

# INDU- Read Mapping

Javier Hidalgo Moreno

DA7066 - Programming techniques for Life Sciences VT25

## Description of the program

All the functions reported here are part of the file `readmapping.py`. The program needs two fasta files as input: one for the reads and other for the reference sequences.

The program searches the best mapping location of a series of reads in different reference sequences. It starts by reading fasta files of the reads and the reference using the `read_fasta()` function. The files must have the extension `.fa` or `.fasta`, and the sequences must contain only A,T,G and C (this will be checked by `check_dna()` function). It is important that the fasta files are placed in the same working directory, otherwise a `FileNotFoundError` will be raised. The data in fasta files will be store in dictionaries.

The sequences will be broken in k-mers using `get_kmers()`. This function will create a dictionary with the k-mers' sequences and their indexes in the original sequence. The type of data structure chosen was dictionary and not list to reduce the time needed to iterate over the kmers.

Then, the function `kmers_alignment()` will search for coincidence between k-mers in the read and the reference sequences. For a single read sequence, this function will create a dictionary with the possible reference sequences and the index where the alignment starts. Then, the function `best_mapping_location()` will determine which is the position in which the alignment has the lowest errors (lowest hamming distance). This is done calculating the hamming distance using `hamming_distance()` function. For each read, the program will calculate from all the references where the best mapping location is. The function can also calculate if there are more than one best locations.

The function `get_data_for_output()` will prepare three dictionaries with data needed to build the output files. For output one, the function will create a dictionary with all the best mapping locations for each read as well as the position of the start of the alignment and the hamming distance. In the case of output two, the keys will be the references. In the dictionary, for each reference it will be store the reads whose best mapping location lies in that reference. Also, it will store the length of the read and the sequence, needed to calculate the coverage. Finally, it will create a dictionary with the number of best locations that each read has. This will be used for the coverage calculation and the histogram plotting.

The `write_output_file_1()` function will create a `.txt` file with the information from the first dictionary returned by `get_data_for_output()`. It will show the best mapping locations following this structure: *read\_annotation, reference\_annotation, start\_of\_the\_homology, number\_of\_errors*.

The function `calculate_coverage()` will get the coverage value for each reference using the data obtained in the dictionary two and three from `get_data_for_output()`. The coverage for each reference will be calculated using

the length of the read, the number of total best mapping locations (third dictionary), the number of best mapping locations for that reference and the length of the reference. Then, it will create a .txt file with the structure *reference\_annotation, coverage\_value*.

Finally, `get_plot()` will create an histogram displaying the number of mapping locations (x-axis) per read using the data from the third dictionary given by `get_data_for_output()`. The function ensures that each bin represent a single number by getting the maximum value from the data.

The program can be executed from the terminal due to the use of the `argparser` module. The user can call the program using:

```
pyhton readmapping.py --k 9 refs0.fa reads0.fa output0
```

`--k` is the length of the k-mers used for the calculation of the best mapping locations. It is an optional input, but it is highly recommended to change it depending on the length of the user's read. The default is 10, which ensure alignment with up to 4 errors for 50 pb long reads (pigeon principle). It should not be very high because the program will miss alignment, but not very low as it will increase the computational cost.

`refs.fa` and `reads.fa` are the names of the fasta files containing the reference and reads sequences, respectively.

`output0` is the name of the folder that will be created (if not existent) to store the three output files (best mapping locations, reference coverage and histogram).

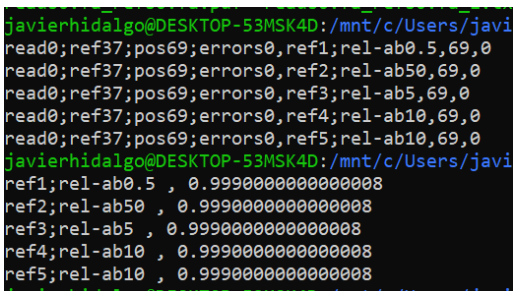
The program uses the modules `argparser` and `os`, which are part of python standard distribution. It also uses `matplotlib` which needs to be previously installed by writing `pip install matplotlib` in the terminal.

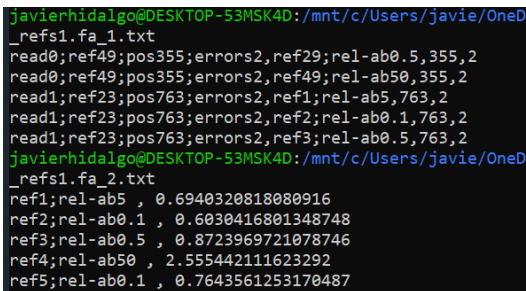
### Outputs for reads0, reads1, reads5 and reads10.

The program was used to calculate the best mapping locations and coverage for the same read file with references having 0%, 1%, 5% and 10% mutation rate to each other.

Here, you can find all the [output files](#).

This is head -n of the .txt files for reads0 (a), reads1 (b), reads5 (c) and reads 10(d).

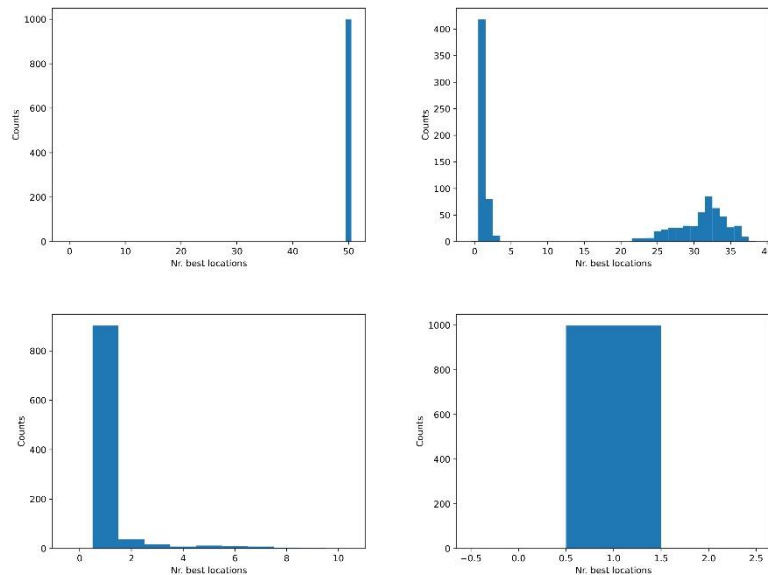
a.  `javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javi`  
`read0;ref37;pos69;errors0,ref1;rel-ab0.5,69,0`  
`read0;ref37;pos69;errors0,ref2;rel-ab50,69,0`  
`read0;ref37;pos69;errors0,ref3;rel-ab5,69,0`  
`read0;ref37;pos69;errors0,ref4;rel-ab10,69,0`  
`read0;ref37;pos69;errors0,ref5;rel-ab10,69,0`  
`javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javi`  
`ref1;rel-ab0.5 , 0.9990000000000008`  
`ref2;rel-ab50 , 0.9990000000000008`  
`ref3;rel-ab5 , 0.9990000000000008`  
`ref4;rel-ab10 , 0.9990000000000008`  
`ref5;rel-ab10 , 0.9990000000000008`

b.  `javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javie/OneD`  
`_refs1.fa_1.txt`  
`read0;ref49;pos355;errors2,ref29;rel-ab0.5,355,2`  
`read0;ref49;pos355;errors2,ref49;rel-ab50,355,2`  
`read1;ref23;pos763;errors2,ref1;rel-ab5,763,2`  
`read1;ref23;pos763;errors2,ref2;rel-ab0.1,763,2`  
`read1;ref23;pos763;errors2,ref3;rel-ab0.5,763,2`  
`javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javie/OneD`  
`_refs1.fa_2.txt`  
`ref1;rel-ab5 , 0.6940320818080916`  
`ref2;rel-ab0.1 , 0.6030416801348748`  
`ref3;rel-ab0.5 , 0.8723969721078746`  
`ref4;rel-ab50 , 2.555442111623292`  
`ref5;rel-ab0.1 , 0.7643561253170487`

c. `javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javie/OneDrive/_refs5.fa_1.txt`  
`read0;ref19;pos53;errors3;ref19;rel-ab50,53,3`  
`read1;ref19;pos1589;errors1;ref19;rel-ab50,1589,1`  
`read2;ref43;pos1831;errors2;ref43;rel-ab50,1831,2`  
`read3;ref33;pos213;errors1;ref33;rel-ab50,213,1`  
`read4;ref43;pos480;errors2;ref43;rel-ab50,480,2`  
`javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javie/OneDrive/_refs5.fa_2.txt`  
`ref1;rel-ab1 , 0.19517857142857148`  
`ref2;rel-ab50 , 2.280119047619048`  
`ref3;rel-ab50 , 2.95583333333333313`  
`ref4;rel-ab50 , 2.660476190476189`  
`ref5;rel-ab1 , 0.21339285714285713`

d. `javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javie/OneDrive/fa_refs10.fa_1.txt`  
`read0;ref9;pos327;errors1;ref9;rel-ab50,327,1`  
`read1;ref39;pos827;errors3;ref39;rel-ab50,827,3`  
`read2;ref35;pos1637;errors3;ref35;rel-ab50,1637,3`  
`read3;ref49;pos975;errors3;ref49;rel-ab50,975,3`  
`read4;ref11;pos1;errors0;ref11;rel-ab50,1,0`  
`javierhidalgo@DESKTOP-53MSK4D:/mnt/c/Users/javie/OneDrive/fa_refs10.fa_2.txt`  
`ref1;rel-ab1 , 0`  
`ref2;rel-ab10 , 1.0000000000000002`  
`ref3;rel-ab0.5 , 0.1`  
`ref4;rel-ab0.5 , 0`  
`ref5;rel-ab1 , 0`

These are the plots, which are very similar to the ones in the instructions:



The increase in the mutation rate reduces the average number of best locations for each read and increases the variability in the coverage for each reference sequence.

### Testing with unittest

The program was tested using unittest in `test_readmapping.py`. These tests cover all main functions of the program. Various scenarios were tested, including

- Invalid DNA sequences
- K-mer generation and the case where  $k > \text{len}(\text{seq})$ .
- Sequences aligning twice to the same reference
- Sequences not aligning to any reference
- Alignments at the very start or end of a reference
- Complex mapping situations involving reads aligning to multiple locations, references covered by the same read multiple times, references with no coverage and references covered by multiple reads