

fastGEAR – Manual

fastGEAR was developed in a collaborative project by Rafal Mostowy, Nicholas J. Croucher, Cheryl P. Andam, Jukka Corander, William P. Hanage, and Pekka Marttinen*

Contents

Introduction.....	2
Copyright notice	2
Installation.....	2
Running the program.....	3
Running the executables In Linux/Unix	3
Running the source code in Matlab.....	4
Changing input specifications (including defining lineages manually)	4
Interpreting the outputs of the program	5
lineage_information.txt.....	5
recombinations_ancestral.txt.....	5
recombinations_recent.txt.....	6
Visualizing the results.....	7
Plotting the detected recent recombinations	7
Plotting the detected ancestral recombinations with a user-specified ordering for the strains.....	8
For reference, show colors used in the recombination plots	8
Plotting probabilities of the different lineages along the sequence for a strain	9
Additional functions for postprocessing fastGEAR outputs	9

Introduction

fastGEAR is a software for analysing sequence alignments, and it has been described in a paper:

Mostowy, R., Croucher, N.J., Andam, C.P., Corander, J., Hanage, W.P. and Marttinen, P. (2017). Efficient inference of recent and ancestral recombination within bacterial populations. *Molecular Biology and Evolution*, 34(5), 1167-1182.

<https://doi.org/10.1093/molbev/msx066>

Copyright notice

© 2016 Authors

fastGEAR can be downloaded from <https://users.ics.aalto.fi/~pemartti/fastGEAR/> (link to be updated upon acceptance) and used freely for academic purposes. If you use *fastGEAR*, please cite the paper specified above. Furthermore, you can modify and redistribute the source code on the condition that the original citation and this notice accompany the modified version.

***fastGEAR* comes with no warranties whatsoever. The user alone is responsible for results obtained with *fastGEAR*.**

Questions and feedback can be sent to:

pekka.marttinen@aalto.fi

Installation

To run the precompiled version of *fastGEAR* you first need to install the Matlab Runtime component (MCR). As the MCR versions change periodically, we provide the version which is compatible with the version of *fastGEAR*, and which you can obtain from <https://users.ics.aalto.fi/~pemartti/fastGEAR/>. Having downloaded it, you should follow instructions for installation from <http://uk.mathworks.com/help/compiler/install-the-matlab-runtime.html>

For Windows, the installation is relatively straightforward. For Linux, the easiest way to install it is to unzip the downloaded file, enter into the uncompressed directory and enter

```
DIR=/m/fs/software/matlab/r2016a
./install -mode silent -agreeToLicense yes -destinationFolder $DIR
```

where \$DIR is an example of prespecified path to a directory where this version of MCR should be installed. This path should be later used to run the script.

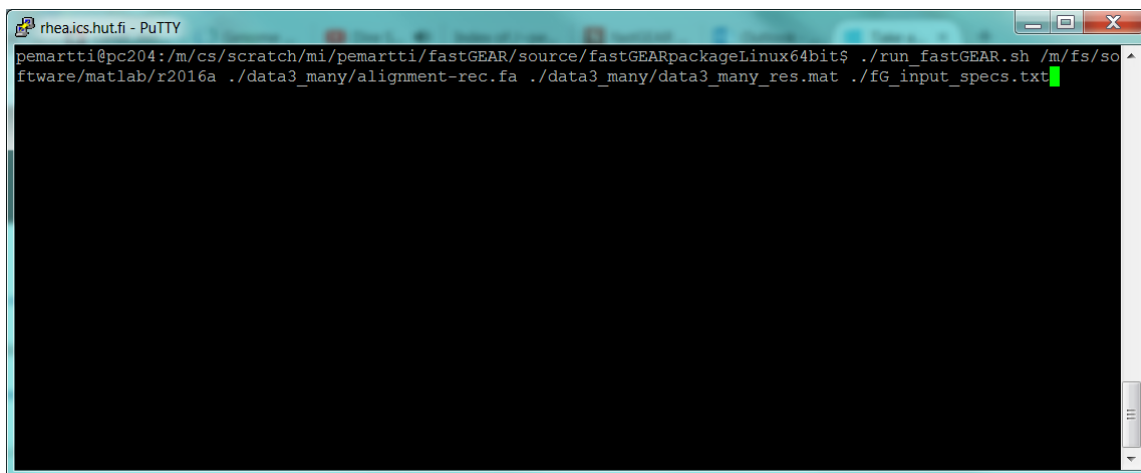
Running the program

Running the executables In Linux/Unix

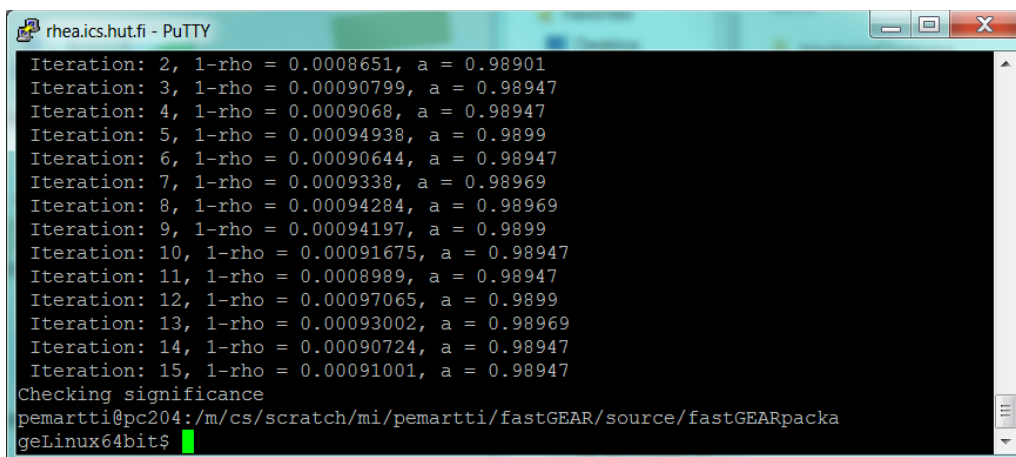
Go to the directory where the executables are located, and start the executable by using `run_fastGEAR.sh` script. The following example runs the analysis for an alignment in a file `alignment-rec.fa` located in sub-directory `./data3_many`. This directory needs to be created prior to running the script. The `outputFile` must have a `'.mat'` extension.

```
# USAGE:
# ./run_fastGEAR.sh <full path to matlab/mcr> <inputFile> <outputFile>
<inputSpecsFile>

# EXAMPLE:
./run_fastGEAR.sh /m/fs/software/matlab/r2016a ./data3_many/alignment-
rec.fa ./data3_many/data3_many_res.mat ./fG_input_specs.txt
```



After the analysis finishes, the window looks as follows:



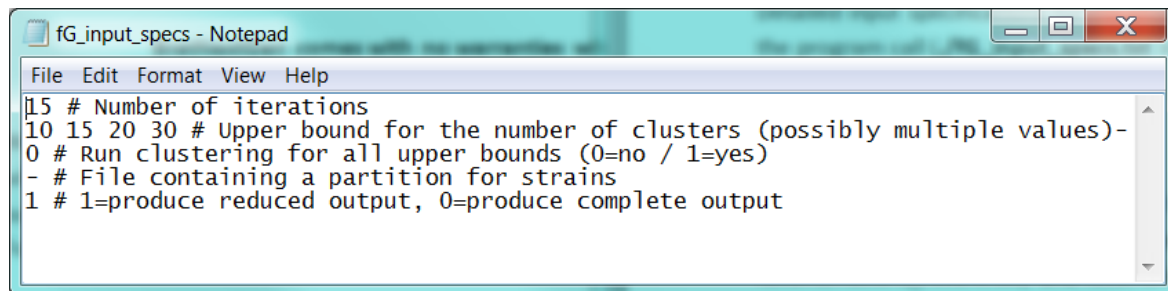
Results as text can now be found in files `"lineage_information.txt"`, `"recombinations_recent.txt"`, and `"recombinations_ancestral.txt"` in the output directory which is located in the same directory as the output file specified. In the example above, the output directory would be `./data3_many/output`.

Running the source code in Matlab

Alternatively, you can run the Matlab source code directly from Matlab. For an example on how to do this, see testFastGear.m in the source code package.

Changing input specifications (including defining lineages manually)

NOTE: by default you don't need to change the input specifications, but you can just use the input specification file included in the package. Detailed input specifications for the program are given in a file that is provided as the fourth argument of the program call (`./fg_input_specs.txt` in the example). This is what the input specifications file looked like in the example (the same file was used in the analyses in the manuscript):



```
fg_input_specs - Notepad
File Edit Format View Help
15 # Number of iterations
10 15 20 30 # Upper bound for the number of clusters (possibly multiple values)-
0 # Run clustering for all upper bounds (0=no / 1=yes)
- # File containing a partition for strains
1 # 1=produce reduced output, 0=produce complete output
```

The first line specifies the number of iterations for iteratively learning recombinations and updating model parameters. Convergence can be monitored from screen output, which shows the estimated parameter values at each iteration. The values don't converge to a single point because of the stochastic nature of the search algorithm (when the parameter values start oscillating, rather than increasing or decreasing monotonically, the algorithm can be considered converged). In our experiments the algorithm usually converges in less than 10 iterations, and the default value 15 is expected to be appropriate for analyses similar to those presented in the article.

The second line specifies the initial number of clusters for the BAPS3 clustering algorithm that is used by *fastGEAR* (this specifies at the same time an upper bound for the number of clusters). The clustering algorithm is run with each of the specified initializations and returns the most probable clustering.

The third line specifies whether the clustering algorithm should stop as soon as it identifies a clustering where the number of clusters is less than the given initial number/upper bound. The rationale here is that if you start with a too small number of clusters, say 10, and you identify 10 clusters, then you should try with a larger upper bound. On the other hand, if you run the clustering algorithm with very many different initial values, that takes time. With this parameter you can set multiple initial numbers of clusters on the second line, but not waste time running all of them, if a suitable clustering is found, having fewer clusters than the specified upper bound.

The fourth line contains a file with **a pre-specified partition** that will be used instead of the clustering algorithm **to define lineages that will be used in the analysis**. If a file name is given here, then specifications related to the clustering algorithm on lines 2 and 3 will have no effect. The partition file should have as many rows as there are strains in the data set, and on each row there should be a single positive integer that specifies the cluster for the corresponding strain. An example partition file is `./data3_many/data3_partition.txt`, which contains lineages that are exactly the same as what would be obtained by running the algorithm. When defining lineages manually, the largest lineage should have the largest cluster label. For example, if there are three lineages of sizes 15,100,30, then the cluster labels

should be 1,3,2, respectively. This is because when detecting ancestral recombinations, fastGEAR assumes the smaller lineage to be recombinant. ***We note that we have validated the method only using the default approach with lineages defined by the clustering algorithm, and therefore it is not clear to us whether sensible results may be obtained by defining the lineages in some alternative way.***

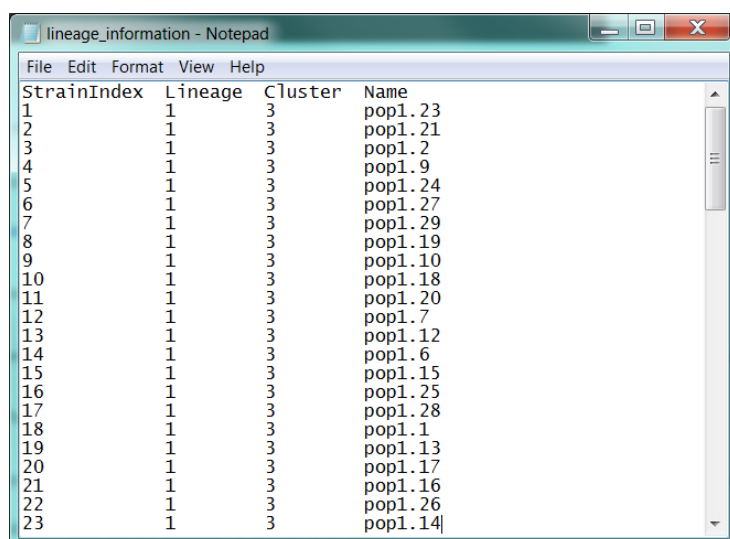
The fifth line specifies whether the program should produce “reduced” or “complete” output. If the “complete” output is selected, then in the output folder there will be one file for each strain containing the marginal probabilities of the different lineages for the strain at each sequence position. These are required if you want to look in detail the marginal probabilities using “plotMarginalsForStrain” (see *Visualizing the results* section). By default, the reduced output is used, because if the analysis is run for many separate gene alignments, and if each run produced one output file per strain, that would total to very many files being produced by the analysis, which might hamper the performance (this depends on the details of the computing cluster you’re using).

Interpreting the outputs of the program

The main outputs of the program are the three text files: *lineage_information.txt*, *recombinations_recent.txt*, and *recombinations_ancestral.txt*, all of which can be found in the output directory.

lineage_information.txt

The beginning of the file *lineage_information.txt* for the example analysis is shown below. The file has as many rows as there were sequences in the input alignment. The columns of the file are interpreted as follows: *StrainIndex*: This is the index of the sequence/strain in the input alignment. *Lineage/cluster* show the inferred lineage/cluster for the strain. In the example there were 4 clusters detected, so the numbers in the *Cluster* column range from 1 to 4. In addition, these clusters were merged to produce 3 lineages. Therefore, the possible values in the Lineage columns are from 1 to 3. *Name* specifies the name of the strain in the input alignment.

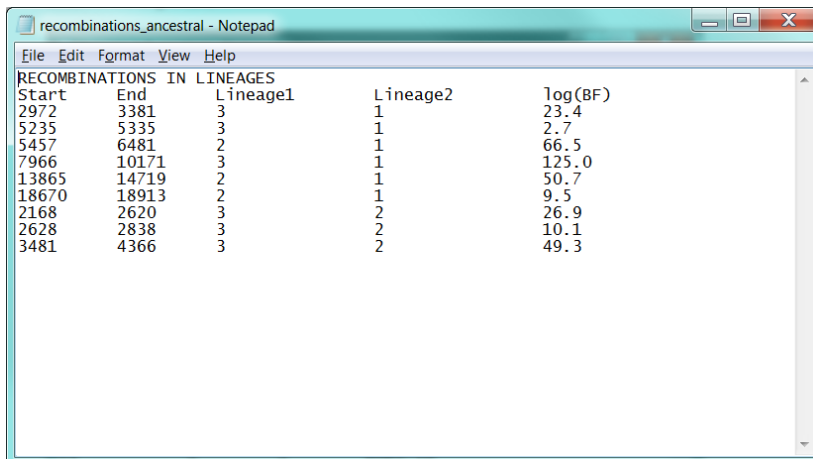


StrainIndex	Lineage	Cluster	Name
1	1	3	pop1.23
2	1	3	pop1.21
3	1	3	pop1.2
4	1	3	pop1.9
5	1	3	pop1.24
6	1	3	pop1.27
7	1	3	pop1.29
8	1	3	pop1.19
9	1	3	pop1.10
10	1	3	pop1.18
11	1	3	pop1.20
12	1	3	pop1.7
13	1	3	pop1.12
14	1	3	pop1.6
15	1	3	pop1.15
16	1	3	pop1.25
17	1	3	pop1.28
18	1	3	pop1.1
19	1	3	pop1.13
20	1	3	pop1.17
21	1	3	pop1.16
22	1	3	pop1.26
23	1	3	pop1.14

recombinations_ancestral.txt

The file *recombinations_ancestral.txt* for our example is shown below. It displays information about all ancestral recombinations detected between the lineages. *Start* and *End* are the estimated starting and ending positions of the recombinant segment. *Lineage1* and *Lineage2* are the lineages between which an ancestral recombination was detected. *Lineage1* is the larger one and *Lineage2* the smaller one of the two

lineages, and therefore, based on the principle of maximum parsimony, *Lineage1* may be considered the donor and *Lineage2* the recipient of the recombination, although we emphasize that the direction is not formally identifiable using *fastGEAR* (see related Discussion in the article). $\log(BF)$ shows the (natural) logarithm of the Bayes factor, a measure of statistical significance, computed for the recombination, and it's based on changes in SNP density between the two lineages. Interpretation of $BF=10$, for example, would be that it is 10 times more probable that there is a change in SNP density supporting a recombination than that there is no such change. Ancestral recombinations with $BF<10$, that is, $\log(BF)<2.3$ are removed in the significance checking step of the program.



Start	End	Lineage1	Lineage2	log(BF)
2972	3381	3	1	23.4
5235	5335	3	1	2.7
5457	6481	2	1	66.5
7966	10171	3	1	125.0
13865	14719	2	1	50.7
18670	18913	2	1	9.5
2168	2620	3	2	26.9
2628	2838	3	2	10.1
3481	4366	3	2	49.3

recombinations_recent.txt

The beginning of the file ***recombinations_recent.txt*** for our example is shown below. The columns *Start* and *End* again denote the boundaries of the recombinant segment. *DonorLineage* is the lineage that donated the recombination and *RecipientStrain* is the sequence that received the recombination. Note that unlike with ancestral recombinations, the direction of recent recombinations can be identified by *fastGEAR*. The column $\log(BF)$ shows the logarithm of the Bayes factor, that represents the statistical significance of the recombination, and is based on changes in SNP density between the recombinant strain and its 'home' lineage, i.e., the lineage to which the strain was assigned in the clustering phase. Threshold $BF=1$, i.e., $\log(BF)=0$ is used for pruning recent recombinations in the significance checking step of the algorithm. This means that recent recombinations where it's more probable that there is a change in SNP density supporting the recombination, than that there is no such change in SNP density, are retained, while other putative recombinations are removed. Note that the same recombination may be seen in multiple strains as a result of a single recombination event that has affected the common ancestor of many strains that all share the recombination. The number on the first line of *recombinations_recent.txt* shows the estimated number of recent recombination events, where overlapping recombinations from the same donor are counted as one.

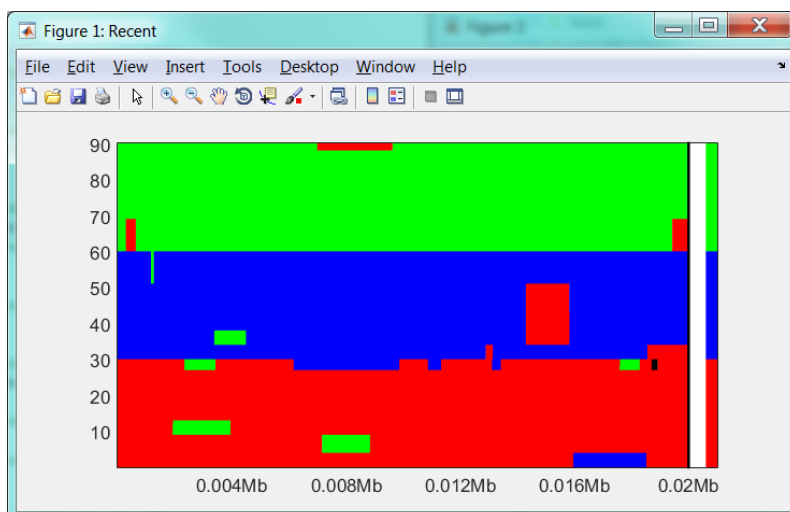
recombinations_recent - Notepad						
File Edit Format View Help						
16 RECENT	RECOMBINATION EVENTS					
Start	End	DonorLineage	RecipientStrain	log(BF)	StrainName	
6991	9627	2	1	300.6	pop1.23	
6991	9627	2	2	302.8	pop1.21	
289	634	2	22	96.3	pop1.26	
19422	20000	2	22	105.1	pop1.26	
289	634	2	23	97.4	pop1.14	
19422	20000	2	23	106.3	pop1.14	
289	634	2	24	95.3	pop1.5	
19422	20000	2	24	103.9	pop1.5	
289	634	2	25	96.3	pop1.30	
19422	20000	2	25	105.1	pop1.30	
289	634	2	26	96.3	pop1.11	
19422	20000	2	26	105.1	pop1.11	
289	634	2	27	97.4	pop1.3	
19422	20000	2	27	106.3	pop1.3	
289	634	2	28	97.4	pop1.8	
19422	20000	2	28	106.3	pop1.8	
289	634	2	29	96.3	pop1.4	
19422	20000	2	29	105.1	pop1.4	
289	634	2	30	97.4	pop1.22	
19422	20000	2	30	106.3	pop1.22	

Visualizing the results

Plotting the detected recent recombinations

```
# USAGE:
# ./run_plotRecombinations.sh <path to matlab/mcr> <outputFile> <type
(1=recent, 2=ancestral)> <strainOrder (1=original, 2=by cluster, or
fileWithOrdering)>
#
# EXAMPLE:
./run_plotRecombinations.sh /m/fs/software/matlab/r2016a
./data3_many/data3_many_res.mat 1 1
```

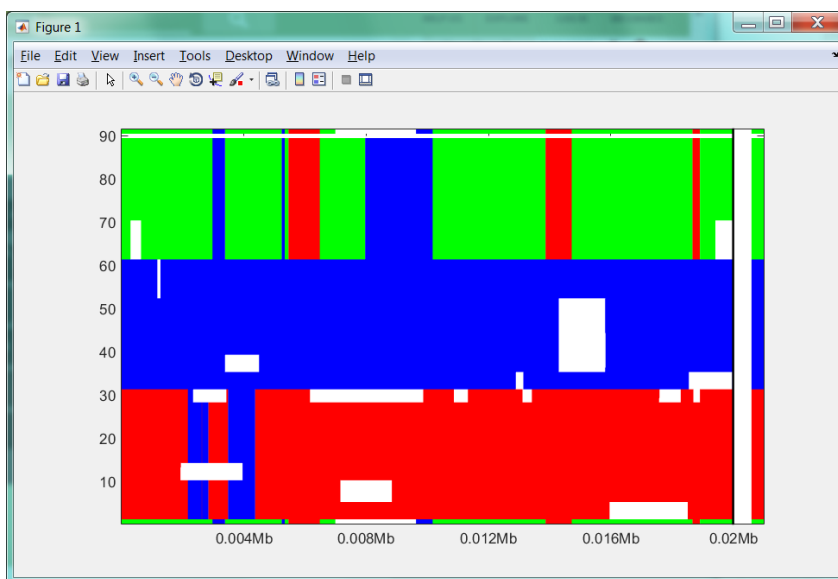
The following figure shows recent recombinations for the example data. On the y-axis are the 90 sequences present in the alignment and on the x-axis the sequence positions. The annotation on the right side of the panel shows division of the strains into three lineages (green, blue, red). The sequences are colored based on the ancestry of the sequences (black color denotes recombination estimated to come from outside of any lineage in the data set). The code that draws this figure produces another output file `./data3_many/output/order_in_plot_original_ordering.txt`, which shows the order in which the strains appear in the figure. Note that the window must be closed before the execution of the script continues.



Plotting the detected ancestral recombinations with a user-specified ordering for the strains

```
# Plot ancestral recombinations
# EXAMPLE:
./run_plotRecombinations.sh /m/fs/software/matlab/r2016a
./data3_many/data3_many_res.mat 2 ./data3_many/data3_given_ordering.txt
```

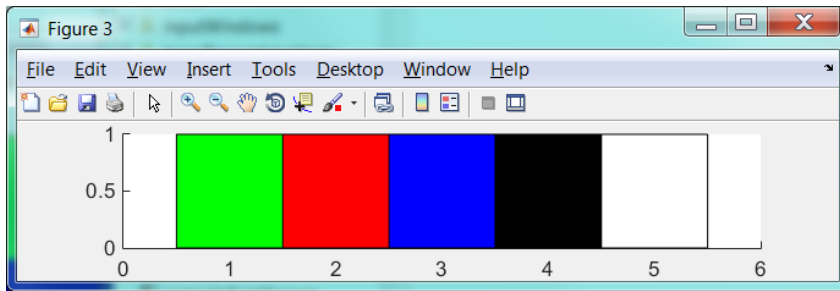
The following figure shows ancestral recombinations in the example data. The recent recombinations are removed before the analysis of ancestral recombinations, and therefore the recent recombinations are shown as white gaps. The order of strains is here specified in file *./data3_many/data3_given_ordering.txt*, and can also be seen in an output file *./data3_many/output/order_in_plot_given_ordering.txt*. **Note that you can specify the ordering in a similar way when plotting recent recombinations**, even if this is not done in the example above. You can compare the results (both recent and ancestral) to the true population structure shown in *./data3_many/plot_original.png*.



For reference, show colors used in the recombination plots

```
# USAGE:
# ./run_plotColors.sh <path to matlab/mcr> <number of lineages, use - for
the number detected> <outputFile>
#
# EXAMPLE:
./run_plotColors.sh /m/fs/software/matlab/r2016a -
./data3_many/data3_many_res.mat
```

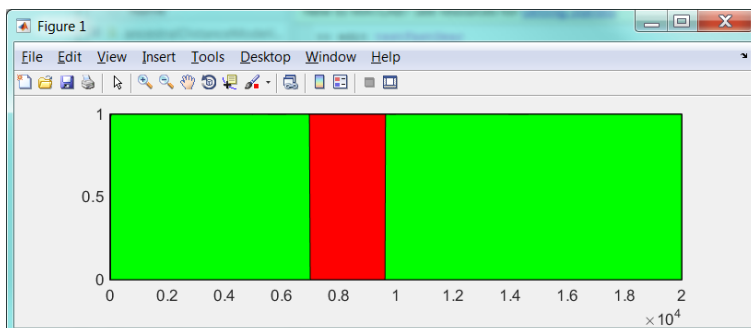
To aid interpretation of the two previous figures, the plotted recent and ancestral recombinations, the *plotColors* function shows which color is related to which lineage label. For example, below the 1st lineage has green color, 2nd lineage red color, 3rd lineage blue, and finally, the outside origin, not corresponding to any of the lineages, has black color.



Plotting probabilities of the different lineages along the sequence for a strain

```
# USAGE:
# ./run_plotMarginalsForStrain.sh <path to matlab/mcr> <outputFile>
# <strainIndex> <donorLineage (if =0, then shows all lineages)>
#
# EXAMPLE
./run_plotMarginalsForStrain.sh /m/fs/software/matlab/r2016a
./data3_many/data3_many_res.mat 1 0
```

This operation draws a figure where the probabilities of the different lineages are shown at all sequence positions for the specified strain. Note that in order to draw the figure, in the input specification file the “complete output” option must have been selected (not default). In this figure we see that the green lineage has probability approximately equal to unity everywhere except in a short region, which corresponds to a recombination from the red lineage. In that region the red lineage has probability approximately equal to unity.



Additional functions for postprocessing fastGEAR outputs

Additional scripts for processing fastGEAR outputs can be found in https://users.ics.aalto.fi/~pemartti/fastGEAR/postprocessing_scripts/

Here you can find, e.g., functions for collecting recombination counts for multiple genes, plotting multiple genes side-by-side, computing the proportion of shared ancestry, and reconstructing changes in the population structure on the branches of a given phylogenetic tree. The documentation for these additional postprocessing functions can find in *fastGEAR_post_processing_example.pdf*, also available behind the link.

Python versions of the postprocessing scripts are available

https://github.com/shimbalama/post_fastGEAR. These scripts have been written by Liam McIntyre, University of Melbourne.