomment line 14 of `frags/FindRecombinationsAmongGenomes.py`. Modify:

```
#from context import frags
```

To:

```
from context import frags
```

Then, from the main **FRAGS** directory, use:

```
python3 frags/FindRecombinationsAmongGenomes.py --help
```

Warning: Using without installation is not recommended, as you need all requirements of *requirements.txt* installed locally and you may encounter issues with Sphinx autodoc or other unwanted behaviors.

4.1.3 Classical use

Here are some typical examples of **FRAGS** usage.

4.1.3.1 Getting help

To access build-in help, use:

```
frags --help
```

4.1.3.2 Find recombinations with one reference genome

To find all recombinations of a reads file, compare to a single reference genome, use:

```
frags -i read_file.fasta -r ref_file.fasta
```

4.1.3.3 Find recombinations with two reference genomes

To find all recombinations of a reads file, compare to two reference genomes, use:

```
frags -i read_file.fasta -r ref_file.fasta ref_file2.fasta
```

4.1.3.4 Using multi fasta/fastq files

To find all recombinations of several reads files, use:

```
frags -i read_file1.fasta read_file2.fasta -r ref_file.fasta ref_file2.fasta
```

4.1.3.5 Using Blast to analyze breakpoints

To perform a Blast on identified breakpoints against the host genome, use:

```
frags -i read_file.fasta -r ref_file.fasta -b -t host_file.fasta
```

See *Blast analyze* for more information.

4.1.4 Working principle of FRAGS

FRAGS is a python tool taking fasta/fastq files of reads as input and one or two reference genomes. It then identifies chimeric reads (reads composed of non-adjacent fragments (or matches), coming either from on one or the two references genomes) and potential breakpoints (insert between matches). Optionally, breakpoints can then be Blasted again the host genome, if provided.

4.1.4.1 K-mer concept

Technically, FRAGS identifies similarities (matches) between a fragment of a read and a reference using k-mers. A k-mer is a small DNA part of the read. Each reads are decomposed in overlapping k-mers and each k-mer is searched in references. The smaller a k-mer is, the most probable it will be found in the reference. But using really small k-mers (i.e. < 20) leads to identifying it in the reference 'by chance'. K-mers size must usually be greater than 25 to ensure robust results.

Note that the search is performed using Aho-Corasick algorithm, enabling a fast and error-free identifications. K-mers that are matching in the reference and are contiguous (modulo *Gap length*) in both the reference and the read are merged together. Due to Aho-Corasick algorithm, **each k-mers of a reference should be unique**. If a k-mer is present more than one time in a reference, **only the first occurrence of this k-mer is used in upcoming the comparisons**. Fortunately, this situation is most of the time encountered in low complexity part of the genome such as poly(A) tail or GA-dinucleotide repeats.

4.1.4.2 Matches

A **match** is a fragment of the read that is almost identical to a fragment of a reference (modulo *Gap length*). All matches are written in result files, with there starting position and size (see `outputfolder` for more details).

Note that each match is unique, and that no match can be included in another one. Accordingly, when a match is found on a read, if the exact same fragment was already found elsewhere (different position of the same reference or into the other reference), only the first match is kept.

A read composed as: `XXXXXXXXXXXXyyyyyyyyyyyyyyyyyy`, with `X` matching on reference 1 and 2, and `y` not matching, will produce the result:

`XXXXXXXXXXXX` is matching on reference 1
`yyyyyyyyyyyyyyyyyy` is not matching

In the same way, if a match is included into a bigger one, only the bigest is kept.

A read composed as: `XXXXxxxxxXXXXyyyyyyyyyyyyyyyyyy`, with `xxxxx` matching on reference 1, `XXXXxxxxxXXXX` on reference 2, and `y` not matching, will produce the result:

`XXXXxxxxxXXXX` is matching on reference 2
`yyyyyyyyyyyyyyyyyy` is not matching

4.1.4.3 Breakpoints

A **breakpoints** is what stand between two matches, where a recombination probably occurs. Size of breakpoints can be of three types.

Empty breakpoint

A breakpoint of size 0 occurs when two contiguous fragments of a read match at different locations.

A read composed as: `XXXXXXXXXXXXyyyyyyyyyyyyyyyyyy`, with `XXXXXXXXXXXX` of size 12 (from nucleotides 1 to 12) matching on reference 1 and `yyyyyyyyyyyyyyyyyy` on reference 2 (or at a different location in ref1) will produce the following breakpoint:

12:0 i.e. breakpoint start after the nucleotide at position 12, for a size of 0 nucleotide.

Positive breakpoint

A breakpoint of positive size occurs when two non contiguous fragments of a read match.

A read composed as: `XXXXXXXXXXXXyyyyyyyZZZZZZZZZZ`, with `XXXXXXXXXXXX` of size 12 (from 1 to 12) matching on a reference, `yyyyyyy` (7 nucleotides) not matching on any reference and `ZZZZZZZZZZ` matching on a reference will produce the following breakpoint:

12:7 i.e. breakpoint start after the nucleotide at position 12 and is composed of the 7 **following** nucleotides.

Negative breakpoint

A breakpoint of negative size occurs when two overlapping fragments of a read match at different locations.

A read composed as: `XXXXXXXXXXXXyyyyyyyZZZZZZZZZZ`, with `XXXXXXXXXXXXyyyyyyy` of size 19 (from 1 to 19) matching at a location and `yyyyyyyZZZZZZZZZZ` (17 nucleotides) matching at a different location will produce the following breakpoint:

17:-7 i.e. breakpoint start after the nucleotide at position 17 and is composed of the 7 **preceding** nucleotides.

Breakpoints can be further investigated using Blast.

4.1.4.4 Blast analyze

Breakpoints can then be Blasted against the host genome where the recombination append. To do so, the host genome must be inputted in FRAGS (-t option) and Blast option switched on (-b option).

Then, each breakpoint of at least -m option nucleotides is locally Blasted against the host genome.

Three files are produced when using Blast: `breakpoints.fasta`, `res_blast.csv` and `compressed.fasta`. See *Output of FRAGS* for more details about results files.

Warning: Blast in command line must be available on your computer. See Blast installation page.

Warning: A Blast database is required to perform local Blast. This database must be at the same location than the host genome file. If not, the database will be automatically created at this location.

4.1.4.5 Output of FRAGS

Main results are in CSV files. Three CSV files contain respectively results for reads that have matches coming from the first reference only, the second reference only or both references. Another CSV file contains headers of reads that did not match anywhere. Finally, three files are produced regarding Blast uses: `breakpoints.fasta`, containing the breakpoint fragments to Blast, `res_blast.csv` containing the result of Blast (if required) on `breakpoints.fasta` and `compressed.fasta`, a compressed version of `res_blast.csv` keeping only results of the best e-value/bit-score for each input sequences and produces a fasta-like file.

Note: `breakpoints.fasta` and `compressed.fasta` are fasta files, regardless of input files being fasta or fatsq files. The starting character of headers in these files are always >.

In `compressed.fasta`, headers are created as follow: `OriginalHeader_IdOfBreakpoint|E-value|Bit-score`. `IdOfBreakpoint` is the identifier of the breakpoint in the read. If there is three breakpoints in a read, identifiers will be `1`, `2` and `3`. `E-value` and `Bit-score` are respectively the e-value and the bit-score of the best(s) hit(s) for this breakpoint. The sequence line is composed of all different Blast hits with the same e-value and bit-score, separated by tabulations. Tabulation and `|` are configurable through *CSV configuration of output files*.

Warning: Because of Blast limitation, headers of input reads must not contain any spaces. FRAGS will replace all spaces in headers by _ symbol.

The CSV file are composed of 9 columns:

- `Read_header` contains header of the input read that matches (see warning below)
- `Reverse_complement` indicates if all matches of this read are in normal strand (0), in revers complement (1) or some in normal strand and some in reverse complement (2)
- `Number_of_breakpoints` indicates the total number of breakpoints identified of this read
- `Breakpoints_positions` indicates the position and size of each identified breakpoint of this read (see *Nomenclature of positions*)
- `Matches_read_positions` indicates the position and size (in the read) of each identified matches of this read (see *Nomenclature of positions*). Each match is preceded by the strand in parentheses, i.e. `(-)` for reverse

complement or `(+)` for normal strand, and the id in parentheses of the reference where it matches, i.e. `(1)` for the first inputted reference genome or `(2)` for the second inputted reference genome

- `Matches_ref_positions` indicates the position and size (in the ref) of each identified matches of this read (see *Nomenclature of positions*). Each match is preceded by the id in parentheses of the reference where it matches, i.e. `(1)` for the first inputted reference genome or `(2)` for the second inputted reference genome
- `Matches_size` indicates all sizes of all matches of this read
- `Insertions` indicates sizes of insertions that append in matches of this read. Note that a single match can have several insertions. They are then separated by `:` (configurable through *CSV configuration of output files*).
- `Blast_results_breakpoints` indicates which breakpoints of this read have a Blast hit

Warning: Because of Blast limitation, headers of input reads must not contain any spaces. FRAGS will replace all spaces in headers by _ symbol.

Nomenclature of positions

The nomenclature used to represent a match or a breakpoint is the following: `X:Y` where `X` is the index **before** the starting nucleotide, `Y` the size of the match/breakpoint and `:` the configurable separator (see *CSV configuration of output files* for configuration).

A read composed as: `XXXXXXXXXXXXyyyyyyyZZZZZZZZZZ`, with `XXXXXXXXXXXX` of size 12 (from 1 to 12) matching on a reference, `yyyyyyy` of size 7 (from 13 to 19) not matching on any reference and `ZZZZZZZZZZ` of size 10 (from 20 to 29) matching on a reference will produce the following matches/breakpoint:

`Match 0:12` i.e. match that start after the nucleotide at position 0 and is composed of the 12 following nucleotides.

`Breakpoint 12:7` i.e. breakpoint that start after the nucleotide at position 12 and is composed of the 7 following nucleotides.

`Match 19:10` i.e. match that start after the nucleotide at position 19 and is composed of the 10 following nucleotides.

Example of a CSV result line

`>read_name 1 2 59:20|114:47 (-)(1)25:34|(-)(2)79:35|(-)(2)161:39 (1)534:34|(2)522:35|(2)336:39 34|35|39 0|3|0 2` The read `read_name` matches only in reverse complement (`1`). It has `2` breakpoints, from nucleotide 60 to 79 (`59:20`, start at nucleotide 60, size of 20 nucleotide) and from nucleotide 115 to 161 (`114:47`).

The first match is in reverse complement (`(-)`) on the first reference genome (`(1)`) from positions 26 to 59 (`25:34`).

The second match is in reverse complement (`(-)`) on the second reference genome (`(2)`) from positions 80 to 114 (`79:35`).

The third match is in reverse complement (`(-)`) on the second reference genome (`(2)`) from positions 162 to 200 (`161:39`).

The first match is on the first reference genome (`(1)`) from positions 535 to 568 (`534:34`).

The second match is on the second reference genome (`(2)`) from positions 523 to 557 (`522:35`).

The third match is on the second reference genome (`(2)`) from positions 337 to 375 (`336:39`).

The first match has a length of 34 nucleotides, the second 35 and the third 39 (`34|35|39`)

The first match has no insertion, the second one an insertion of 2 nucleotides, and the third one, 0 (`0|3|0`)

The second breakpoint had a hit with Blast (`2`)

Note: In CSV files, symbols `:`, `|` and tabulation are configurable. See *CSV configuration of output files* for this.

4.1.5 Options

Here are all available options in **FRAGS**:

4.1.5.1 Default options

-h, –help: Show this help message and exit.

-i, –inputfiles: Input reads files. See *Input files* for more information.

-r, –reffiles: Input reference files. See *Reference files* for more information.

-o, –outputfolder: Output folder containing result files. See *Output of FRAGS* for more information.

-k, –kmer: K-mer length of the search (default: 30). See *K-mers length* for more information.

-g, –gap: Gap length to consider not contiguous hits as contiguous (default: 10). See *Gap length* for more information.

-p, –processes: Number of parallel processes to use (default: 1). See *Multi-process* for more information.

-q, –quiet: No standard output, only error(s).

-v, –verbose: Increase output verbosity. See *Verbosity* for more information.

–version: Show program's version number and exit.

4.1.5.2 Blast options

See *Blast analyze* for more detailed informations.

-b, –blast: Use Blast to analyze breakpoints greater than -m/–minsizeblast argument.

-t, –host: Host genome file. Required if -b/–blast argument is used. Note: a Blast database will be created at this location.

-m, –minsizeblast: Minimum size of breakpoint to Blast (default: 20).

4.1.5.3 CSV options

See *CSV configuration of output files* for more detailed informations.

-s, –posizesep: Separator between position and size (default: :).

-f, –fieldsep: Field separator inside columns (default: |).

-c, –csvsep: Column separator (default: \t).

4.1.5.4 Detailed options

Input files

Input files should be fasta or fastq files. They can be compressed in gzip.

To input several files, use:

```
frags -i read_file1.fastq read_file2.fasta read_file3.fastq.gz ...
```

This will be equivalent to first merge all input files into a big one.

Reference files

Reference genomes files should be fasta or fastq files. They can be compressed in gzip.

You can input one or two reference genomes.

```
frags -i ... -r ref1.fastq ref2.fasta.gz ...
```

Note: Genomes must be composed of a single sequence. If a reference file is composed of more than one sequence, only the first one will be used.

K-mers length

K-mers size used to find similarities (matches) between a read and a reference. Note that k-mers size should ideally be greater than 25 to ensure robust results. See *K-mer concept* to understand how similarities are found and used.

Gap length

The gap option allows none contiguous sub-parts of a read to be considered as contiguous, if the size between the different parts are smaller than –gap option.

A read composed as: `XXXXXXXXXXXXyyyXXXXXXXXXXXXXXyyyyyXXXXXXXX`, with positions `X` matching on the reference and `y` not matching, will produce the result (using gap at 3):

`XXXXXXXXXXXXyyyXXXXXXXXXXXXXX` is matching
`yyyyy` is not matching
`XXXXXXXX` is matching

It will produce the result (using gap option at 2):

`xxxxxxxxxxxx` is matching
`yyy` is not matching
`xxxxxxxxxxxxxx` is matching
`yyyyy` is not matching
`xxxxxxxx` is matching

Multi-process

FRAGS can be launched in a single core process (default behavior) or on many cores at the same time in order to speed-up some part of the computation.

Using 1 process (default):

```
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fna
Query time: 19.16s
Write file to be Blasted: 0.03s
Blast time: 0.78s
Write compressed Blast results: 0.00s
Write results files: 0.06s
Total time: 20.06s
```

Using 2 processes:

```
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fna -p 2
Query time: 10.35s
Write file to be Blasted: 0.03s
Blast time: 0.69s
Write compressed Blast results: 0.00s
Write results files: 0.06s
Total time: 11.15s
```

Using 4 processes:

```
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fna -p 4
Query time: 5.79s
Write file to be Blasted: 0.03s
Blast time: 0.61s
Write compressed Blast results: 0.00s
Write results files: 0.06s
Total time: 6.50s
```

Using 8 processes:

```
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fna -p 8
Query time: 5.52s
Write file to be Blasted: 0.03s
Blast time: 0.62s
Write compressed Blast results: 0.00s
Write results files: 0.06s
Total time: 6.26s
```

Not that some parts of the computations are not parallelized and increasing the number of processes will not speed-up the computation after a certain point.

Verbosity

Verbosity can be increased or decreased. The output file is not affected by **-v** or **-q** options.

With default verbosity level (no **-v** nor **-q** option), the output is:

```
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fna
Total time: 6.61s
$
```

Increasing verbosity, *i.e.* using **-v**, adds information about time. For example:

```
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fna -v
Query time: 5.56s
Write file to be Blasted: 0.03s
Blast time: 0.61s
Write compressed Blast results: 0.00s
Write results files: 0.06s
Total time: 6.28s
$
```

Decreasing verbosity, *i.e.* using **-q** option, removes all information but errors. For example:

```
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fdna -q
Error, host file host.fdna not found
$ frags -i reads.fastq -r ref1.fasta ref2.fasta -o res -p 8 -b -t host.fna -q
$
```

CSV configuration of output files

Main results files are in CSV format. Three separators are configurable. The first one (**-c**) is the column separator. By default columns are separated by tabulations.

The second one (**-s**) is the separator between position and size for matches or breakpoints (and potentially also for insertions, see *Output of FRAGS*). See *Nomenclature of positions* for more informations. By default, position and size are separated by `:`.

The third one (**-f**) is used to separate informations about the different matches in a same cell. By default, different informations in a single cell are separated by `|`.

See *Example of a CSV result line* for a better understanding.

Note: In `compressed.fasta` output file, option **-c** and **-f** are used. See *Output of FRAGS* for this.

4.2 frags

4.2.1 frags package

4.2.1.1 Submodules

4.2.1.2 frags.context module

4.2.1.3 frags.core module

Contains generic functions used by FRAG

`frags.core.`**`build_graph`**(*ref*, *k*)

Index each k-mers of a genome Aho-Corasick implementation, requires pypi package pyahocorasick

Parameters

- **`ref`** (*`str`*) – the reference to index
- **`k`** (*`int`*) – k-mer size

`frags.core.`**`find_hits`**(*graph*, *a_read*)

Find all kmers of ref present in a read All hits are on the form: start_pos_read: start_pos_ref

Parameters

- **`graph`** (`pyahocorasick`) – the graph to parse
- **`a_read`** (*`str`*) – the read to search in the index

`frags.core.`**`get_all_queries`**(*file*, *nb_proc*, *k*, *gap*, *graph1*, *graph2=None*)

Launch all parallel process to get all queries from a file

Parameters

- **`file`** (*`string`*) – the filename of the file where to take sequences from
- **`nb_proc`** (*`int`*) – number of precess to run in parallel
- **`k`** (*`int`*) – size of kmers
- **`gap`** (*`int`*) – maximum authorized gap size for continuous hits
- **`graph1`** – the graph to parse for genome1
- **`graph2`** – the graph to parse for genome2

`frags.core.`**`get_recombinations`**(*offset_start*, *offset_end*, *file*, *k*, *gap*, *graph1*, *graph2=None*)

Main parallelized function that retrieve each read of a offset range and find matches and breakpoint of them.

Parameters

- **`offset_start`** (*`int`*) – where to start taking sequences in the file
- **`offset_end`** (*`int`*) – where to stop taking sequences in the file
- **`file`** (*`string`*) – the filename of the file where to take sequences from
- **`k`** (*`int`*) – size of kmers

- **gap** (*int*) – maximum authorized gap size for continuous hits
- **graph1** – the graph to parse for genome1
- **graph2** – the graph to parse for genome2

`frags.core.`**`get_reference`**(*input_file*)

Get the reference genome in one-string. Only take the first sequence of the file. Can be fasta or fastq, gzipped or not.

Parameters **input_file** (*str*) – fasta/fastq file to use as reference

`frags.core.`**`next_read`**(*file*, *offset_start*, *offset_end*)

Return each sequence between offsets range of a file as a tuple (header, seq) using a generator. Can be fasta or fastq, gzipped or not. WARNING: spaces in headers are replaced by _

Parameters

- **file** (*str*) – fasta/fastq file to read
- **offset_start** (*int*) – offset in the file from where to read
- **offset_end** (*int*) – offset in the file until where to read

`frags.core.`**`prepare_blast_file`**(*breakpoint_file*, *all_queries*, *minsizeblast*)

Prepare a fasta file to be Blasted containing all breakpoints of at least minsizeblast nucleotides. WARNING: this wrote a FASTA file, regardless of the format of the original file WARNING: headers of original files are modified to add the information of which breakpoint(s) of a specific read are Blasted: original_header_#bp

Parameters

- **breakpoint_file** (*string*) – the filename of the file to be written
- **all_queries** (list(Read)) – all queries that may contain breakpoints
- **minsizeblast** (*int*) – minimal size of breakpoint accepted

`frags.core.`**`process_blast_res`**(*compressed_file*, *res_blast_file*, *sep*, *all_breakpoints*)

Compress Blast result to only show the bests hits and output result in a fasta-like file. Header is the original header with breakpoint id, e-value and bit-score. WARNING: this wrote a FASTA file, regardless of the format of the original file

Parameters

- **compressed_file** (*string*) – the filename of the file to be written
- **res_blast_file** (*string*) – Blast result file
- **sep** (*list (char)*) – separator to use in the result file
- **all_breakpoints** (*dict*) – dict of Breakpoints/index created before the Blast

`frags.core.`**`reverse_complement`**(*seq*)

Take an input sequence and return its revcomp

Parameters **seq** (*str*) – the seq to compute

`frags.core.`**`write_header`**(*output_file*, *sep='\t'*)
Write header of CSV output files

Parameters

- **`output_file`** (*`str`*) – CSV file to write in
- **`sep`** (*`char`*) – Separator to use between CSV columns

4.2.1.4 frags.read module

Contains class and functions related to read definition and use

`class` `frags.read.`**`Breakpoint`**(*beg_pos_read*, *size*)
Bases: `object`

Define a breakpoint.

Parameters

- **`beg_pos_read`** (*`int`*) – starting position in the read of this match
- **`size`** (*`int`*) – size of the match

`output`(*sep*)
Proper output of a line in the result file

Parameters **`sep`** (*`list (char)`*) – Separator to use in CSV

`class` `frags.read.`**`Match`**(*beg_pos_read*, *beg_pos_ref*, *strand*, *ref*, *size*, *inserts*, *seq_l*)
Bases: `object`

Define a match.

Parameters

- **`beg_pos_read`** (*`int`*) – starting position in the read of this match
- **`beg_pos_ref`** (*`int`*) – starting position in the ref of this match
- **`strand`** (*`int`*) – strand of this match
- **`ref`** (*`int`*) – the ref index for this match
- **`size`** (*`int`*) – size of the match
- **`inserts`** (*`list (int)`*) – size of potential insertions (possible to have several insertions in ONE match)
- **`seq_l`** (*`int`*) – size of the read (needed for rev comp computation)

`is_include_in`(*other*)
Check if this match is included in another match

Parameters **`other`** (*`Match`*) – the match to compare with

`output_read`(*sep*)
Correct output of read infos

Parameters **`sep`** (*`list (char)`*) – Separator to use in CSV

`output_ref`(*sep*)
Correct output of ref infos

Parameters **`sep`** (*`list (char)`*) – Separator to use in CSV

class `frags.read.`**`Read`**(*header*, *sequence*)

Bases: `object`

Define a read.

Parameters

- **`header`** (*`str`*) – header of the read
- **`sequence`** (*`str`*) – sequence of the read

`add_a_match`(*match*)

Test if this match should be added or not. It must be added if it is not a subpart of an already added other match. In some case, some already added matches are subparts of the match to add. If so, they are removed.

Parameters **`match`** (*`Match`*) – the match to add

`get_breakpoints`()

Populate breakpoints list using all hits, for both strands

`get_matches`(*hits*, *gap*, *k*, *strand*, *ref*)

Populate matches list from all hits, for one strand

Parameters

- **`hits`** (*`dict`*) – matching position on read and ref
- **`gap`** (*`int`*) – maximum authorized gap size for continuous hits
- **`k`** (*`int`*) – k-mer size
- **`strand`** (*`int`*) – the strand of this hit
- **`ref`** (*`int`*) – the reference index of this hit

`get_ref`()

Compute the ref of this Read (0=nothing / 1=ref1 / 2=ref2 / 3=ref1 AND ref2)

`get_strand`()

Compute the strand of this Read (-1=nothing / 0=normal / 1=revcomp / 2=normal AND revcomp)

`output`(*sep*)

Proper output of a line of the result file

Parameters **`sep`** (*`list(char)`*) – Separator to use in CSV

4.2.1.5 Module contents

Contains everything related to FRAGS software

4.3 CHANGELOG

- **0.1.4 (2020/12/18)** First beta release

PYTHON MODULE INDEX

INDEX