seq-tools Documentation

Release 0.2.0

Jason L Weirather

January 29, 2017

Contents

Contents:

CHAPTER 1

seqtools package

Subpackages

seqtools.cli package

Subpackages

seqtools.cli.utilities package

Submodules

seqtools.cli.utilities.bam_bgzf_index module The bam bgzf utility produces gzipped index of a bam file with the following fields for each line of the bam file (not counting header)

1. Query name
2. Target range
3. BlockStart
4. InnerStart
5. Aligned Base Count
6. Flag

This index can be used to provide random access into a bam file

class seqtools.cli.utilities.bam_bgzf_index.**Queue**(*val*)

get()

seqtools.cli.utilities.bam_bgzf_index.**do_chunk**(*coords*, *ecount*, *args*)

seqtools.cli.utilities.bam_bgzf_index.**do_inputs**()

seqtools.cli.utilities.bam_bgzf_index.**external_cmd**(*cmd*)

seqtools.cli.utilities.bam_bgzf_index.**main**(*args*)

seqtools.cli.utilities.bam_bgzf_index.**setup_tempdir**(*args*)

seqtools.cli.utilities.bam_to_bed_depth module Convert a BAM or a SAM into a bed depth file

The file is a TSV format with the fields

1. Chromosome
2. Start (0-index)
3. End (1-index)
4. Read depth

The file is ordered and covers all regions covered by alignments

seqtools.cli.utilities.bam_to_bed_depth.**do_inputs**()

seqtools.cli.utilities.bam_to_bed_depth.**do_output**(*outputs*)

seqtools.cli.utilities.bam_to_bed_depth.**external_cmd**(*cmd*)

seqtools.cli.utilities.bam_to_bed_depth.**get_output**(*bedarray*, *z*)

seqtools.cli.utilities.bam_to_bed_depth.**main**(*args*)

seqtools.cli.utilities.bam_to_bed_depth.**setup_tempdir**(*args*)

seqtools.cli.utilities.fasta_to_fake_fastq module Convert a FASTA file into a FASTQ file. You can designate what to include in the quality score by setting the –ascii paramater (default 'I')

seqtools.cli.utilities.fasta_to_fake_fastq.**do_inputs**()

seqtools.cli.utilities.fasta_to_fake_fastq.**external_cmd**(*cmd*)

seqtools.cli.utilities.fasta_to_fake_fastq.**main**(*args*)

seqtools.cli.utilities.fasta_to_tsv module Convert a FASTA file into a TSV with the fields

1. Header
2. Sequence

The Header cannot contain tabs, and any linebreaks in the sequence will be lost

seqtools.cli.utilities.fasta_to_tsv.**do_inputs**()

seqtools.cli.utilities.fasta_to_tsv.**external_cmd**(*cmd*)

seqtools.cli.utilities.fasta_to_tsv.**main**(*args*)

seqtools.cli.utilities.fastq_to_fasta module Convert a FASTQ file to a FASTA

seqtools.cli.utilities.fastq_to_fasta.**do_inputs**()

seqtools.cli.utilities.fastq_to_fasta.**external_cmd**(*cmd*)

seqtools.cli.utilities.fastq_to_fasta.**main**(*args*)

seqtools.cli.utilities.fastq_to_tsv module Convert a FASTQ to a TSV with the following fields

1. Header (without the '@' prepending)
2. Sequence
3. Line 3 (still has the '+' sign)

4. Quality

lines cannot contain tabs

seqtools.cli.utilities.fastq_to_tsv.**do_inputs**()

seqtools.cli.utilities.fastq_to_tsv.**external_cmd**(*cmd*)

seqtools.cli.utilities.fastq_to_tsv.**main**(*args*)

seqtools.cli.utilities.trim_fasta module left or right trim a FASTA file (option ot invert the trim to keep the ends)

seqtools.cli.utilities.trim_fasta.**do_inputs**()

seqtools.cli.utilities.trim_fasta.**external_cmd**(*cmd*)

seqtools.cli.utilities.trim_fasta.**main**(*args*)

seqtools.cli.utilities.trim_fastq module Trim a FASTQ file/stream ends of all entries (option to invert and only keep the ends)

seqtools.cli.utilities.trim_fastq.**do_inputs**()

seqtools.cli.utilities.trim_fastq.**external_cmd**(*cmd*)

seqtools.cli.utilities.trim_fastq.**main**(*args*)

seqtools.cli.utilities.tsv_to_fasta module Undo the fasta_to_tsv command and put it back in fasta format

seqtools.cli.utilities.tsv_to_fasta.**do_inputs**()

seqtools.cli.utilities.tsv_to_fasta.**external_cmd**(*cmd*)

seqtools.cli.utilities.tsv_to_fasta.**main**(*args*)

seqtools.cli.utilities.tsv_to_fastq module Undo the fastq_to_tsv command and put it back in fastq format

seqtools.cli.utilities.tsv_to_fastq.**do_inputs**()

seqtools.cli.utilities.tsv_to_fastq.**external_cmd**(*cmd*)

seqtools.cli.utilities.tsv_to_fastq.**main**(*args*)

Module contents

Submodules

seqtools.cli.cli_front module

The cli_front is a the command line utility that is used to list all the accessable command line utilities and to call the command line utility you want to run.

seqtools.cli.cli_front.**do_args**()

seqtools.cli.cli_front.**main**()

Module contents

seqtools.format package

Submodules

seqtools.format.bamindex module

bamindex class describes a custom index format used by AlignQC

class seqtools.format.bamindex.**BAMIndex**(*index_file*)

Index file is a gzipped TSV file with these fields:

1.qname

2.target range

3.bgzf file block start

4.bgzf inner block start

5.aligned base count

6.flag

Usage:

name_to_num is used to get all the names at random get_longest_target_alignment_coords_by_name is used to get the best random coord hash is import for random access There are some inactive methods because the datastructures they needed were not getting used and were memory intensive. subsequent updates could put them back or even better only use them when the methods requring them are called the first time This class is actually incredibly bulky for working with a big index > 1M reads. I think some more specific cases may need to be written

Parameters **index_file** (*string*) – filename (of the gzipped index file)

check_ordered()

True if each chromosome is listed together as a chunk and if the range starts go from smallest to largest otherwise false

Returns is it ordered?

Return type bool

destroy()

Try to clear memory up by setting values to None

get_coord_line_number(*coord*)

return the one-indexed line number given the coordinates

get_coords_by_name(*name*)

> **Warning:** not implemented

get_index_line(*lnum*)

Take the 1-indexed line number and return its index information

get_length()

number of indexed entries

get_longest_target_alignment_coords_by_name(*name*)

For a name get the best alignment

Returns[filebyte,innerbyte] describing the to distance the zipped block start, and the distance within the unzipped block

Return typelist

get_names()
get all the query names

get_range_start_coord(*rng*)

> **Warning:** not implemented

get_range_start_line_number(*rng*)

> **Warning:** not implemented

get_unaligned_lines()
get the lines that are not aligned

get_unaligned_start_coord()

> **Warning:** not implemented

class seqtools.format.bamindex.**BAMIndexRandomAccessPrimary**(*index_file=None*, *alignment_file=None*, *verbose=False*)

The best index class will read an index file and only provide accessto primary alignment coordinates

Parameters

- **index_file** (*string*) – the bam index file
- **alignment_file** (*string*) – the bam alignment file
- **verbose** (*bool*) – more visual output

destroy()

get_random_coord()

seqtools.format.bamindex.**check_flag**(*flag*, *inbit*)

seqtools.format.bamindex.**write_index**(*path*, *index_file*, *verbose=False*, *samtools=False*)
Index file is a gzipped TSV file with these fields:

1.qname

2.target range

3.bgzf file block start

4.bgzf inner block start

5.aligned base count

6.flag

Parameters

- **path** – bamfile
- **index_file** – bam index to write

- **verbose** (*bool*) – default False
- **samtools** (*bool*) – use samtools default False

seqtools.format.bed module

class seqtools.format.bed.**Bed12**(*bed_line*)
Bases: *seqtools.structure.Transcript*

Bed format with 9 optional fields

Parameters **bed_line** (*string*) – one line of a bed file

get_bed_line()
get the bed line

get_line()
get the bed line

value(*key*)
access the bed line by key

Parameters **key** (*string*) – which attribute of the bed12

seqtools.format.bgzf module

seqtools.format.bgzf.**get_block_bounds**(*filename*)

Pre block starts start 0-indexted, end 1-indexted

Parameters **filename** (*string*) – filename

Returns 0-index start and 1-index end

Return type array of arrays with the [start end] of each block

seqtools.format.bgzf.**is_bgzf**(*filename*)
Pre: filename to test if it is a bgzf format

Post: True or False

Parameters **filename** (*string*) –

Returns if its a bgzf

Return type bool

class seqtools.format.bgzf.**reader**(*handle*, *blockStart=None*, *innerStart=None*)
Methods adapted from biopython's bgzf.py

(optional) blockStart is the byte start location of a block (optional) innerStart says how far into a decompressed bock to start

Parameters

- **handle** (stream) –
- **blockStart** (*int*) – start from here (optional)
- **innerStart** (*int*) – start from here (optional)

get_block_start()

get_inner_start()

read(*size*)
read size bytes and return them

seek(*blockStart*, *innerStart*)

class seqtools.format.bgzf.**writer**(*handle*)
Give it the handle of the stream to write to

close()

write(*bytes*)

seqtools.format.fasta module

class seqtools.format.fasta.**FASTA**(*fasta_text*)
Bases: *seqtools.sequence.Seq*

FASTA()

class seqtools.format.fasta.**FASTAData**(*data=None*, *file=None*, *dict=None*)
Slicable fast fasta It loses any additional header information in fasta header only the first non-whitespace is what we use

Parameters

- **data** (*bytes*) – bytes of the fasta file
- **file** (*string*) – filename
- **dict** (*dict()*) – dictionary of chromosomes

clear()

get_sequence(*chr=None*, *start=None*, *end=None*, *dir=None*, *rng=None*)
get a sequence

Parameters

- **chr** (*string*) –
- **start** (*int*) –
- **end** (*int*) –
- **dir** (*char*) – charcter +/-

Parma rng

Returnssequence

Return typestring

keys()

remove(*key*)

class seqtools.format.fasta.**FASTAFile**(*fname*, *index=None*)
Do random access with an indexed Fasta File Creates the index if its not there already

Pre: An uncompressed fasta file Can be called by chromosome and location slices Slices are same as array - zero indexed

Post: Makes index if doesn't exist upon being called. Can access sequence

Modifies: File IO reads the fasta, and writes a fasta index file

> **Warning:** this uses a subclass. probably should avoid that

class **Chromosome**(*outer*, *chr*)

FASTAFile.**get_sequence**(*chr=None*, *start=None*, *end=None*, *dir=None*, *rng=None*)

class seqtools.format.fasta.**FASTAStream**(*fh*, *custom_buffer_size=10000000*)
Iterable Stream

Parameters

- **fh** (stream) – file handle
- **custom_buffer_size** (*int*) – default size (10000000)

get_entry()

next()

seqtools.format.fastq module

class seqtools.format.fastq.**FASTQ**(*v*)
Bases: *seqtools.sequence.Seq*

fastq single entry

Parametersv (*string*) – one entry

FASTQ()

copy()

rc()

class seqtools.format.fastq.**FASTQStream**(*fh*)
Iterable Stream

get_entry()

next()

seqtools.format.gpd module

class seqtools.format.gpd.**GPD**(*gpd_line*)
Bases: *seqtools.structure.Transcript*

This whole format is a subclass of the Transcript subclass

Parametersgpd_line (*string*) –

exons

get_gpd_line()
output the original gpd line Overrides Structure.Transcript

get_line()

get_range()
override, we are garunteed to have the range since we initialize on reading a line

junctions

value(*key*)

class seqtools.format.gpd.**GPDStream**(*fh*)
Iterate over GPD entries

next()

read_entry()

class seqtools.format.gpd.**SortedOutputFile**(*filename*, *type='location'*, *tempdir=None*)
a stream to write to for outputing a sorted file

Parameters

- **filename** (*string*) – output file
- **type** (*string*) – how to sort (default location)
- **tempdir** (*string*) –

close()

write(*value*)

seqtools.format.psl module

Classes to work with the psl format

class seqtools.format.psl.**PSL**(*psl_line*, *reference=None*, *query_sequences=None*, *query_sequence=None*, *query_quality=None*)
Bases: *seqtools.align.Alignment*

Class to define a psl line

Parameters

- **psl_line** (*string*) – is a psl formated line
- **reference** – a dict/slice accessable sequences
- **query_sequences** (*dict()*) – a dict/slice accessable sequences
- **query_sequence** (*string*) – just the string that is the query sequence

class PrivateValues
This class was an attempt at creating closures for some values. It may be overkill for what we are doing, or worse, it may be slow. Values from the orginal should just be accessed though functions for consistancy sake. This class should remind us well that entires need to be accessed this way

get_entry(*key*)

is_entry_key(*key*)

set_entries_dict(*mydict*)

set_entry(*key*, *value*)

PSL.**get_PSL**()
Overrides parent to make the PSL generation just return self

PSL.**get_line**()

PSL.**get_query_length**()
overrides parent to get the query length

PSL.**get_query_quality**()

PSL.**get_query_sequence**()
Do our overrides parent to get query sequence

Returnsquery sequence

Return typestring

PSL.**get_reference**()
overrides parent to get the reference genome dict()

PSL.**get_strand**()
same as direction

Returnsstrand + or -

Return typechar

PSL.**value**(*key*)
Access spefific attributes of the PSL by key name. Here is how we access value by keys

Parameters**key** (*string*) – which attribute of the PSL to get

seqtools.format.sam module

Classes to work with sam and bam files

class seqtools.format.sam.**BAM**(*bin_data*, *ref_names*, *fileName=None*, *blockStart=None*, *innerStart=None*, *ref_lengths=None*, *reference=None*, *line_number=None*)
Bases: *seqtools.format.sam.SAM*

Very much like a sam entry but optimized for access from a bam Slows down for accessing things that need more decoding like sequence, quality, cigar string, and tags

Warning: Having the reference names and the reference sizes we may save some time by not reading the header each time we access the file. Theres probably a more efficent coarse to go by defining a bamfile object and having a BAMline entry being the extension of sam, and drawing most of this stuff from the bam file

Parameters

- **bin_data** (*bytes*) – byte data for just a single bam entry (seems unnecessary since we have the file)
- **ref_names** (*list of names*) – array of refernece names
- **fileName** – the bam file name
- **blockStart** (*where to begin in the file*) –
- **innerStart** (*where to begin in the decompressed block*) –
- **ref_lengths** (*dict()*) – seems unncessary to take the reference lengths because we can always get that from the header
- **reference** (*dict()*) –
- **line_number** (*int*) –

get_alignment_ranges()
return the basics for defining an alignment

get_block_start()

get_cigar()
produce the cigar in list form

ReturnsCigar list of [value (int), type (char)] pairs

Return typelist

get_coord()
get the current coordinate

Returns[blockStart, innerStart]

Return typelist is a pair [int, int]

get_file_position_string()

get_filename()

get_inner_start()

get_line_number()

get_tag(*key*)
retrieve the value of a single tag by its key.

> **Warning:** Not sure if it accommodates multiple of the same keys

get_target_length()
length of the entire chromosome

value(*key*)
Access basic attributes of BAM by key

class seqtools.format.sam.**BAMFile**(*filename*, *blockStart=None*, *innerStart=None*, *cnt=None*, *reference=None*)
iterable class to open and access a bam file

Parameters

- **filename** (*string*) –
- **blockStart** (*int*) –
- **innerStart** (*int*) –
- **cnt** (*int*) –
- **reference** (*dict()*) – dictionary of genomic sequences

close()

fetch_by_coord(*coord*)
get a single entry by the coordinate location [blockStart, innerStart]

> **Warning:** creates a new instance of a BAMFile object when maybe the one we had would have worked

fetch_starting_at_coord(*coord*)
starting at a certain coordinate was supposed to make output

> **Warning:** creates a new instance of a BAMFile object when maybe the one we had would have worked

get_header()

next()

read_entry()

read_entry2()

class seqtools.format.sam.**BGZF**(*filename*, *blockStart=None*, *innerStart=None*)
Methods adapted from biopython's bgzf.py

> **Warning:** We already have a BGZF class, i winder why we don't put this there

Parameters

- **filename** (*string*) –
- **blockStart** (*int*) –
- **innerStart** (*int*) –

close()

get_block_start()

get_inner_start()

read(*size*)
Read this many bytes from where you currently are

seek(*blockStart*, *innerStart*)

class seqtools.format.sam.**SAM**(*line*, *reference=None*, *reference_lengths=None*)
Bases: *seqtools.align.Alignment*

Class to define the SAM format.

Parameters

- **line** (*string*) –
- **reference** (*dict()*) –
- **reference_lengths** (*dict() dictionary of chromosome keyed lengths*) –

class PrivateValues
My attempt at closures again. Still think its probably not worth the trouble doing it this way. Bam files need a specific override to get_tags and get_cigar that would break other parts of the class if we access the variables other ways Force tags and cigars to be hidden so we don't accidently change them.

get_cigar()

get_entry(*key*)

get_tags()

is_entry_key(*key*)

set_cigar(*cigar*)

set_entries_dict(*mydict*)

set_entry(*key*, *value*)

set_tags(*tags*)

SAM.**check_flag**(*inbit*)

SAM.**get_SAM**()
Override parent to just return itself

SAM.**get_actual_original_query_range**()
This accounts for hard clipped bases and a query sequence that hasnt been reverse complemented

Returnsthe range covered on the original query sequence

Return type*GenomicRange*

SAM.**get_cigar**()
Get list of cigar data in the form [[value1,char1],[value2,char2]...]

Returnscigar data

Return typelist of [int value, char type] pairs

SAM.**get_line**()
assemble the line if its not there yet

> **Warning:** this should probably not exist if the constructor takes a line

SAM.**get_original_query_length**()
Similar to get_get_query_length, but it also includes hard clipped bases if there is no cigar, then default to trying the sequence

Returnsthe length of the query before any clipping

Return typeint

SAM.**get_query_length**()

SAM.**get_query_quality**()
Overrides align

> **Warning:** this returns the full query quality, not just the aligned portion

SAM.**get_query_sequence**()
Overrides align

> **Warning:** this returns the full query sequence, not just the aligned portion

SAM.**get_range**()
Necessary function for doing a locus stream For the context of a SAM file we set this to be the target range

Returnstarget range

Return type*GenomicRange*

SAM.**get_strand**()
Overrides parent to get direction from the flag

Returnsstrand/direction + or -

Return typechar

SAM.**get_tag**(*key*)
access tags by key and get the value

> **Warning:** Some of the key values may be better returned as numerical when they are. Right now i'm not sure how its implemneted but probably just as a string.

Parameters **key** – type key: string return: key rtype: string

SAM.**get_tags**()
Get all the tags

Returns tag information

Return type dict() of key value attributes

SAM.**get_target_length**()
Get the length of the target sequence. length of the entire chromosome :return: length :rtype: int

SAM.**get_target_range**()
Get the range on the target strand

Returns target range

Return type *GenomicRange*

SAM.**is_aligned**()

SAM.**value**(*key*)

class seqtools.format.sam.**SAMHeader**(*header_text*)
class to retain information about a SAMheader and access data from it

get_sequence_length(*sname*)

get_sequence_lengths()

get_sequence_names()

class seqtools.format.sam.**SAMStream**(*fh=None*, *minimum_intron_size=0*, *minimum_overhang=0*, *reference=None*)
minimum_intron_size greater than zero will only show sam entries with introns (junctions) minimum_overhang greater than zero will require some minimal edge support to consider an intron (junction)

Parameters

- **fh** (stream) – filehandle to go through
- **minimum_intron_size** (*int*) – (default 0)
- **minimum_overhang** – require some minimum edge support to consider a junction. Probably should make more use of this
- **reference** (*dict()*) – dictionary of reference sequences

assign_handle(*fh*)

get_header()
Return the object representing the header

next()

read_entry()

set_junction_only(*mybool=True*)

class seqtools.format.sam.**SamtoolsBAMStream**(*path*, *minimum_intron_size=0*, *minimum_overhang=0*, *reference=None*)
Bases: *seqtools.format.sam.SAMStream*

Stream but use samtools

close()

seqtools.format.sam.**check_flag**(*flag*, *inbit*)
Check a flag is true or false

seqtools.format.sam.**is_header**(*line*)
true if we are in a header

seqtools.format.sam.**is_junction_line**(*line*, *minlen=68*, *minoverhang=0*)

seqtools.format.sam.**sort_header**(*header_text*)
sort the chromosomes in a header text

Module contents

seqtools.simulation package

Submodules

seqtools.simulation.emitter module

These classes should produce simulation products

class seqtools.simulation.emitter.**TranscriptomeEmitter**(*transcriptome*, *seed=None*, *rand=None*)

Give it a transcriptome definition and a reference genome for itinitialy give it uniform probability

Parameters

- **transcriptome** (Transcriptome) – A transcriptome from which to produce transcripts
- **seed** (*int*) – Seeded random generation
- **rand** (RandomSource) – A class that can generate randome numbers if you have one already seeded or want totally random

emit_transcript()
Get a transcript according according to weight of transcript

ReturnsOne random Transcript

Return type*Transcript*

set_weights_by_dict(*weights*)

input: an array of weights <<txname1> <weight1>> <<txname2> <weight2>>...if this does not get set then even weighting will be used

Parameters**weights** (*list*) – [[tx1,wght1],[tx2,wght2],...[txN,wightN]]

seqtools.simulation.permute module

These classes are here to alter simulation products or other sequences

class seqtools.simulation.permute.**MakeCuts**(*rand=None*, *seed=None*)
Class to cut the sequence to different sizes

Parameters

- **rand** (RandomSource) – pass a random source, otherwise it gets a new RandomSource

- **seed** (*int*) – if you want to set a seed here

get_cut(*seq*)

set_custom(*gmin*, *gmu*, *gsigma*)
Set a minimum lengtha, and then the gaussian distribution parameters for cutting For any sequence longer than the minimum the guassian parameters will be used

set_lr_cuts()

set_sr_cuts()

class seqtools.simulation.permute.**MakeErrors**(*rand=None*, *seed=None*)
Class to define how to make errors, and to introduce those errors

random_deletion(*fastq*, *rate*)
Perform the permutation on the sequence

Parameters

- **fastq** (format.fastq.FASTQ) – FASTQ sequence to permute
- **rate** (*float*) – how frequently to permute

ReturnsPermutted FASTQ

Return type*format.fastq.FASTQ*

random_flip(*sequence*)
Change the direction of the sequence with 0.5 probability

random_insertion(*rate*, *max_inserts=1*)
Perform the permutation on the sequence. If authorized to do multiple bases they are done at hte rate defined here.

Parameters

- **fastq** (format.fastq.FASTQ) – FASTQ sequence to permute
- **rate** (*int*) – how frequently to permute
- **max_inserts** – the maximum number of bases to insert (default 1)

ReturnsPermutted FASTQ

Return type*format.fastq.FASTQ*

random_substitution(*fastq*, *rate*)
Perform the permutation on the sequence

Parameters

- **fastq** (format.fastq.FASTQ) – FASTQ sequence to permute
- **rate** (*float*) – how frequently to permute

ReturnsPermutted FASTQ

Return type*format.fastq.FASTQ*

set_after_context(*base*)
Limit errors to a specific following base context

set_before_context(*base*)
Limit errors to a specific preceeding base context

set_modified_base(*base*)
Limit errors to a specific type of sequenced base

set_observed_base(*base*)
Limit errors to a specific reference base

seqtools.simulation.permute.**phred33_to_rate**(*q*)
Convert a phred33 character to an error rate

seqtools.simulation.permute.**random_flip**(*sequence*, *rnum=None*)
Flip a sequence direction with 0.5 probability

seqtools.simulation.permute.**rate_to_phred33**(*rate*)
Convert an error rate to a phred 33 character

seqtools.simulation.randomsource module

A class to aid in generating random numbers and sequences

class seqtools.simulation.randomsource.**RandomSource**(*seed=None*)
You can asign it a seed if you want

Parameters **seed** (*int*) – seed the pseduorandom number generator

choice(*arr*)
Uniform random selection of a member of an list

Parameters **arr** (*list*) – list you want to select an element from

Returns one element from the list

different_random_nt(*nt*)

gauss(*mu*, *sigma*)
Generate a random number based on a gaussian distribution

Parameters

- **mu** (*float*) – mean of distribution
- **sigma** (*float*) – standard deveiation of distribution (i think)

get_weighted_random_index(*weights*)

Return an index of an array based on the weights if a random number between 0 and 1 is less than an index return the lowest index

Parameters **weights** (*list*) – a list of floats for how to weight each index [w1, w2, ... wN]

Returns index

Return type int

randint(*a*, *b*)
Generate a random integer uniform distribution between a and b like randint of the usual random class

Returns random int between a and b

Return type int

random()
generate a random number

Returns uniform random float between 0 and 1

Return type float

random_nt()
Produce a random nucleotide (uniform random)

Returnsnucleotide

Return typechar

Module contents

Submodules

seqtools.align module

This module contains the most basic classes for describing and working with alignments.

class seqtools.align.**Alignment**
Basic class for common elements of alignments. You don't have to have a query sequence and a reference sequence to do an alignment.

construct_cigar(*min_intron_size=68*)
Create a CIGAR string from the alignment

ReturnsCIGAR string

Return typestring

get_PSL(*min_intron_size=68*)
Get a PSL object representation of the alignment.

ReturnsPSL representation

Return type*PSL*

get_SAM(*min_intron_size=68*)
Get a SAM object representation of the alignment.

ReturnsSAM representation

Return type*SAM*

get_actual_query_range()
This is the actual query range for the positive strand

ReturnsRange of query positive strand covered

Return type*GenomicRange*

get_aligned_bases_count()
The sum of the aligned bases.

Returnslength (in base pairs)

Return typeint

get_alignment_ranges()
Return an array of alignment ranges.

get_alignment_strings(*min_intron_size=68*)

Process the alignment to get information likethe alignment strings for each exon. These strings are used by the pretty print.

ReturnsString representation of the alignment in an easy to read format

Return typestring

get_query_length()

> **Warning:** Must be overridden

get_query_quality()
Get the quality.

Returnsquality

Return typestring

get_query_sequence()

> **Warning:** Must be overridden

get_reference()
Return the reference sequence

Returnsreference sequence

Return typestring

get_strand()

> **Warning:** Must be overridden

get_target_length()

> **Warning:** Must be overridden

get_target_range()
Get the range covered on the target/reference strand

ReturnsGenomic range of the target strand

Return type*GenomicRange*

get_target_transcript(*min_intron=1*)
Get the mapping of to the target strand

ReturnsTranscript mapped to target

Return type*Transcript*

print_alignment(*chunk_size=40*, *min_intron_size=68*)
print the nice looking alignment. Must have data accessable from get_query_sequence() and get_refernece_sequencec()

ReturnsPretty print string.

Return typestring

set_query_sequence(*seq*)
Assign the query sequence.

Parameters**seq** (*string*) – sequence of the query

set_reference(*ref*)
Set the reference sequence

Parameters **ref** (*string*) – reference sequence

seqtools.errors module

This module contains classes for analyzing error patterns in alignments

Its in pretty rough shape as its an early, but working, form. It works with alignqc. But it really needs some love to be a good module.

Error Analysis

I am to describe errors at several levels

Errors in the query sequence

1. Is a query base an error or not?
   - Probability - Sometimes it can be ambiguous which base is in error
2. What is the basic type of error?
   - Mismatch
   - **Insertion**
     - Total insertion
     - Homopolymer insertion
     - **Deletion**
       - **Total deletion**
         - Before
         - After
       - Homopolymer deletion
   - sum of probabilities should add up to 1.)
3. What is the more specific error?
   - Mismatch type
   - insertion/deletion - Base, Length

class seqtools.errors.**AlignmentErrors**(*alignment*, *min_intron_size=68*)

Take an alignment between a target and query Uses get_strand from alignment to orient the query All results are on the positive strand of the query (meaning may be the reverse complement of target if negative)

Parameters

- **alignment** (Alignment) – alignment to be used in error calculation
- **min_intron_size** (*int*) – minmum length for an intron

class **HPAGroup**(*parent*, *mydict*)

Homopolymer alignment grouptakes a chunk of homopolymer alignment as a dictionary with 'query' and 'target' sequences set query should always be positive strand

Parameters **mydict** (*dict() {'query':query sequence,'target':target sequence}*) – dictionary with target sequences and a parent object

get_exon()
return the exon number

get_length()
return the lengths of the query and the target
Returnslengths object
Return typedict() with {'query':query length,'target': target length}

get_nt()

get_quality()
get the quality score info or false if we cannot

get_query()
always + strand

get_string()
Describe the group as a string

get_target()
could be + or - strand

has_quality()
Do we have quality score info?

type()

AlignmentErrors.**analyze_quality**()
Go through HPAGroups and store the distro of ordinal values of quality scores

AlignmentErrors.**close**()

AlignmentErrors.**get_HPAGroups**()
get a list of the HPA groups :returns: list of HPA groups :rtype: HPAGroup

AlignmentErrors.**get_context_query_errors**()
A more straitfoward calculation of the context-specific errors relative to the query

Returnsmatrix of observed contexts and values

Return typematrix of [before][after][query]{types} with types being any base or a deletion.

AlignmentErrors.**get_context_target_errors**()
A more straitfoward calculation of the context-specific errors relative to the target

Returnsmatrix of observed contexts and values

Return typematrix of [before][after][reference]{types} with types being any base or a deletion.

AlignmentErrors.**get_general_errors**()
way to accumulate totals of error types General error report will be relative to to the total alignment length error rate = mismatches + insertions + deletions / alignment length

This looks oddly written, probably should be careful not to run it twice because it looks like it would accumulate.

AlignmentErrors.**get_quality_report_string**()
get a report on quality score distribution. currently prints to stdout

AlignmentErrors.**get_query_error**(*i*)
Just get a single error characterization based on the index

Parametersi (*int*) – list index

Returnsbase-wise error

Return typeHPA group description

AlignmentErrors.**get_query_errors**()
Return a list of base-wise error observations for the query

Returnslist of base-wise errors

Return typelist of HPA groups

AlignmentErrors.**get_query_sequence**()
return the query sequence reconstructed from the descriptions

AlignmentErrors.**get_target_error**(*i*)
Just get a single error characterization based on the index relative to the target

Parameters**i** (*int*) – list index

Returnsbase-wise error

Return typeHPA group description

AlignmentErrors.**get_target_errors**()
Just get a single error characterization based on the index relative to the target

Parameters**i** (*int*) – list index

Returnslist of base-wise errors

Return typelist of HPA groups

AlignmentErrors.**get_target_sequence**()
return the target sequence reconstructed from the descriptions

AlignmentErrors.**has_quality**()
Does the current data have quality information?

class seqtools.errors.**BaseError**(*type*)
Class for describing an error at a sinle base relative to the target or query.

class ObservableError(*type*)

Class to describe a homopolymer error or an observableinsertion or deletion. Future versions of this should probably avoid using a nested class for this

Parameters**type** (*string*) – Either 'query' or target

get_attributable_length()
For calculating total error counts

get_changed_length()
How much the homopolymer length differs between target and query
Returnsabs(qlen-tlen)
Return typeint

get_error_probability()
Probability that this base is the product of an error
Returnsprobability
Return typefloat

get_homopolymer()
Return a class to describe the homopolymer
Returnshomopolymer details
Return typedict() return {'tseq':string,'seq':string}

get_query_base()
Just the query base

get_target_base()
Just the target base

get_type()
get the type of the observable error
Returnserror details
Return typelist with 1. main type, 2. subtype, 3. details [target [nucleotide, length],query [nucleotide, length]]

set(*tlen*, *qlen*, *tnt*, *qnt*)
Set the error we are observing for the homopolymer block
Parameters
- **tlen** (*int*) – target homopolymer length
- **qlen** (*int*) – query homopolymer length
- **tnt** (*char*) – target nucleotide
- **qnt** (*char*) – query nucleotide

class BaseError.**UnobservableError**(*type*)
Unobservable error is a deletion for a query base an insertion for a target base A non base error has a probability of occuring before a base and a probability of occuring after

Parameterstype (*string*) – Either 'query' or target

get_after_probability()

get_after_type()

get_attributable_length()

get_before_probability()

get_before_type()

get_error_probability()

set_after(*tlen*, *qlen*, *nt*, *p*)

set_before(*tlen*, *qlen*, *nt*, *p*)

BaseError.**get_adjusted_error_count**()

Get the total error count associated with this single base. This would typically be one but sometimes it may be larger for instertions.

Returnserror_count

Return typefloat

BaseError.**get_base**()
Get the single base at this position.

Returnsbase

Return typechar

BaseError.**get_error_probability**()

This means for the base we are talking about how many errors between 0 and 1 do we attribute to it?
For the 'unobserved' errors, these can only count when one is adjacent to base

Returnserror probability p(error_observed)+(1-p_error_observed)*error_unobserved

Return typefloat

BaseError.**get_homopolymer**()
Return the hompolymer on target and the homopolymer on query assicated with this base

Returnshomopolymer dict {tseq:sequence,qseq:sequence}

Return typedict()

BaseError.**get_observable**()
Get error information that can be seen

ReturnsObservable error object

Return type*ObservableError*

BaseError.**get_observable_error_probability**()
get the probability of an observable error occuring at a base

Returnserror probability

Return typefloat

BaseError.**get_string**()
Get a string representation of this single base error.

Returnsreport

Return typestring

BaseError.**get_unobservable**()
Unobservable errors inferred, like if its relative to the target and an insertion, then it is not observed in the target, we just know it was inserted between two bases in the target.

ReturnsUnobservable error object

Return type*UnobservableError*

BaseError.**get_unobservable_error_probability**()
get the probability of an unobservable error occuring at a base

Returnserror probability

Return typefloat

BaseError.**is_any_error**()
If theres any reason to attribute this base to an error return True otherwise false

Returnsthere_is_error

Return typebool

BaseError.**set_observable**(*tseq*, *qseq*)
Set the observable sequence data

Parameters

- **tseq** (*string*) – target sequence (from the homopolymer)
- **qseq** (*string*) – query sequence (from the homopolymer)

BaseError.**set_unobserved_after**(*tlen*, *qlen*, *nt*, *p*)
Set the unobservable sequence data after this base

Parameters

- **tlen** (*int*) – target homopolymer length

- **qlen** (*int*) – query homopolymer length
- **nt** (*char*) – nucleotide
- **p** (*float*) – p is the probability of attributing this base to the unobserved error

BaseError.**set_unobserved_before**(*tlen*, *qlen*, *nt*, *p*)
Set the unobservable sequence data before this base

Parameters

- **tlen** (*int*) – target homopolymer length
- **qlen** (*int*) – query homopolymer length
- **nt** (*char*) – nucleotide
- **p** (*float*) – p is the probability of attributing this base to the unobserved error

class seqtools.errors.**ErrorProfileFactory**
This class is used to create an error profile. It doesn't require any special input to create a new instance of it. You add to it with the add_alignment() function.

add_alignment(*align*)
Calculate alignment errors from the alignment and add it to the profile.

add_alignment_errors(*ae*)
If you alread have thealignment errors, add them for profile construction.

close()
Set some objects to None to hopefully free up some memory.

combine_context_errors()
Each alignment contributes some information to the error report. These reports for each alignment need to be gone through and combined into one report.

ReturnsDictionary containing the error counts on context base

Return typedict()

get_alignment_errors()
Return an object that describes the errors

ReturnsAlignment Errors

Return type*GeneralErrorStats*

get_min_context_count(*context_type*)
Calculate out which context has the minum coverage thusfar.

Parameters**context_type** (*string*) – 'target' or 'query'

ReturnsMinimum Coverage

Return typeint

get_query_context_error_report()
Get a report on context-specific errors relative to what is expected on the query strand.

ReturnsObject with a 'header' and a 'data' where data describes context: before,after ,reference, query. A total is kept for each reference base, and individual errors are finally checked

Return typedict()

get_query_context_errors()
Return the query context errors

ReturnsDictionary containing the error counts on context base

Return typedict()

get_string()
Make a string reprentation of the error stats.

Returnserror profile

Return typestring

get_target_context_error_report()
Get a report on context-specific errors relative to what is expected on the target strand.

ReturnsObject with a 'header' and a 'data' where data describes context: before,after ,reference, query. A total is kept for each reference base, and individual errors are finally checked

Return typedict()

get_target_context_errors()
Return the target context errors

ReturnsDictionary containing the error counts on context base

Return typedict()

write_context_error_report(*file*, *context_type*)
Write a context error report relative to the target or query into the specified filename

Parameters

- **file** (*string*) – The name of a file to write the report to
- **context_type** (*string*) – They type of profile, target or query based

class seqtools.errors.**GeneralErrorStats**
Keep track of general errors across the length of an alignment

add_alignment_errors(*ae*)
Add alignment errors to the group

Parameters

- **ae** – one set of alignment errors
- **type** –

get_report()
Another report, but not context based

get_stats()
Return a string describing the stats

get_string()
make a string representation of the general error report

seqtools.graph module

This module has classes to provide graph structures and graph-based operations.

class seqtools.graph.**Edge**(*node1*, *node2*, *directionless=False*, *weight=None*)
Class defines an edge.

directed graph by default

Parameters

- **node1** (Node) – required - node 2
- **node2** (Node) – required - node 1
- **directionless** (*bool*) – by defalt we are directed graph
- **weight** – value to weight the edge

get_id()
get the internal id of the edge. probably uuid4

get_node1()
get what is called node1

Returnsnode1

Return type*Node*

get_node2()
get what is called node2

Returnsnode2

Return type*Node*

get_node_ids()
get the uuid4 ids of the nodes in the edge

Returnslist of [id1,id2]

Return typelist

get_weight()
get the weight if its been set

is_directionless()
get the direction status of the edge

set_weight(*weight*)
can set weight to some number

class seqtools.graph.**Graph**(*directionless=False*)
Graph basic structure.

Use directed graph by default

Parameters**directionless** (*bool*) – use an undirected graph if set to true

add_edge(*edge*, *verbose=True*)
add an edge to the graph

Parameters

- **edge** (Edge) –
- **verbose** (*bool*) –

–optional default (True)

add_node(*node*)
add a node to the graph

Parameters**node** (Node) –

connected_nodes(*node*, *exclude_ids=None*)
get all the connected nodes

Parameters

- **node** (Node) –
- **exclude_ids** (*list or None*) –

Returns list of connected nodes

Return type Node[]

find_cycle()

return a single cycle, greedy first one found in terms of nodes return as an array of nodes or None. done by depth first search through nodes

Returns nodes in the cycle (list) or None

Return type Nodes[] or None

get_children(*node*)

Find all the children of a node. must be a undirectional graph with no cycles

Parameters **node** (Node) –

Returns list of nodes

Return type Node[]

get_directed_paths_from_node(*node*, *prev=[]*)

get all the paths in terms of lists of nodes from a node. needs to be a directed graph with no cycles.

Parameters

- **node** (Node) –
- **prev** (*list*) – do not used, used by the class when calling it recurrsively

get_edges()

a list of edges

Returns edges

Return type Edge[] list

get_node_edges(*node*, *type='both'*)

given a node return the edges attached, by default get both incoming and outgoing

Parameters

- **node** (Node) –
- **type** (*string - default 'both'*) –

Returns edge list

Return type Edge[] edge list

get_nodes()

a list of the nodes

Returns Nodes

Return type Node[] list of nodes

get_report()

describe the graph

Returns report

Return typestring

get_roots()
get the roots of a graph. must be a directed graph

Returnsroot list of nodes

Return type Node[]

get_status_string()
get a string describing some stats about a graph

merge_cycles()
remove cycles by mergine cyclic nodes into single nodes their payloads are added to a list

partition_graph(*verbose=False*)
break a graph into multiple graphs if they are not connected

Returnslist of graphs

Return type Graph[]

remove_edge(*edge*)
remove an edge from the graph

Parameters**edge** (Edge) –

remove_node(*node*)
remove a node from the graph

Parameters**node** (Node) –

class seqtools.graph.**Node**(*payload=None*)
Class to describe a node

Parameters**payload** (*anything you want*) – Empty payload by default

get_id()
return the uuid4 id

get_payload()
return whats curently held in payload

set_payload(*payload*)
set the payload to anything you want

seqtools.range module

These classes are to help deal with genomic coordinates and things associated with those coordinates.

class seqtools.range.**Bed**(*chrom*, *start*, *finish*, *dir=None*)
Bases: *seqtools.range.GenomicRange*

Bed format is a chromosome, start (0-index), end (1-index). It is a child of GenomicRange but modifies the class to use the 0-based start 1-based end style of a bed file

Parameters

- **chrom** (*string*) –
- **start** (*int*) – 0-indexed
- **finish** (*int*) – 1-indexed

- **dir** (*char*) –

  –or - (optional)

copy()
Override copy to make another copy

Returnsa new copy of this object

Return type*Bed*

class seqtools.range.**BedArrayStream**(*bedarray*)
Make a stream from a bedarray

Read as an interator or with read_entry()

Parameters**bedarray** (*Bed[]*) –

next()
call read_entry() from inside this iterator

read_entry()
get the next value from the array, and set internal iterator so next call will be next entry

ReturnsThe next GenomicRange entry

Return type*GenomicRange*

class seqtools.range.**BedStream**(*fh*)
Make a stream from a handle, keep it as an iterator

Parameters**fh** (*handle*) – readable file handle or stream

next()

read_entry()
read the next bed entry from the stream

class seqtools.range.**GenomicRange**(*chr=None*, *start=None*, *end=None*, *dir=None*, *range_string=None*)
A basic class for keeping genomic range data. It is 1-indexed for both start and end.

Parameters

- **chr** (*char*) – chromosome name
- **start** (*int*) – 1-indexed starting base
- **end** (*int*) – 1-indexed ending base
- **dir** (*char*) – direction
- **range_string** (*string*) – set from string like chr5:801-900

adjacent(*rng2*, *use_direction=False*)
Returns true if the two beds are directly next to eachother, 'touching'

Parameters

- **rng2** –
- **use_direction** – false by default
- **type** – GenomicRange
- **type** – use_direction

cmp(*range2*, *overlap_size=0*)
the comparitor for ranges

- return 1 if greater than range2
- return -1 if less than range2
- return 0 if overlapped

Parameters

- **range2** (GenomicRange) –
- **overlap_size** (*int*) – allow some padding for an 'equal' comparison (default 0)

copy()
Create a new copy of selfe. does not do a deep copy for payload

Returnscopied range

Return type*GenomicRange*

distance(*rng*)
The distance between two ranges.

Parameters**rng** (GenomicRange) – another range

Returnsbases separting, 0 if overlapped or adjacent, -1 if on different chromsomes

Return typeint

dump_serialized()
dump the pickle for this object

Returnspickled object

Return typepickle_object

equals(*rng*)

get_bed_array()
Return a basic three meber bed array representation of this range

Returnslist of [chr,start (0-indexed), end (1-indexed]

Return typelist

get_bed_coordinates()

Same as get bed array. These are the 0-indexed start, 1-indexted stop coordinates

Returnsbed array [chr,start-1, end]

get_direction()
return the direction

Returnsthe direction or strand +/- (or None if not set)

Return typechar

get_genomic_coordinates()
These are the 1-indexed coordiantes in list form

Returnslist of coords [chr, start (1-indexed), end(1-indexed)

Return typelist

get_payload()
Returns the payload, whatever it may be

get_range()
For compatability with some range-based tools that need to call this function

Returnsthis object

Return type*GenomicRange*

get_range_string()
get the range string represetation. similar to the default input for UCSC genome browser

Returnsrepresentation by string like chr2:801-900

Return typestring

length()
get the length of the range

Returnslength

Return typeint

load_serialized(*instr*)
load a pickeled range back into the object

Parameters**instr** (*pickled_object*) –

merge(*range2*, *use_direction=False*)
merge this bed with another bed to make a longer bed. Returns None if on different chromosomes or direction is set true and they are in differet strands.

Parameters

- **range2** (GenomicRange) –
- **use_direction** (*bool*) – consider direction for overlapping? (default False)

Returnsbigger range with both

Return type*GenomicRange*

overlap_size(*in_genomic_range*)
The size of the overlap

Parameters**in_genomic_range** (GenomicRange) – the range to intersect

Returnscount of overlapping bases

Return typeint

overlaps(*in_genomic_range*, *use_direction=False*, *padding=0*)
do the ranges overlap?

Parameters

- **in_genomic_range** (GenomicRange) – range to compare to
- **use_direction** (*bool*) – default (False)
- **padding** (*int*) – add to the ends this many (default 0)

ReturnsTrue if they overlap

Return typebool

overlaps_with_padding(*in_genomic_range*, *padding*)
Does the range overlap with a padded range. Looks like this is fairly redundnat with the overlaps function

Parameters

- **in_genomic_range** (GenomicRange) – range to compare to
- **padding** (*int*) – amount to add onto ends to search

Returnstrue if they overlap, false if they do not

Return typebool

print_range()
print the range string to stdout

set_direction(*dir*)
set he direction

Parameters**dir** (*char*) – direction + or -

set_payload(*inpay*)
Set the payload. Stored in a list to try to keep it as a reference

Parameters**inpay** – payload input - any type that can be pushed into a list

subtract(*range2*, *use_direction=False*)
Take another range, and list of ranges after removing range2, no garuntees on payload

Parameters

- **range2** (GenomicRange) –
- **use_direction** (*bool*) –

ReturnsList of Genomic Ranges

Return type GenomicRange[]

union(*range2*)
Intersection may be a better description. Return the chunk they overlap as a range. direction is destroyed

Parameters**range2** (GenomicRange) –

ReturnsRange with the intersecting segement, or None if not overlapping

Return type*GenomicRange*

class seqtools.range.**Loci**
multiple locus combined together when new members are added based on parameters

add_locus(*inlocus*)
Adds a locus to our loci, but does not go through an update our locus sets yet

merge_down_loci()
Called internally to make loci overlapping into one set

set_minimum_distance(*over*)
In preparation for combining loci specify how many basepairs they may be separated by but still get merged

set_use_direction(*inbool*)
Do we want to only combine loci when they have the same direction, if so, set to True

update_loci()
Goes through and combines loci until we have one set meeting our overlap definition

class seqtools.range.**Locus**
A Locus is a colloction of GenomicRanges that fall within some distance of one another

add_member(*grange*)
Add a genomic range to the locus

Parameters **grange** (GenomicRange) –

set_use_direction(*inbool*)
Set to true if you want all locus members to share the same direction

Parameters **inbool** (*bool*) –

seqtools.range.**merge_ranges**(*inranges*, *already_sorted=False*)

from a list of genomic range or bed entries, whether or not they are already sorted, make a flattend range list of ranges where if they overlapped, they are now joined (not yet) The new range payloads will be the previous ranges

Parameters

- **inranges** (*GenomicRange[]*) –
- **already_sorted** (*bool*) – has this already been sorted (defaults to False)

Returns sorted ranges

Return type GenomicRange[]

seqtools.range.**pad_ranges**(*inranges*, *padding*, *chr_ranges=None*)
Add the specfied amount onto the edges the transcripts

Parameters

- **inranges** (*GenomicRange[]*) – List of genomic ranges in Bed o GenomicRange format.
- **padding** (*int*) – how much to add on
- **chr_ranges** – looks like the list of ranges within which to pad

seqtools.range.**ranges_to_coverage**(*rngs*, *threads=1*)
take a list of ranges as an input output a list of ranges and the coverage at each range :param rngs: bed ranges on a single chromosome. not certain about that single chromosome requirement :type rngs: GenomicRange[] or Bed[] :param threads: Not currently being used :type threads: int

Returns out is the non-overlapping bed ranges with the edition of depth

Return type GenomicRange[]

seqtools.range.**sort_genomic_ranges**(*rngs*)
sort multiple ranges

seqtools.range.**sort_ranges**(*inranges*)
from an array of ranges, make a sorted array of ranges

Parameters **inranges** (*GenomicRange[]*) – List of GenomicRange data

Returns a new sorted GenomicRange list

Return type GenomicRange[]

seqtools.range.**string_to_genomic_range**(*rstring*)
Convert a string to a genomic range

Parameters **rstring** – string representing a genomic range chr1:801-900

Returnsobject representing the string

Return type*GenomicRange*

seqtools.range.**subtract_range_array**(*bed1*, *beds2*, *is_sorted=False*)
subtract several ranges from a range, returns array1 - (all of array2)

Parameters

- **bed1** (*Bed or GenomicRange*) –
- **beds2** (*Bed[] or GenomicRange[]*) – subtract all these beds from bed1
- **is_sorted** – has it been sorted already? Default (False)
- **is_sorted** – bool

seqtools.range.**subtract_ranges**(*r1s*, *r2s*, *already_sorted=False*)
Subtract multiple ranges from a list of ranges

Parameters

- **r1s** (*GenomicRange[]*) – range list 1
- **r2s** (*GenomicRange[]*) – range list 2
- **already_sorted** – default (False)

Returnsnew range r1s minus r2s

Return type GenomicRange[]

seqtools.range.**union_range_array**(*bed1*, *beds2*, *payload=None*, *is_sorted=False*)
Does not do a merge if the payload has been set

Parameters

- **bed1** (GenomicRange) –
- **bed2** (GenomicRange) –
- **payload** (*int*) – payload=1 return the payload of bed1 on each of the union set, payload=2 return the payload of bed2 on each of the union set, payload=3 return the payload of bed1 and bed2 on each of the union set
- **is_sorted** (*bool*) –

seqtools.sequence module

class seqtools.sequence.**Seq**(*seq=None*, *name=None*)
Basic Sequence structure

Parameters

- **seq** (*string*) – nucleotide sequence
- **name** – name is optional

copy()
a new sequence object

Returnscopy of the sequence

Return type*Seq*

fasta()
Get the fasta formated string Pre: seq and name are set Post: string representation of the fasta entry

Returnsfasta string

Return typestring

gc_content()
Calculate the GC content of the sequence

ReturnsGC content

Return typefloat

n_count()
Count the numbers of 'N's in a sequence. case insensitive.

ReturnsN count

Return typeint

rc()
reverse complement

Returnsreverse complemented sequence of same name

Return type*Seq*

seqtools.sequence.**decode_name**(*safename*)
Make an encoded name into its decoded.

Parameters**safename** – thing to be decoded

Returnsdecoded name

Return typestring

seqtools.sequence.**encode_name**(*conversion_string*)
Make a name into an encoding that can store any character

Parameters**conversion_string** (*string*) – thing to be encoded

Returnsencoded_name

Return typestring

seqtools.sequence.**rc**(*seq*)
Fast reverse complement function using a translation table and slice

Parameters**seq** (*string*) – string to reverse complement

Returnsreverse complemented sequence

Return typestring

seqtools.statistics module

This module contains many list-based functions to calculate descriptive statistics.

seqtools.statistics.**N50**(*arr*)
N50 often used in assessing denovo assembly.

Parameters**arr** (*number[] a number array*) – list of numbers

ReturnsN50

Return typefloat

seqtools.statistics.**average**(*arr*)
average of the values, must have more than 0 entries.

Parameters **arr** (*number[] a number array*) – list of numbers

Returns average

Return type float

seqtools.statistics.**median**(*arr*)
median of the values, must have more than 0 entries.

Parameters **arr** (*number[] a number array*) – list of numbers

Returns median

Return type float

seqtools.statistics.**standard_deviation**(*arr*)
standard deviation of the values, must have 2 or more entries.

Parameters **arr** (*number[] a number array*) – list of numbers

Returns standard deviation

Return type float

seqtools.statistics.**variance**(*arr*)
variance of the values, must have 2 or more entries.

Parameters **arr** (*number[] a number array*) – list of numbers

Returns variance

Return type float

seqtools.stream module

Classes to help stream biological data

class seqtools.stream.**GZippedOutputFile**(*filename*)
use gzip utility to compress output

Parameters **filename** (*string*) – filename to write to

close()

write(*value*)

class seqtools.stream.**LocusStream**(*stream*)

Works for any stream with ordered range bound objects that have the functions. LocusStream is a stream itself, and is iterable

1.read_entry()

2.get_range()

Data is not stored as an actual Locus object, but rather in list in the payload of the range covered by the locus

Parameters **stream** (*Stream*) – ordered stream with range

next()

read_entry()

As long as entires overlap keep putting them together in a list that isthe payload for a range that describes the bounds of the list

Returnsrange with payload list of elements

Return type*GenomicRange*

class seqtools.stream.**MultiLocusStream**(*streams*)
Take an array streams Each element should be sorted by position Streams need to have this method:

1.read_entry()

2.get_range()

Parameters**streams** (*list*) – list of streams

next()

read_entry()
get the next aggrogate of streams

Returnsrange containing a list of entries from each stream that are from the overlapping part

Return type*GenomicRange*

seqtools.structure module

this collection of classes helps us operate on mappings of transcripts

class seqtools.structure.**Exon**(*rng=None*)
class to describe an exon

Parameters**rng** (GenomicRange) –

dump_serialized()

get_length()

get_range()

load_serialized(*instr*)

set_is_leftmost(*boo=True*)

set_is_rightmost(*boo=True*)

set_left_junc(*junc*)

set_right_junc(*junc*)

class seqtools.structure.**Junction**(*rng_left=None*, *rng_right=None*)
class to describe a junction

Parameters

- **rng_left** (GenomicRange) – left side of junction
- **rng_right** (GenomicRange) – right side of junction

cmp(*junc*, *tolerance=0*)
output comparison and allow for tolerance if desired

- -1 if junc comes before self
- 1 if junc comes after self
- 0 if overlaps
- 2 if else

Parameters

- **junc** (Junction) –
- **tolerance** (*int*) – optional search space (default=0, no tolerance)

Returns value of comparison

Return type int

dump_serialized()
Get string representation of the junction

Returns serialized object

Return type string

equals(*junc*)
test equality with another junction

get_left_exon()
get the exon to the left of the junction

Returns left exon

Return type *Exon*

get_range_string()
Another string representation of the junction. these may be redundant.

get_right_exon()
get the exon to the right of the junction

Returns right exon

Return type Exon or GenomicRange

get_string()
A string representation of the junction

Returns string represnetation

Return type string

load_serialized(*instr*)
load the string

Parameters **instr** (*string*) –

overlaps(*junc*, *tolerance=0*)
see if junction overlaps with tolerance

set_exon_left(*ex*)
assign the left exon

set_exon_right(*ex*)
assign the right exon

set_left(*rng*)
Assign the leftmost range

set_right(*rng*)
Assign the right most range

class seqtools.structure.**Transcript**
Class to describe the mapping of a basic transcript

class ExonOverlap(*self1*, *tx_obj1*, *tx_obj2*, *multi_minover=10*, *multi_endfrac=0*, *multi_midfrac=0.8*, *single_minover=50*, *single_frac=0.5*, *multi_consec=True*)
class to describe exon overlap

Parameters

- **tx** –
- **multi_minover** (*int*) – multi-exons need to overlap by at lest this much to be considered overlapped (default 10)
- **multi_endfrac** (*float*) – multi-exons need an end fraction coverage of at least this by default (default 0)
- **multi_midfrac** (*float*) – multi-exons need (default 0.8) mutual coverage for internal exons
- **single_frac** (*float*) – at least this fraction of single exons must overlap (default 0.5)

Parma single_minoversingle-exons need at least this much shared overlap (default 50)

Parma multi_consecexons need to have multiexon consecutive mapping to consider it a match (default True)

ReturnsExonOverlap report

Return type*Transcript.ExonOverlap*

analyze_overs(*self1*)
A helper function that prepares overlap and consecutive matches data

calculate_overlap(*self1*)
Create the array that describes how junctions overlap

consecutive_exon_count(*self1*)
Best number of consecutive exons that overlap
Returnsmatched consecutive exon count
Return typeint

is_compatible(*self1*)
Return True if the transcripts can be combined together
Returnscan be combined together
Return typebool

is_full_overlap(*self1*)
true if they are a full overlap
Returnsis full overlap
Return typebool

is_subset(*self1*)
Return value if tx_obj2 is a complete subset of tx_obj1 or tx_obj1 is a complete subset of tx_obj2

Values are:
•Return 1: Full overlap (mutual subests)
•Return 2: two is a subset of one

•Return 3: one is a subset of two
•Return False if neither is a subset of the other

Returnssubset value
Return typeint

match_exon_count(*self1*)
Total number of exons that overlap
Returnsmatched exon count
Return typeint

class Transcript.**JunctionOverlap**(*self1*, *tx_obj1*, *tx_obj2*, *tolerance=0*)
Class for describing the overlap of junctions between transcripts

This should probably be not a child.

Parameters

•**tx_obj1** (Transcript) – transcript1

•**tx_obj2** (Transcript) – transcript2

•**tolerance** (*int*) – how far before its no longer a matched junction

analyze_overs(*self1*)
A helper function to prepare values describing overlaps

calculate_overlap(*self1*)
Create the array that describes how junctions overlap

is_compatible(*self1*)
Return True if the transcripts can be combined together
ReturnsTrue if we can combine
Return typebool

is_full_overlap(*self1*)
True if its a full overlap
ReturnsTrue if its a full overlap
Return typebool

is_subset(*self1*)
Return value if tx_obj2 is a complete subset of tx_obj1 or tx_obj1 is a complete subset of tx_obj2

values:
•Return 1: Full overlap (mutual subests)
•Return 2: two is a subset of one
•Return 3: one is a subset of two
•Return False if neither is a subset of the other

match_junction_count(*self1*)

Transcript.**copy**()
A copy of the transcript

Returnstranscript copy

Return type*Transcript*

Transcript.**dump_serialized**()
Generate a string representation of the transcript

Returnsserialized_object

Return typestring

Transcript.**exon_overlap**(*tx*, *multi_minover=10*, *multi_endfrac=0*, *multi_midfrac=0.8*, *single_minover=50*, *single_frac=0.5*, *multi_consec=True*)

Get a report on how mucht the exons overlap

Parameters

- **tx** –
- **multi_minover** (*int*) – multi-exons need to overlap by at lest this much to be considered overlapped (default 10)
- **multi_endfrac** (*float*) – multi-exons need an end fraction coverage of at least this by default (default 0)
- **multi_midfrac** (*float*) – multi-exons need (default 0.8) mutual coverage for internal exons
- **single_frac** (*float*) – at least this fraction of single exons must overlap (default 0.5)

Parma single_minoversingle-exons need at least this much shared overlap (default 50)

Parma multi_consecexons need to have multiexon consecutive mapping to consider it a match (default True)

ReturnsExonOverlap report

Return type*Transcript.ExonOverlap*

Transcript.**exons**

Maybe the most core property of the transcript are the exon defintions. This is an array of exons.

Transcript.**get_chrom**()

the reference chromosome. greedy return the first chromosome in exon array

Returnschromosome

Return typestring

Transcript.**get_exon_count**()

Count the exons in the transcript

Returnsexon count

Return typeint

Transcript.**get_fake_gpd_line**()

Convert a mapping to a fake GPD line. not sure why its called fake

Returnsgpd line

Return typestring

Transcript.**get_fake_psl_line**(*ref*)

Convert a mapping to a fake PSL line

Parameters**ref** (*dict()*) – reference genome dictionary

Returnspsl line

Return typestring

Transcript.**get_gene_name**()

retrieve the gene name

Returnsgene name

Return typestring

Transcript.**get_gpd_line**(*transcript_name=None*, *gene_name=None*, *strand=None*)
Get the genpred format string representation of the mapping

Parameters

- **transcript_name** (*string*) –
- **gene_name** (*string*) –
- **strand** (*string*) –

ReturnsGPD line

Return typestring

Transcript.**get_id**()
Return a unique id created for this transcript when it was made

Returnsuuid4 id as a string

Return typestring

Transcript.**get_junction_string**()
Make a string representation of all the junctions

Returnsjunctions as a string

Return typestring

Transcript.**get_junctions_string**()
Get a string representation of the junctions. This is almost identical to a previous function.

That function is get_junction_string. A refactor should clear this redundancy.

Returnsstring representation of junction

Return typestring

Transcript.**get_length**()
Return the length of the transcript in bp. Its the sum of the exons

Returnslength

Return typeint

Transcript.**get_payload**()
Get the payload currently being stored

Returnspayload

Return typeanything that can be stored in a list

Transcript.**get_range**()
Get the range from the leftmost exon to the rightmost

Returnstotal range

Return type*GenomicRange*

Transcript.**get_sequence**(*ref_dict=None*)
A strcutre is defined so get, if the sequence is not already there, get the sequence from the reference

Parameters**ref_dict** (*dict()*) – reference dictionary (only necessary if sequence has not been set already)

Transcript.**get_strand**()
Get the strand

Returnsdirection + or -

Return typechar

Transcript.**get_transcript_name**()
retrieve the transcript name

Returnstranscript name

Return typestring

Transcript.**junction_overlap**(*tx*, *tolerance=0*)
Calculate the junction overlap between two transcripts

Parameters

- **tx** (Transcript) – Other transcript
- **tolerance** (*int*) – how close to consider two junctions as overlapped (default=0)

ReturnsJunction Overlap Report

Return type*Transcript.JunctionOverlap*

Transcript.**junctions**
Can be inferred from the exons, this is an array of junctions

Transcript.**load_serialized**(*instr*)
Load a serialized object string into the object

Parameters**instr** (*string*) – The serialized string

Transcript.**overlap_size**(*tx2*)
Return the number of overlapping base pairs between two transcripts

Parameters**tx2** (Transcript) – Another transcript

Returnsoverlap size in base pairs

Return typeint

Transcript.**set_exons_and_junctions_from_ranges**(*rngs*)

set all exons and subsequestly juntions from these exon ranges;does not set direction of transcript; ranges need to be ordered in target order left to right

This is a core feature for setting up a transcript.

Parameters**rngs** (*GenomicRange[]*) – A list of ranges ordered left to right

Transcript.**set_gene_name**(*name*)
assign a gene name

Parameters**name** (*string*) – name

Transcript.**set_payload**(*val*)
Set a payload for this object

Parameters**val** (*Anything that can be put in a list*) – payload to be stored

Transcript.**set_range**()
Use the exons that are already present to set the range. In a refactor this seems like it should go away or become private.

Transcript.**set_sequence**(*ref_dict*)
use the reference dictionary to set the transcript's sequence

Parameters**ref_dict** (*dict()*) – reference dictionary

Transcript.**set_strand**(*dir*)
Set the strand (direction)

Parameters **dir** (*char*) – direction + or -

Transcript.**set_transcript_name**(*name*)
assign a transcript name

Parameters **name** (*string*) – name

Transcript.**smooth_gaps**(*min_intron*)
any gaps smaller than min_intron are joined, andreturns a new transcript with gaps smoothed

Parameters **min_intron** (*int*) – the smallest an intron can be, smaller gaps will be sealed

Returns a mapping with small gaps closed

Return type *Transcript*

Transcript.**subset**(*start*, *finish*)

Make a trimmed transcript Pre: Start base index 0 Post: Finish base index 1

Parameters

- **start** (*int*) – 0-index start
- **end** (*int*) –

Returns subset transcript

Return type *Transcript*

Transcript.**union**(*tx2*)
Find the union, or perhaps intersection is a better word for it, for two transcripts. This makes a new transcript.

Parameters **tx2** (Transcript) – transcript 2

Returns overlapping portion of the transcripts

Return type *Transcript*

Transcript.**validate**()
be certain the scructure is a transcriptome

Returns true if exon order is compatible with a transcriptome

Return type list

class seqtools.structure.**TranscriptGroup**
A transcript group is like the fuzzy gpd class we had before

class **JunctionGroup**(*self1*, *outer*)
Describe a junction as a group of junctions with options for junction tolerance

add_junction(*self1*, *tx_index*, *junc_index*, *tolerance=0*)
add a junction

get_junction(*self1*)
return the consensus junction

TranscriptGroup.**add_transcript**(*tx*, *juntol=0*, *verbose=True*)

TranscriptGroup.**get_transcript**(*exon_bounds='max'*)
Return a representative transcript object

class seqtools.structure.**TranscriptLoci**
combine together compatible multiple transcript groups to form a simpler set of transcripts

add_transcript(*tx*)
Add a transcript to the locus

Parameters **tx** (Transcript) – transcript to add

get_depth_per_transcript(*mindepth=1*)
using all the transcripts find the depth

get_range()
Return the range the transcript loci covers

Returns range

Return type *GenomicRange*

get_transcripts()
a list of the transcripts in the locus

partition_loci(*verbose=False*)
break the locus up into unconnected loci

Returns list of loci

Return type TranscriptLoci[]

remove_transcript(*tx_id*)
Remove a transcript from the locus by its id

Parameters **tx_id** (*string*) –

set_merge_rules(*mr*)
Define rules for how to merge transcripts

Parameters **mr** (TranscriptLociMergeRules) –

class seqtools.structure.**TranscriptLociMergeRules**(*merge_type*)
Establish rules up on which to merge loci

get_exon_rules()

get_juntol()

get_merge_type()

get_use_junctions()

get_use_multi_exons()

get_use_single_exons()

set_juntol(*juntol*)

set_use_junctions(*boo=True*)

class seqtools.structure.**Transcriptome**(*gpd_file=None*, *ref_fasta=None*)
a class to store a transcriptome

Parameters

- **gpd_file** (*string*) – filename
- **ref_fasta** (*dict()*) –

add_transcript(*transcript*)

dump_serialized()

get_transcripts()

load_serialized(*instr*)

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

S

A

B

C

D

E

F

G

N

O

P

Q

R

S

T

U

V

W