The Livnat Lab

How Evolution Happens

The MEMDS analysis pipeline

Parameter file guide

By

Assaf Malik, Evgeni Bolotin, Daniel Melamed, Yuval Nov and Adi Livnat

Department of Evolution - University of Haifa;

Corresponding author: Adi Livnat;
Email: adi.livnat@sci.haifa.ac.il

Table of contents

Introduction

The "Parameter_file_preparation" guide provides detailed information regarding parameter files utilized by the MEMDS analysis pipeline during its run. It encompasses parameter file preparation chapter in the main MEMDS pipeline guide, and is provided as a separate file for user convenience.

Preparation of the parameter files

The pipeline relies on user-defined data in the parameter files during its run. All the files are housed under the "scripts" directory. **Before starting a new run of the pipeline always remember to check that the parameters are correct and match your data!**

The pipeline utilizes the following parameter files during its run:

fastq_merging/samples_table_0.txt

This file defines parameters for merging partial ".fastq" files into a single data file. This is an optional step for same-sample data sequenced across several lanes. The file contains three columns:

1) **pair**: a serial number assigned by the user to the analyzed fastq files. For **single-end** data, **each read file** should have a unique pair number. For **paired-end** data **each pair of files** should have a unique pair number. Forward and reverse read files belonging to the same pair should share same pair number.

2) **sample:** A name of the sample to which the read files belong. All files sharing same sample name would be merged, resulting in a single ".fastq" file per sample for single-end data and in a pair of files for paired-end data (forward and reverse reads).

3) **file:** A full path to the location where the read files are stored, ending with the name of the file:

e.g.: /data/home/user/experiment/Raw_data/S1_L001_R1_001.fastq.gz.

For **paired-end** data the script is designed to parse a **first** path in the pair as containing **forward-read** file and the second path - as containing **reverse-read** file. **Before concatenating the files always ensure that they are listed in a correct**

order across all analyzed pairs!

scripts/wildcard_adapters_1.fa

This file contains sequence information of adapters and other contaminants that might be present in the analyzed data. Sequences specified in this file would be removed from the raw sequence data during the quality control step.
The file can be opened with any text editor program to check its contents and add additional sequences to it, as needed. New sequences should be added in FASTA format, as follows:

> Sequence_name

Nucleotide Sequence

config_files/samples_table.txt

This file stores information regarding the location of the analyzed data files. **It has the same structure as the "samples_table_0.txt" parameter file used for merging partial ".fastq" files (see above)**.

Note: If the data files underwent merging before the analysis, remember to provide here location of the merged files and not of the original!

config_files/params_1.sh

This file defines a number of parameters used by the pipeline to analyze the data:

1) **Conda settings:**

. /path_to_conda_install_dir/miniconda2/etc/profile.d/conda.sh

conda activate modules3

The first line invokes "conda.sh" script so Conda commands can be used from the shell. This line should contain **a full path** to the "conda.sh" script (found within Conda installation directory under "etc/profile.d/"). **Important:** The dot before the path is a part of the command, not a typo!
The second line activates Conda environment. If the pipeline-related environment was created with a custom name (see explanation on Conda above), the default "modules3" name in the command should be replaced by the custom name.
In case that the dependencies were installed manually and installation folders were

added to the $PATH, this part of the script can be removed. Alternatively, it can be used to export location of the manually installed programs to the environment, instead of the Conda commands. To export paths, use the following command:
export PATH=$PATH:/path/to/program/:/path/to/program2/

2) **params_adapters_1:** Specifies path to the file containing sequence information of adapters and other contaminants that might be present in the analyzed data. This file is used to identify unwanted sequences in the data during the quality control step. By default **params_adapters_1** points to the adapter file distributed with the pipeline - "wildcard_adapters_1.fa" (see explanation above).

3) **params_dir_out_1:** Specifies location of the output directory to store results produced by the pipeline.

4) **params_dir_reference:** Specifies location of the reference file directory. The pipeline aligns analyzed data against the reference files to identify mutations. See the "Reference file preparation" section below for further information on the reference files.

5) **Trimmomatic options:** Options supplied to the Trimmomatic program during the quality control step. This program is used to identify and remove potential contaminants and low quality bases from the analyzed reads. The following options can be defined:
a) **params_minimum_fastq_size_1:** Defines minimal length threshold for the reads to be included in the subsequent analyses. Sequence shorter than the threshold would be removed (default: 90 bp).
b) **qual_threshold:** Average quality score threshold for read bases. Bases having lower quality score than the threshold would be cut from the read by Trimmomatic (default: 30).
c) **qual_window:** Specifies number of consequent bases in the read over which average quality score is calculated (default: 3)

6) **is_SE:** Defines whether the analyzed data is paired-end or single-end. Specify '0' for **paired-end** data and '1' for **single-end** (default: 0)

7) **offset (read sorting):** In the read sorting step of the pipeline (see below) this parameter specifies the offset (in bp) relative to **sorting positions (**see explanation on **factors_table.txt: sort_pos** below) for pipeline to search for nucleotides identifying the read as belonging to gene of interest. The search is done within **sort_pos +- offset** (default: 3).

8) **BC3_min:** During mutation calling step this parameter specifies how many different secondary barcodes should be associated with a mutation for it to be included in further analyses (default: 1). It is inadvisable to use stringent parameter here, since additional steps aimed at filtering out potential artifact mutations are undertaken down the road.

9) **TSS:** This variable allows the pipeline to report mutation position relative to the Translation Start Site (TSS) of the analyzed gene, in addition to its position on the read itself. Here two numbers, separated by comma, are specified (**e.g.: -30,1):**

a) **Position of the first base in the read relative to the TSS**. If TSS occurs before the analyzed portion of the gene, this value should represent position of the first read base in the analyzed gene (e.g. 1100). If TSS falls **inside** the read, this value should be negative and calculated as **1 - TSS position in the read**. E.g., if TSS occurs at **position 31 of the read**, it relative position is: **1 - 31 = -30.**
b) **Read orientation relative to the coding strand.** 1 - same orientation; -1 - opposite orientation (complimentary strand was sequenced).

Thus, **-30,1** example above means that the read and the coding strand of the gene are in the same orientation and TSS falls in the 31st position of the read.

Note: If TSS data is not relevant, one can specify **1,1** here to report same position for mutation location in the read and its location relative to the TSS.

10) **filter_pos:** Defines position of the control insertion used to identify artifacts occurring during the amplification stage. This parameter tells the pipeline to disregard mutations at this position (reads with the control insertion are filtered out beforehand),

since they do not represent mutations of interest. If control insertion is not used, position 0 should be specified here.

11) **Consensus cut-offs:** These parameters set cutoff criteria used to identify consensus variants. Following parameters can be defined:

a) **min_freq -** Comma-separated list defining variant frequency cutoffs within read family for variant to be considered "true". Variants not passing the cutoff criteria would be considered artifacts.

E.g: **min_freq="0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0"**

b) **min_count -** Comma-separated list defining minimal size of read families to be used in the consensus variant analysis. Read families having less reads than the cutoff criteria would be removed from the analysis.

E.g.: **min_count="0,1,2,3,4,5,6,7,8,9,10,25,50"**

c) **min_bc3 -** Comma-separated list defining minimal number of unique secondary barcodes to be associated with a variant within read family for variant to be considered true. Variants not passing the cutoff criteria would be considered as artifacts, unless they pass criteria for number of secondary barcodes associated with more than one read (see below).

E.g.: **min_bc3="0,1,2,3,4,5"**

d) **bc3groupCountOK -** Variable defining minimal number of unique secondary barcodes to be associated with a variant within read family for variant to be considered true. If **min_bc3** cutoff defined above is smaller than this variable, the variant would be considered true only if it passes **bc3groupsWithReadsAbove1** criteria (see below).

By controlling this variable, the user sets a threshold above which sheer number of secondary barcodes associated with a variant is enough to confirm its authenticity. But below it, some of the secondary barcode read groups associated with a variant have to be of a specific size for variant to pass the secondary barcode group thresholds. The idea is that a variant associated with large enough number of reads is more likely to be authentic than not, even if it is associated with smaller number of different secondary barcodes.

E.g.: **bc3groupCountOK=0**

a) **bc3groupsWithReadsAbove1 -** Variable defining minimal number of unique secondary barcode groups containing user-defined number of reads each to be associated with a variant within read family for variant to be considered true. A variant needs to pass either **bc3groupsWithReadsAbove1** or **bc3groupCountOK** criteria to be considered true. Minimal number of reads in the secondary barcode read groups analyzed here is defined by the **BC3_min** parameter (see point 8 above).
E.g.: **bc3groupsWithReadsAbove1=0**

config_files/factors_table.txt

This file defines parameters of barcode trimming, sorting by origin and variant search for analyzed sequences. It is a tab-delimited file with 17 columns:

1) **sample**: Name of the analyzed sample to which parameters appearing in the next columns would be applied. Sample names listed here should be consistent with the names listed in the 'sample' field of the 'samples_table.txt' parameter file.

2) **reference_size:** Length of the reference sequences to which the analyzed reads are aligned (in bp). **The pipeline can work with multiple references per sample, but only if they have same length, hence this field accepts only a single length value for all reference sequences used.**

3) **size_f:** Size of the primary barcode (5'), in bp, attached to the analyzed reads by the MEMDS procedure. The barcode includes unique sequence to distinguish between group of reads in the sample + identifier bases to distinguish between reads from different samples.

4) **size_r:** Size of the secondary barcode (3'), in bp, attached to the analyzed reads by the MEMDS procedure. The barcode includes unique sequence to distinguish between group of reads in the sample + identifier bases to distinguish between reads from different samples.

5) **limit_starts:** Start position of the window in which the pipeline would search for

mutations in the analyzed reads. If left empty, the pipeline would search from the first position of the reference. To specify multiple search windows, use comma-separated list of values (e.g: 15,31). Positions of "planted" mutations that indicate problems with barcode attachment (see explanation below for columns 7 - 9) also should be listed here.

6) **limit_ends:** End position of the window in which the pipeline would search for mutations in the analyzed reads. If left empty, the pipeline would search until the end of the reference. To specify multiple search windows, use comma-separated list of values (e.g: 15,84). The order of the end position values in the list should match the order of the start positions in the previous column. Positions of "planted" mutations that indicate problems with barcode attachment also should be listed here.

7) **seq_pos:** During the MEMDS procedure each analyzed DNA sequence is barcoded with a set of unique barcodes at its 5' and 3' ends. To account for the rare events when oligonucleotides used to attach the primary barcode (5') undergo extension themselves, using the barcoded DNA as a template, a single base insertion is planted in the oligo sequence. 'Seq_pos' field specifies a position of this insertion, which allows the pipeline to identify these sequences and remove them from further analyses.

8) **seq_mut:** This field specifies what allele should be found at the position specified by 'seq_pos' field to mark the sequence as an extended oligo and remove it from the analysis. The allele is listed as <reference_nucleotide><query_nucleotide>, with INDELs marked by hyphen ("-") (e.g.: -G).

9) **seq_action:** This field specifies what action to take on sequences containing the allele defined by the previous fields. **Currently, the pipeline is designed to remove such alleles and accepts only 'exclude' keyword in this field.**

10) **read_seq:** The primary and the secondary barcodes attached to the analyzed DNA contain a sequence of four nucleotides that serve as a sample identifier. The 'read_seq' field lists these identifier sequences, to allow the pipeline distinguish between sample sequences and contaminants from other libraries. Comma is used to separate between primary and secondary barcode identifiers. Vertical bar ("|") is used to separate

variants of the sequence in the same barcode. For the primary barcode sequence it is advisable to include also first three - four nucleotides of the analyzed gene following the identifier to ensure that the barcode was attached to the right DNA sequence (e.g. - ACGTTGT|ACGTAGT,CGTG). **Note:** The pipeline assumes that the order of listed identifiers is always <5' barcode ID>,<3' barcode ID>

11) **read_pos:** This field specifies start position of the identifier sequences listed in the 'read_seq' field, so the pipeline knows where in the read it should look for the identifiers. As in the previous field, comma separates primary and secondary barcode positions and vertical bar separates start positions of the variants in the same barcode. The secondary barcode identifier start position is determined **from the end** of the read, therefore it is expressed as a negative value (E.g. - 15|15,-6).

12) **read_action:** This field specifies what action to take if identifier sequences were found at right positions within the analyzed reads. **The pipeline accepts 'include' keyword to indicate that the reads should be included in further analyses and any other string - to remove them**. The keywords should be listed as a comma separated list, with separate values for primary and secondary barcode identifiers (e.g.: include,include).

13) **sort_pos:** In case that the analyzed DNA originates from multiple genes, this column specifies positions in the read that can be used to sort reads by their origin gene. The positions are listed as a comma separated list, with the semicolon separating identifying positions of different origin gene (e.g. - 63,64,65,66,67,68;63,64,65,66,67,68). Identifying positions of different haplotypes belonging to the same gene should be separated by ampersand ("&").

14) **sort_nucl:** This column specifies which nucleotides should be found at the positions listed in the 'sort_pos' field to consider analyzed read as originating from a specific gene. As in the 'sort_pos' field, identifying nucleotides should be comma separated with semicolon separating data for different genes and ampersand separating haplotypes (e.g. - C,G,T,T,A,C;T,G,T,C,A,A). The order in which identifying nucleotides are listed should match the order of identifying positions listed in the previous field.

15) **sort_refs:** This column specifies names of genes from which the reads originate. These genes serve as a reference against which the reads of matching origin are aligned for mutation search. Names of different genes should be separated by semicolon (e.g. - HBB;HBD). The order in which gene names appear should match the order in which identifying nucleotide lists appear in the previous column.

16) **sort_ref:** This column specifies the name of a default reference against which all reads whose origin couldn't be determined are aligned. This column should contain only a single gene name (e.g. - HBB). It can be one of the genes listed in the previous field or another, unrelated gene.

17) **sort_match:** This column specifies what fraction of nucleotides found at the positions specified by the 'sort_pos' field should match nucleotides listed in the 'sort_nucl' field for the read to be considered as originating from a specific gene.

Preparation of the reference sequence files

To identify mutations in the analyzed sample, the pipeline compares reads against a set of reference genes defined by the user. To prepare reference sequence files for use by the pipeline:

1) Place **all reference sequence files** in a **directory** specified by the 'params_dir_reference' parameter in the 'params_1.sh' file. The pipeline is not designed to search for reference files at any other destination.

2) Match the names of the reference files to the names appearing in the "sort_refs" and "sort_ref" fields of the "factors_table.txt".

3) Make sure that all reference files have **".fa" extension**. The pipeline doesn't recognize reference files with a different extension. Remember to check that OS is not configured to hide file extensions by default and ".fa" is the actual extension of the file!

4) Check that each reference file contains only **a single reference sequence** in FASTA

format. The first line should include sequence name, starting with the “>” symbol (E.g.: >PPIA). The second line should include the sequence itself.

Example: If the “sort_refs” field contains values “HBB;HBD” and “sort_ref” field contains “HBB” value - the reference sequence directory should contain **two files**: “HBB.fa” and “HBD.fa”. Each file should contain **a single reference sequence** of the appropriate gene (or part of it).