MitoHiFi

Marcela Uliano-Silva

May 23, 2023

CONTENTS:

CHAPTER

ONE

SRC

1.1 alignContigs

Align multiple sequence FASTA files with MAFFT.

This script allows the user to concatenate multiple sequence files in FASTA format using the concatenate_contigs() function and to align the multifasta file with the mafft_align() function.

alignContigs.**concatenate_contigs**(*contigs_list*, *out_file='all_mitogenomes.rotated.fa'*)

alignContigs.**mafft_align**(*multifasta_file*, *threads='1'*, *out_file='all_mitogenomes.rotated.aligned.aln'*, *clustal_format=False*)

1.2 circularizationCheck

circularizationCheck.**circularizationCheck**(*resultFile*, *contig_id*, *circularSize=220*, *circularOffSet=40*)

Check, with blast, if there is a match between the start and the end of a sequence. Returns a tuple with (True, start, end) or False, accordingly.

circularizationCheck.**get_circo_mito**(*contig_id*, *circular_size*, *circular_offset*)

It takes a contig ID and circularizes it, returning the circularization points.

Parameters

- **contig_id** (*str*) – ID from contig to be circularized
- **circular_size** (*int*) – size to consider when checking for circularization

- **circular_offset** (*int*) – offset from start and finish to consider when looking for circularization

Returns
circularization history for the input contig

Return type
list

1.3 cleanUpCWD

This script cleans up the working directory after MitoHiFi finishes running.

During the cleanup process, intermediate files are either deleted or moved to the proper subdirectories in order to better organize the final files.

cleanUpCWD.**clean_up_work_dir**(*contigs_list*)

cleanUpCWD.**main**()

1.4 compareGenesLists

This script allows the user to compare two list of genes and return the set of genes that are either i) shared by both lists; ii) present in only one list; iii) present only in the other list.

The comparison of the lists is done with the compare_genes_dicts() function. The get_genes_counts() function works to count the number of occurrence of each gene in a list, and the get_clean_gene() function returns the "clean" string represenation of a gene's name (one final clean representation is needed to keep the compare_genes_dicts() from considering the same gene written in two different ways as different genes -> e.g. *cytb* and *cob*.

compareGenesLists.**compare_genes_dicts**(*genes1*, *genes2*, *alphabetically_sorted=False*)

Takes two lists of genes and return genes shared and specific.

compareGenesLists.**get_clean_gene**(*in_gene*)

Takes a raw gene name and returns its clean format.

compareGenesLists.**get_genes_counts**(*genes_list*)

Takes a list of genes and returns a dictionary containing the name of the gene and the number of occurrences.

1.5 createCoveragePlot

This script builds a plot (*coverage_plot.png*) containing the mean coverage depth distribution for the final representative mitogenom and other assembled potential mito contigs.

createCoveragePlot.**create_coverage_plot**(*mapped_contigs*, *winSize*, *repr_contig*, *is_final_mito=False*)

createCoveragePlot.**get_contigs_headers**(*in_fasta*)

Takes a multifasta file and returns a dictionary with the contigs_ids as keys and the contigs headers as values

createCoveragePlot.**get_contigs_to_map**()

Iterates over *contigs_stats.tsv* to list all potential contigs and returns a list with the filenames of their rotated FASTA files

createCoveragePlot.**map_final_mito**(*in_reads*, *threads=1*, *covMap=20*)

Parameters

- **in_reads** (*list*) – reads file to be mapped against contigs
- **threads** (*int*) – number of threads to be used for computation
- **covMap** (*int*) – minimum mapping quality to filter reads when building final coverage plot

Returns
(str) Filename of the sorted mapping file in BAM format

createCoveragePlot.**map_potential_contigs**(*in_reads*, *contigs*, *threads=1*, *covMap=20*)

Parameters

- **in_reads** (*list*) – reads file to be mapped against contigs
- **contigs** (*list*) – list of contigs FASTA files to concatenate and map the reads against
- **threads** (*int*) – number of threads to be used for computation
- **covMap** (*int*) – minimum mapping quality to filter reads when building final coverage plot

Returns
(str) Filename of the sorted mapping file in BAM format

createCoveragePlot.**merge_images**(*img_list*, *out_file*)

createCoveragePlot.**split_mapping_by_contig**(*all_contigs_mapping*, *contigs_headers*, *threads=1*)

Takes a mapping file with reads mapped to a multifasta file and creates individual mapping files for each contig from the *contigs_headers* dictionary. *contigs_headers* contains contigs_ids as values and contigs headers as values

1.6 fetch

This script allows the calculation of statistics over MitoHiFi execution.

The *get_num_seqs* is used to calculate the number of sequences in a FASTA file. The *get_ref_tRNA* allows the definition of the reference tRNA to be used to rotate the potential mito contigs.

fetch.**get_num_seqs**(*in_fasta*)

Gets the number of sequences in a FASTA file.

Parameters
in_fasta (*str*) – input FASTA file

Returns
number of sequences in FASTA file

Return type
int

fetch.**get_ref_tRNA**()

Defines the reference tRNA to be used for rotating contigs.

Returns
reference tRNA

Return type
str

1.7 fetch_mitos

This script allows the calculation of statistics over MitoHiFi execution. It is analogous to the *fetch.py*, except the *fetch.py* is suitable to process files when *MitoFinder* was used for annotation (default) and *fetch_mitos* is used when *MITOS* was chosen as the annotation tool.

The *get_num_seqs* is used to calculate the number of sequences in a FASTA file. The *get_ref_tRNA* allows the definition of the reference tRNA to be used to rotate the potential mito contigs.

fetch_mitos.**get_num_seqs**(*in_fasta*)

Gets the number of sequences in a FASTA file.

Parameters
in_fasta (*str*) – input FASTA file

Returns
number of sequences in FASTA file

Return type
int

fetch_mitos.**get_ref_tRNA**()

Defines the reference tRNA to be used for rotating contigs.

Returns
reference tRNA

Return type
str

1.8 filterfasta

Copyright 2014-2019 Felix Heeger This script is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

filterfasta.**filterFasta**(*inStream*, *outPath*, *minLength=None*, *idList=None*, *random=None*, *fastq=False*, *regex=False*, *neg=False*, *log=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>*)

filterfasta.**filterLengthIdList**(*inStream*, *outPath*, *format*, *minLength=None*, *idList=None*, *regex=False*, *neg=False*, *log=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>*)

filterfasta.**sampleRandom**(*inStream*, *outPath*, *format*, *number*, *log*)

1.9 findFrameShifts

This script iterates over a multifasta sequence file and returns all proteins that contains stop codons in the middle of their sequences.

The find_frameshifts() function is used for the (default) annotation using *MitoFinder*, while find_frameshifts_mitos() is used when *MITOS* was selected as the annotation tool.

findFrameShifts.**find_frameshifts**(*in_gb*)

findFrameShifts.**find_frameshifts_mitos**(*in_proteins_fasta*)

Takes a multifasta protein file and returns information on proteins that contain frameshifts.

findFrameShifts.**get_gb_stats**(*in_gb*)

findFrameShifts.**get_mitos_stats**(*in_gff*, *in_fasta*)

Takes in a GFF file produced by MITOS and returns annotation statistics.

findFrameShifts.**main**()

1.10 findMitoReference

Copyright 2022 Ksenia Krasheninnikova This script is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

findMitoReference.**find_full_mito**(*group*, *outfolder*, *length_threshold*, *considered*, *org_type='mitochondrion'*, *n=1*)

findMitoReference.**get_lineage**(*species*)

1.11 find_frameshifts_mitos

This script iterates over a multifasta sequence file and returns all proteins that contains stop codons in the middle of their sequences.

find_frameshifts_mitos.**find_frameshifts_mitos**(*in_proteins_fasta*)

Takes a multifasta protein file and returns information on proteins that contain frameshifts.

1.12 fixContigHeaders

This script fixes headers from contigs given as input to MitoHiFi in contigs mode (*-c*).

The fix removes/changes suffixes from the contigs IDs in case they conflict with suffixes that are given by MitoHiFi while the pipeline runs.

fixContigHeaders.**fix_headers**(*fasta_in*, *fasta_out*)

1.13 fix_MitoFinder_headers

fix_MitoFinder_headers.**fix_header**(*in_gb*, *out_gb*)

1.14 fix_improper_gff

This script checks and removes features that have start position greater than end position.

This is necessary because we have noticed that MITOS sometimes create features that follow this pattern, although it is not compatible with the GFF standard format.

fix_improper_gff.**fix_improper_gff**(*in_gff*)

Takes a GFF file and fixes all features that have start position greater than end position.

1.15 getGenesList

This script returns a list of annotated genes.

It recognizes annotations in either genbank or GFF format.

getGenesList.**get_genes_list**(*in_annotation*, *format='genbank'*)

Takes a GFF/Genbank file and returns a list of annotated genes.

1.16 getMitoLength

This script calculates statistics for the related mitogenome given as input for MitoHiFi.

The get_mito_length() function calculates the total length of the input related mitogenome, while the get_mito_genes() function returns the number of genes in the related mitogenome.

getMitoLength.**get_mito_genes**(*mito_file*)

Returns the number of genes from a mitogenome

Keyword arguments: mito_file – the input mitogenome GENBANK file

getMitoLength.**get_mito_length**(*mito_file*)

Returns the length of a mitogenome

Keyword arguments: mito_file – the input mitogenome FASTA file

getMitoLength.**main**()

1.17 getReprContig

This script chooses between all potential mito contigs the representative one that is going to be considered the final mitogenome.

The get_repr_contig() is the function originally referred to by the main mitohifi.py script. Over the process of choosing the final mitogenome, the get_circularization_info() function is called to check if the contig was circularized and the get_repr_contig_info() function calculates some other metrics for all potential contigs and does the actual choice of the final mitogenome, retriving its ID and its related stats.

The

getReprContig.**get_circularization_info**(*seq_id*)

Retrieves information if contig was circularized

Parameters
seq_id (*str*) – identifier of the target sequence (contig)

Returns
returns True if contig was circularized and False otherwise

Return type
bool

getReprContig.**get_repr_contig**(*contigs_fasta*, *rel_mito_len*, *rel_mito_num_genes*, *threads='1'*, *debug=False*)

Gets representative contig from a multifasta file

Parameters

- **contigs_fasta** (*str*) – file containing all sequences
- **threads** (*str*) – number of threads to be used when running CDHIT

Returns
Representative contig ID, CDHIT cluster where the representative contig came from

Return type
tuple

getReprContig.**get_repr_contig_info**(*cdhit_clstr_file*, *rel_mito_len*, *rel_mito_num_genes*, *rel_mito_perc=0.35*, *debug=False*)

1.18 get_depth

get_depth.**get_depth**(*bam_file*, *genome_file*, *target_seq*)

get_depth.**get_windows_depth**(*windows_file*, *bam_file*, *target_seq*)

1.19 get_mitos_stats

get_mitos_stats.**get_mitos_stats**(*in_gff*, *in_fasta*)

Takes in a GFF file produced by MITOS and returns annotation statistics.

1.20 gfa2fa

Converts genome from GFA to FASTA format.

gfa2fa.**gfa2fa**(*gfa_input*, *fasta_output=''*)

1.21 gff_to_gbk

Convert a GFF and associated FASTA file into GenBank format.

Usage:

gff_to_genbank.py <GFF annotation file> [<FASTA sequence file> <molecule type>]

<FASTA sequence file>: input sequences matching records in GFF. Optional if sequences are in the GFF
<molecule type>: type of molecule in the GFF file. Defaults to DNA, the most common case.

gff_to_gbk.**main**(*gff_file*, *fasta_file=None*, *molecule_type='DNA'*)

1.22 make_genome

This script creates a genome file compatible with bedtools

The make_genome_file() function creates the genome file itself, while the make_genome_windows() function creates a file containing windows of a given size (win_size) for each sequence from the genome.

make_genome.**make_genome_file**(*in_bam*, *target_seq*)

make_genome.**make_genome_windows**(*genome_file*, *win_size*)

1.23 mitohifi

mitohifi.**main**()

1.24 parallel_annotation

This script circularizes, annotates and rotates mito contigs using MitoFinder for annotation.

The process_contig() function does the circularization, i.e. removal of artifactual repeated sequences at both ends of the contig and the annotation, i.e. gene prediction. The process_contig_02() function rotates the contig, given a reference gene that will be set as the beginning of the sequence. This function also calculates statistics for the contig, which are saved to a file named *{contig_id}.individual.stats*.

In the context of the mitohifi.py script, both functions are usually run in parallel for each potential mito contig. The process_contig_02() function will only be called after process_contig() is run for all potential contigs.

parallel_annotation.**process_contig**(*threads_per_contig*, *circular_size*, *circular_offset*, *contigs*, *max_contig_size*, *rel_gbk*, *gen_code*, *contig_id*)

Circularize and annotate a contig.

Parameters

- **threads_per_contig** (*int*) – number of threads to be used
- **circular_size** (*int*) – size to consider when checking for circularization
- **circular_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)
- **max_contig_size** (*int*) – maximum contig size allowed
- **rel_gbk** (*str*) – filename of related mito genbank file
- **gen_code** (*str*) – species genetic code
- **contig_id** (*str*) – target contig ID

Returns

None

parallel_annotation.**process_contig_02**(*ref_tRNA*, *threads_per_contig*, *circular_size*, *circular_offset*, *contigs*, *max_contig_size*, *rel_gbk*, *gen_code*, *contig_id*)

Rotate a contig related to a reference tRNA gene and calculate contig statistics.

Parameters

- **ref_tRNA** (*str*) – tRNA gene to be used as reference for rotation (contig starts at reference tRNA)
- **threads_per_contig** (*int*) – number of threads to be used
- **circular_size** (*int*) – size to consider when checking for circularization
- **circular_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)
- **max_contig_size** (*int*) – maximum contig size allowed
- **rel_gbk** (*str*) – filename of related mito genbank file
- **gen_code** (*str*) – species genetic code
- **contig_id** (*str*) – target contig ID

Returns
None

1.25 parallel_annotation_mitos

This script circularizes, annotates and rotates mito contigs using MITOS for annotation.

The process_contig_mitos() function does the circularization, i.e. removal of artifactual repeated sequences at both ends of the contig and the annotation, i.e. gene prediction. The process_contig_02_mitos() function rotates the contig, given a reference gene that will be set as the beginning of the sequence. This function also calculates statistics for the contig, which are saved to a file named *{contig_id}.individual.stats*.

In the context of the mitohifi.py script, both functions are usually run in parallel for each potential mito contig. The process_contig_02_mitos() function will only be called after process_contig_mitos() is run for all potential contigs.

parallel_annotation_mitos.**process_contig_02_mitos**(*ref_tRNA*, *threads_per_contig*, *circular_size*, *circular_offset*, *contigs*, *max_contig_size*, *rel_gbk*, *gen_code*, *contig_id*)

Rotate a contig related to a reference tRNA gene and calculate contig statistics.

Parameters

- **ref_tRNA** (*str*) – tRNA gene to be used as reference for rotation (contig starts at reference tRNA)
- **threads_per_contig** (*int*) – number of threads to be used
- **circular_size** (*int*) – size to consider when checking for circularization
- **circular_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)
- **max_contig_size** (*int*) – maximum contig size allowed
- **rel_gbk** (*str*) – filename of related mito genbank file
- **gen_code** (*str*) – species genetic code
- **contig_id** (*str*) – target contig ID

Returns
None

parallel_annotation_mitos.**process_contig_mitos**(*threads_per_contig*, *circular_size*, *circular_offset*, *contigs*, *max_contig_size*, *rel_gbk*, *gen_code*, *refseq_db*, *contig_id*)

Circularize and annotate a contig.

Parameters

- **threads_per_contig** (*int*) – number of threads to be used
- **circular_size** (*int*) – size to consider when checking for circularization
- **circular_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)
- **max_contig_size** (*int*) – maximum contig size allowed

- **rel_gbk** (*str*) – filename of related mito genbank file
- **gen_code** (*str*) – species genetic code
- **contig_id** (*str*) – target contig ID
- **refseq_db** (*str*) – refseq database to be used

Returns
None

1.26 parse_blast

Copyright 2020 Marcela Uliano-Silva This script is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

parse_blast.**get_contigs_ids**(*blast_output*)

args: blast_line is a blast output, that can be either parsed_blast.txt or parsed_blast_all.txt

returns: The ID from each contig from blast_output, i.e., the BLAST queries

parse_blast.**main**()

parse_blast.**parse_blast**(*query_perc=50*, *min_query_perc=80*, *max_query_len=5*)

1.27 plot_annotation

This script creates a plot with all the genes annotated in the mito contigs

The plot_annotation() function is called when the input contig was annotated with MitoFinder and the plot_annotation_mitos() function when the annotation was done with MITOS.

The merge_images() function concatenates the plots generated for all potential mito contigs into a single image.

class plot_annotation.**MyCustomTranslator**(**args: Any*, ***kwargs: Any*)

Bases: BiopythonTranslator

compute_filtered_features(*features*)

class plot_annotation.**MyCustomTranslatorMitos**(**args: Any*, ***kwargs: Any*)

Bases: BiopythonTranslator

compute_feature_label(*feature*)

compute_filtered_features(*features*)

plot_annotation.**merge_images**(*img_list*, *out_file*)

plot_annotation.**plot_annotation**(*in_gb*, *out_gb=None*)

plot_annotation.**plot_annotation_mitos**(*in_gff*, *out_gff=None*)

1.28 plot_annotation_GFF

class plot_annotation_GFF.**MyCustomTranslator**(**args: Any*, ***kwargs: Any*)

Bases: BiopythonTranslator

compute_feature_label(*feature*)

compute_filtered_features(*features*)

plot_annotation_GFF.**merge_images**(*img_list*, *out_file*)

plot_annotation_GFF.**plot_annotation**(*in_gff*, *out_gff=None*)

1.29 plot_coverage

This script supports the creation of an image showing the distribution of the sequencing depth over the final mito contigs.

The make_genome_file() function creates a genome file compatible with bedtools.

The make_genome_windows() function creates a file in bed format containing the coordinates of all sequence windows (size of window is given as input).

The get_windows_depth() function does the actual calculation of the mean depth for each sequence window, which are the values that will be used to create the depth image.

The final_mitogenome_coverage() function moves intermediate files created while building the depth image to a directory named final_mitogenome_coverage/. This is for better organization of output files.

The plot_coverage() function creates the actual depth image based on the mean depths calculated for each sequence window.

plot_coverage.**get_windows_depth**(*windows_file*, *bam_file*)

plot_coverage.**main**()

plot_coverage.**make_genome_file**(*contig_id*)

plot_coverage.**make_genome_windows**(*genome_file*, *winSize*)

plot_coverage.**move_intermediate_files**(*files_list*)

plot_coverage.**plot_coverage**(*contig_id*, *depth_file*, *winSize*, *isFinalMito=False*)

1.30 plot_coverage_final_mito

This script supports the creation of an image containing the depth distribution over the final mitogenome.

plot_coverage_final_mito.**get_windows_depth**(*windows_file*, *bam_file*)

plot_coverage_final_mito.**main**()

plot_coverage_final_mito.**make_genome_file**(*contig_id*)

plot_coverage_final_mito.**make_genome_windows**(*genome_file*, *winSize*)

plot_coverage_final_mito.**move_intermediate_files**(*files_list*)

plot_coverage_final_mito.**plot_coverage**(*contig_id*, *depth_file*, *winSize*, *isFinalMito=False*)

1.31 reverse_complement

This script creates the reverse complement of a sequence.

The reverse_complement() function is the default function to create the reverse complement. The reverse_complement_mitos() function is the one that should be used when dealing with MITOS annotations. The reverse_complement_annotation() function adjusts the annotation coordinates of features to match the ones from the reverse complemented sequence.

reverse_complement.**reverse_complement**(*in_gb*, *out_gb*)

reverse_complement.**reverse_complement_annotation**(*mitogenome_annotation*, *mitogenome_fasta*, *mitogenome_annotation_rc*)

reverse_complement.**reverse_complement_mitos**(*in_fasta*, *out_fasta*)

1.32 rotate_genbank

This script updates the annotation coords of features to match the rotated version of the contig (i.e. where the reference gene is set as the start)

rotate_genbank.**get_feat_info**(*feat*)

rotate_genbank.**rotate_genbank**(*in_gbk*, *ref_gene*, *out_gbk*)

1.33 rotation

This script rotates a contig setting a reference gene as the start.

rotation.**annotate**(*workdir*, *path*, *ref_gb*, *contig_id*, *o_code*, *max_contig_size*, *threads*)

Annotate reverse complemented genome.

rotation.**get_phe_pos**(*path*)

Gets the position of the tRNA-Phe gene

Parameters
path (*str*) – path of input genbank file

Returns
Start position of tRNA-Phe, Strand

Return type
tuple

rotation.**get_trna_pos**(*path*)

Gets the position for each tRNA in an input file

Parameters
path (*str*) – input file to be processed

Returns
Positions for each tRNA gene found

Return type
dict

rotation.**make_rc**(*path*, *rc_path*)

Creates a reverse complement.

Parameters

- **path** (*str*) – input FASTA file
- **rc_path** (*str*) – new FASTA file created (reverse complement of input)

Returns
None

rotation.**rotate**(*genome*, *start*, *contig_id*)

Creates new genome file (suffix .mitogenome.rotated.fa) after rotating it.

Parameters

- **genome** (*str*) – input genome file
- **start** (*int*) – position at which rotate the input genome
- **contig_id** (*str*) – identifier representing the original contig

Returns
None

1.34 rotation_mitos

This script rotates a contig setting a reference gene as the start.

It is equivalent to the rotation.py script, but adjusted to deal with MITOS annotations.

rotation_mitos.**annotate**(*workdir*, *path*, *ref_gb*, *contig_id*, *o_code*, *max_contig_size*, *threads*)

Annotate reverse complemented genome.

rotation_mitos.**get_phe_pos**(*path*)

Gets the position of the tRNA-Phe gene

Parameters
path (*str*) – path of input genbank file

Returns
Start position of tRNA-Phe, Strand

Return type
tuple

rotation_mitos.**get_trna_pos**(*path*)

Gets the position for each tRNA in an input file

Parameters
path (*str*) – input file to be processed

Returns
Positions for each tRNA gene found

Return type
dict

rotation_mitos.**make_rc**(*path*, *rc_path*)

Creates a reverse complement.

Parameters

- **path** (*str*) – input FASTA file
- **rc_path** (*str*) – new FASTA file created (reverse complement of input)

Returns
None

rotation_mitos.**rotate**(*genome*, *start*, *contig_id*)

Creates new genome file (suffix .mitogenome.rotated.fa) after rotating it.

Parameters

- **genome** (*str*) – input genome file
- **start** (*int*) – position at which rotate the input genome
- **contig_id** (*str*) – identifier representing the original contig

Returns
None

rotation_mitos.**rotate_annotation**(*mitogenome_annotation*, *mitogenome_fasta*, *start*, *contig_id*)

Takes a GFF file and the reference gene position and rotates the annotation to have the reference gene at position 1.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

M

P

R

S