Counts File Library Documentation

Release 1.1.0

Dominik Schrempf

Jul 26, 2016

CONTENTS

This library provides functions and classes to handle file conversion between standard formats (e.g., fasta or VCF files) to counts files that are used by IQ-TREE with PoMo, and implementation of a polymorphism aware phylogenetic model.

Created by:

- Dominik Schrempf

For a reference, please see and cite: Schrempf, D., Minh, B. Q., De Maio, N., von Haeseler, A., & Kosiol, C. (2016). Reversible Polmorphism-Aware Phylotenetic Models and their Application to Tree Inference. Journal of Theoretical Biology, in press.

Feel free to post any suggestions, doubts and bugs.

CHAPTER

ONE

CFLIB

cflib contains several modules that ease the handling and preparation of data files in variant call format (vcf), fasta format and counts format (cf).

The *libPoMo* package is split into the following modules:

- *main*: Contains functions that are used by PoMo.
- *seqbase*: Provides basic functions and classes needed to work with sequence data.
- *fasta*: Provides functions to read, write and access fasta files.
- *vcf*: Provides functions to read, write and access vcf files.
- *cf*: Provides functions to read, write and access files that are in counts format.

CHAPTER

TWO

CONTENTS

2.1 libPoMo.main

This library contains functions that are used by PoMo.

libPoMo.main.**a**(*n*)
Calculate the Watterson's Theta coefficient.

libPoMo.main.**binom**(*s*, *p*, *n*)
Binomial Distribution

Calculate the binomial sampling probability (not very efficient, but not much effieciency is needed with small samples).

libPoMo.main.**dsRatio**(*dsR*)
Downsampling ratio **type** for argparse.

libPoMo.main.**get_data_from_cf_line**(*cfStr*)
Read in the data of a single counts format line.

The return type is a list with the number of samples and a two dimensional array of the form data[species][nucleotide], where species is the index of the species and nucleotide is the index of the nucleotide (0,1,2 or 3 for a,c,g and t, respectively).

Parameters **CFStream**(*cfStr*) – The CFStream pointing to the line to be read in.

Return type ([int] n_samples, [[int]] data)

libPoMo.main.**get_species_from_cf_headerline**(*line*)
Get the number of species and the names fom a counts format header line.

Parameters **line**(*str*) – The header line.

Return type (int n_species, [str] sp_names)

libPoMo.main.**is_number**(*s*)
Determine if value is an integer.

libPoMo.main.**mutModel**(*mm*)
Mutation model **type** for argparse.

libPoMo.main.**probability_matrix**(*n*)
Create probability matrices for the HyPhy batch file.

libPoMo.main.**read_data_write_HyPhy_input**(*fn*, *N*, *thresh*, *path_bf*, *muts*, *mutgamma*, *sels*, *selgamma*, *PoModatafile*, *PoModatafile_cons*, *theta=None*, *vb=None*)
Read the count data and write the HyPhy input file.

The provided filename has to point to a data file in counts format (cf. *cf*). The data will be downsampled if necessary and the HyPhy batch and input files will be written. The number of species, the species names, the number of species samples and the theta value (usr_def) will be returned in a tuple.

Parameters

- **fn** (*str*) – Counts format file name.
- **N** (*int*) – Virtual population size.
- **thresh** (*float*) – Trheshold of data discard for downsampling.
- **path_bf** (*str*) – Path to the HyPhy batch files
- **muts** (*str*) – Mutation model (*mutModel()*).
- **mutgamma** (*str*) – Gamma of the mutation model (*setGM()*).
- **sels** (*str*) – Selection model (*selModel()*).
- **selgamma** (*str*) – Gamma of selection model (*setGS()*).
- **PoModatafile** (*str*) – Path to HyPhy input file.
- **PoModatafile_cons** (*str*) – Path to HyPhy input file.
- **vb** (*Boolean*) – Verbosity.

Return type (int n_species, [str] sp_names, [str] sp_samples, Boolean all_one, float usr_def)

libPoMo.main.**selModel**(*sm*)
Selection model **type** for argparse.

libPoMo.main.**setGM**(*gm*)
Set variable mutation rate, if *gm* is given.

libPoMo.main.**setGS**(*gs*)
Set fixation bias, if *gs* is given.

libPoMo.main.**timeStr**()
Time in human readable format.

2.2 libPoMo.seqbase

This module provides basic functions and classes needed to work with sequence data.

2.2.1 Objects

Classes:

- *Seq*, stores a single sequence
- *Region*, region in a genome

Exception Classes:

- *SequenceDataError*
- *NotAValidRefBase*

Functions:

- *stripFName()*, strip filename off its ending

exception libPoMo.seqbase.**NotAValidRefBase**
Reference base is not valid.

class libPoMo.seqbase.**Region**(*chrom*, *start*, *end*, *name=None*, *orientation='+'*)
Region in a genome.

The start and end points need to be given 1-based and are converted to 0-based positions that are used internally to save all positional data.

Parameters

- **chrom** (*str*) – Chromosome name.
- **start** (*int*) – 1-based start position.
- **end** (*int*) – 1-based end position.
- **name** (*str*) – Optional, region name.

Variables

- **chrom** (*str*) – Chromosome name.
- **start** (*int*) – 0-based start position.
- **end** (*int*) – 0-base end position.
- **name** (*str*) – Region name.

print_info()
Print information about the region.

class libPoMo.seqbase.**Seq**
A class that stores sequence data. .. _seqbase-seq:

Variables

- **name** (*str*) – Name of the sequence (e.g. species or individual name).
- **descr** (*str*) – Description of the sequence.
- **data** (*str*) – String with sequence data.
- **dataLen** (*int*) – Number of saved bases.
- **rc** (*Boolean*) – True if *self.data* stores the reverse-complement of the real sequence.

get_base(*pos*)
Returns base at 1-based position *pos*.

get_exon_nr()
Try to find the current and the total exon number of the sequence.

Extract the exon number and the total number of exons, if the name of the sequence is of the form (cf. UCSC Table Browser):

```
>CCDS3.1_hg18_2_19
```

Return type (int nEx, int nExTot)

Raises *SequenceDataError*, if the format of the sequence name is invalid.

get_in_frame()
Try to find the *inFrame* of the gene.

inFrame: the frame number of the first nucleotide in the exon. Frame numbers can be 0, 1, or 2 depending on what position that nucleotide takes in the codon which contains it. This function gets the *inFrame*, if the description of the sequence is of the form (cf. UCSC Table Browser):

```
918 0 0 chr1:58954-59871+
```

Return type int

Raises *SequenceDataError*, if format of description is invalid.

get_out_frame()
Try to find the *outFrame* of the gene.

outFrame: the frame number of the last nucleotide in the exon. Frame numbers can be 0, 1, or 2 depending on what position that nucleotide takes in the codon which contains it. This function gets the *outFrame*, if the description of the sequence is of the form (cf. UCSC Table Browser):

```
918 0 0 chr1:58954-59871+
```

Return type int

Raises *SequenceDataError*, if format of description is invalid.

get_rc()
Return True if the sequence is reversed and complemented.

Return type Boolean

get_region()
Try to find the *Region* that the sequence spans.

The sequence might not physically start at position 1 but at some arbitrary value that is indicated in the sequence description. This function gets this physical *Region*, if the description of the sequence is of the form (cf. UCSC Table Browser):

```
918 0 0 chr1:58954-59871+
```

Raises *SequenceDataError*, if format of description is invalid.

get_region_no_description(*offset=0*)
Get the region of the sequence.

If no regional information is available in the sequence description (cf. *get_region()*), the position of the first base in the reference genome can be given manually. E.g., if the first base of the sequence does not correspond to the first but to the 11th base of the reference sequence, the offset should be 10.

The name of the chromosome will be set to the name of the sequence.

Parameters **offset** (*int*) – Optional, offset of the sequence.

is_synonymous(*pos*)
Return True if the base at *pos* is 4-fold degenerate.

This function checks if the base at *pos* is a synonymous one. The description of the sequence has to be of the form (cf. UCSC Table Browser):

```
918 0 0 chr1:58954-59871+
```

Variables **pos** (*int*) – Position of the base in the sequence (0 to self.dataLen).

Rtype Boolean True if base is 4-fold degenerate.

Raises *SequenceDataError*, if format of description is invalid.

print_data (*fo=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Print the sequence data.

Variables **fo** (*fileObject*) – Print to file object fo. Defaults to stdout.

print_fa_entry (*maxB=None*, *fo=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Print a fasta file entry with header and sequence data.

Variables **maxB** (*int*) – Print a maximum of maxB bases. Default: print all bases.

print_fa_header (*fo=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Print the sequence header line in fasta format.

Variables **fo** (*fileObject*) – Print to file object fo. Defaults to stdout.

print_info (*maxB=50*)
Print sequence information.

Print sequence name, description, the length of the sequence and a maximum of *maxB* bases (defaults to 50).

purge ()
Purge data saved in this sequence.

rev_comp (*change_sequence_only=False*)
Reverses and complements the sequence.

This is rather slow for long sequences.

set_rc ()
Set the *self.rc*.

The instance variable *self.rc* is a Boolean value that is true if the saved sequence is reversed and complemented. This function sets this value according to the last character in the sequence description.

Raises *ValueError()* if state could not be detected.

toggle_rc ()
Toggle the state of *self.rc*.

exception libPoMo.seqbase.**SequenceDataError**
General sequence data error exception.

libPoMo.seqbase.**gz_open** (*fn*, *mode='r'*)
Open file with io.open() or gzip.open().

Parameters

- **fn** (*str*) – Name of the file to open.
- **md** (*char*) – Mode '**r**' | 'w'.

libPoMo.seqbase.**stripFName** (*fn*)
Convenience function to strip filename off the ".xyz" ending.

2.3 libPoMo.fasta

This module provides functions to read, write and access fasta files.

2.3.1 Objects

Classes:

- *FaStream*, fasta file sequence stream object
- *MFaStream*, multiple alignment fasta file sequence stream object
- *FaSeq*, fasta file sequence object
- *MFaStrFilterProps*, define multiple fasta file filter preferences

Exception Classes:

- *NotAFastaFileError*

Functions:

- *filter_mfa_str()*, filter a given *MFaStream* according to the filters defined in *MFaStrFilterProps*
- *init_seq()*, initialize fasta sequence stream from file
- *open_seq()*, open fasta file
- *save_as_vcf()*, save a given *FaSeq* in variant call format (VCF)
- *read_seq_from_fo()*, read a single sequence from file object
- *read_align_from_fo()*, read an alignment from file object

---

class libPoMo.fasta.**FaSeq**

Store sequence data retrieved from a fasta file.

Variables

- **name** (*str*) – Name of the *FaSeq* object.
- **seqL** (*[Seq]*) – List of *Seq* objects that store the actual sequence data.
- **nSepcies** (*int*) – Number of saved species / individuals / chromosomes.

get_distance()

Number of segregating bases.

get_seq_base(*seq*, *pos*)

Return base at 1-based position *pos* in sequence with name *seq*.

get_seq_by_id(*i*)

Return sequence number *i* as *Seq* object.

get_seq_names()

Return a list with sequence names.

print_info(*maxB=50*)

Print fasta sequence information.

Print fasta sequence identifier, species names, the length of the sequence and a maximum of *maxB* bases (defaults to 50).

class libPoMo.fasta.**FaStream**(*name*, *firstSeq*, *nextHL*, *faFileObject*)
A class that stores a fasta file sequence stream.

The sequence of one species / individual / chromosome is saved and functions are provided to read in the next sequence in the file, if there is any. This saves memory if files are huge and doesn't increase runtime.

This object is usually initialized with *init_seq()*.

Parameters

- **name** (*str*) – Name of the stream.
- **firstSeq** (Seq) – First sequence (*Seq* object) to be saved.
- **nextHL** (*str*) – Next header line.
- **faFileObject** (*fo*) – File object associated with the stream.

Variables

- **name** (*str*) – Stream name.
- **seq** (Seq) – Saved sequence (*Seq* object)
- **nextHeaderLine** (*str*) – Next header line.
- **fo** (*fo*) – File object that points to the start of the data of the next sequence.

close()
Close the linked file.

print_info(*maxB=50*)
Print sequence information.

Print information about this FaStream object, the fasta sequence stored at the moment the length of the sequence and a maximum of *maxB* bases (defaults to 50).

read_next_seq()
Read next fasta sequence in file.

The return value is the name of the next sequence or None if no next sequence is found.

class libPoMo.fasta.**MFaStrFilterProps**(*nSpecies*)
Define filter preferences for multiple fasta alignments.

Define the properties of the filter to be applied to an *MFaStream*.

By default, all filters are applied (all variables are set to True).

Parameters **nSpecies** (*int*) – Number of species that are aligned.

Variables

- **check_all_aligned** (*Boolean*) – Check if all treated species are available in the alignment (*nSpecies* gives the number of species, given to the object upon initialization).
- **check_divergence** (*Boolean*) – Check if the divergence of the reference genome (the first sequence in the alignment) is lower than *maxDiv* (defaults to 10 percent).
- **check_start_codons** (*Boolean*) – Check if all start codons are conserved.
- **check_stop_codons** (*Boolean*) – Check if all stop codons are conserved.
- **check_frame_shifting_gaps** (*Boolean*) – Check, that there are no frame-shifting gaps.
- **check_for_long_gaps** (*Boolean*) – Check if no gap is longer than *maxGapLength* (defaults to 30) bases.

- **check_nonsense_codon** (*Boolean*) – Check if there is no premature stop codon).
- **check_exon_length** (*Boolean*) – Check that the exon is longer than *minExonLen* (defaults to 21).
- **check_exon_numbers** (*Boolean*) – Check if exon number match for all sequences in the alignment.

class libPoMo.fasta.**MFaStream**(*faFileName*, *maxskip=50*, *name=None*)

Store a multiple alignment fasta file sequence stream.

The sequences of one gene / alignment are saved for all species / individuals / chromosomes. Functions are provided to read in the next gene / alignment in the file that fulfills the given criteria, if there is any. This saves memory if files are huge and doesn't increase runtime.

Initialization of an *MFaStream* opens the given fasta file, checks if it is in fasta format and reads the first alignment. The end of an alignment is reached when a line only contains the newline character. This object can later be used to parse the whole multiple alignment fasta file.

Alignments can be filtered with *filter_mfa_str()*.

Parameters

- **faFileName** (*str*) – File name of the multiple alignment fasta file.
- **maxskip** (*int*) – Only look *maxskip* lines for the start of a sequence (defaults to 50).
- **name** (*str*) – Set the name of the stream to *name*, otherwise set it to the stripped filename.

Variables

- **name** (*str*) – Stream name.
- **seqL** (*[Seq]*) – Saved sequences (*Seq* objects) in a list.
- **nSpecies** (*int*) – Number of saved sequences / species in the alignment.
- **nextHeaderLine** (*str*) – Next header line.
- **fo** (*fo*) – File object that points to the start of the data of the next sequence.

Please close the associated file object with *FaStream.close()* when you don't need it anymore.

close()

Close the linked file object.

orient(*firstOnly=False*)

Orient all sequences of the alignment to be in forward direction.

This is rather slow for long sequences.

Parameters **firstOnly** (*Boolean*) – If true, orient the first sequence only.

print_info(*maxB=50*)

Print sequence information.

Print information about this MFaStream object, the fasta sequence stored at the moment the length of the sequence and a maximum of *maxB* bases (defaults to 50).

print_msa(*fo=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Print multiple sequence alignment at point.

Variables **fo** (*fileObject*) – Print to file object fo. Defaults to stdout.

read_next_align()

Read next alignment in fasta file.

The return value is the name of the newly saved alignment or None if no next alignment is found.

exception libPoMo.fasta.**NotAFastaFileError**
Exception raised if given fasta file is not valid.

libPoMo.fasta.**filter_mfa_str**(*mfaStr*, *fp*, *verb=None*)
Check multiple sequence alignment of an MFaStream.

Multiple sequence alignments usually include alignments that are not apt for analysis. These low quality alignments need to be filtered out of the original multiple sequence alignment fasta file. If *verb* is unset from None, information about any possible rejection is printed to the standard output.

Variables

- **mfaStr** (MFaStream) – *MFaStream* object to check.
- **fp** (MFaStrFilterProps) – *MFaStrFilterProps*; Properties of the filter to be applied.
- **verb** (*Boolean*) – Verbosity.

Return type Boolean, True if all filters have been passed.

libPoMo.fasta.**init_seq**(*faFileName*, *maxskip=50*, *name=None*)
Open a fasta file and initialize an *FaStream*.

This function tries to open the given fasta file, checks if it is in fasta format and reads the first sequence. It returns an *FaStream* object. This object can later be used to parse the whole fasta file.

Please close the associated file object with *FaStream.close()* when you don't need it anymore.

Parameters

- **faFileName** (*str*) – File name of the fasta file.
- **maxskip** (*int*) – Only look *maxskip* lines for the start of a sequence (defaults to 50).
- **name** (*str*) – Set the name of the sequence to *name*, otherwise set it to the stripped filename.

libPoMo.fasta.**open_seq**(*faFileName*, *maxskip=50*, *name=None*)
Open and read a fasta file.

This function tries to open the given fasta file, checks if it is in fasta format and reads the sequence(s). It returns an *FaSeq* object that contains a list of species names, a list of the respective desriptions and a list with the sequences.

Parameters

- **faFileName** (*str*) – Name of the fasta file.
- **maxskip** (*int*) – Only look *maxskip* lines for the start of a sequence (defaults to 50).
- **name** (*str*) – Set the name of the sequence to *name* otherwise set it to the stripped filename.

libPoMo.fasta.**read_align_from_fo**(*line*, *fo*)
Read a single fasta alignment.

Read a single fasta alignment from file object *fo* and save it to new *Seq* sequence objects. Return the header line of the next fasta alignment and the newly created sequences in a list. If no new alignment is found, the next header line will be set to None.

Parameters

- **line** (*str*) – Header line of the sequence.
- **fo** (*fo*) – File object of the fasta file.

Return type (str, [Seq])

libPoMo.fasta.**read_seq_from_fo**(*line*, *fo*, *getAlignEndFlag=False*)

Read a single fasta sequence.

Read a single fasta sequence from file object *fo* and save it to a new *Seq* sequence object. Return the header line of the next fasta sequence and the newly created sequence. If no new sequence is found, the next header line will be set to None.

Parameters

- **line** (*str*) – Header line of the sequence.
- **fo** (*fo*) – File object of the fasta file.
- **getAlignFlag** (*Boolean*) – If set to true, an additional Boolean value that specifies if a multiple sequence alignment ends, is returned.

Return type (str, Seq) | (str, Seq, Boolean)

libPoMo.fasta.**save_as_vcf**(*faSeq*, *ref*, *VCFFileName*)

Save the given :classL'FaSeq' in VCF format.

In general, we want to convert a fasta file with various individuals with the help of a reference that contains one sequence to a VCF file that contains all the SNPs. This can be done with this function. Until now it is not possible to do this conversion for several chromosomes for each individual in one run. Still, the conversion can be done chromosome by chromosome.

This function saves the SNPs of *faSeq*, a given *FaSeq* (fasta sequence) object in VCF format to the file *VCFFileName*. The reference genome *ref*, to which *faSeq* is compared to, needs to be passed as a *Seq* object.

The function compares all sequences in *faSeq* to the sequence given in *ref*. The names of the individuals in the saved VCF file will be the sequence names of the *faSeq* object.

```
#CHROM = sequence name of the reference
POS    = position relative to reference
ID     = .
REF    = base of reference
ALT    = SNP (e.g. 'C' or 'G,T' if 2 different SNPs are present)
QUAL   = .
FILTER = .
INFO   = .
FORMAT = GT
```

Parameters

- **faSeq** (FaSeq) – *FaSeq* object to be converted.
- **ref** (Seq) – *Seq* object of the reference sequence.
- **VCFFileName** (*str*) – Name of the VCF output file.

2.4 libPoMo.vcf

This module provides functions to read, write and access vcf files.

2.4.1 Objects

Classes:

- *NucBase*, store a nucleotide base
- *VCFStream*, a variant call format (VCF) stream object
- *VCFSeq*, a VCF file sequence object

Exception Classes:

- *NotAVariantCallFormatFileError*
- *NotANucBaseError*

Functions:

- *update_base()*, read a line into a base
- *get_nuc_base_from_line()*, create a new *NucBase* from a line
- *check_fixed_field_header()*, check a VCF fixed field header string
- *get_indiv_from_field_header()*, extract list of individuals from header
- *init_seq()*, open VCF file and initialize *VCFStream*
- *open_seq()*, open VCF file and save it to a *VCFSeq*
- *get_header_line_string()*, print vcf header line

---

exception libPoMo.vcf.**NotANucBaseError**
Exception raised if given nucleotide base is not valid.

exception libPoMo.vcf.**NotAVariantCallFormatFileError**
Exception raised if given VCF file is not valid.

class libPoMo.vcf.**NucBase**
Stores a nucleotide base.

FIXME: Bases are split by '/'. They should also be split by '|'.

A class that stores a single nucleotide base and related information retrieved from a VCF file. Please see http://www.1000genomes.org/ for a detailed description of the vcf format.

Variables

- **chrom** (*str*) – Chromosome name.
- **pos** (*int*) – 1-based position on the chromosome.
- **id** (*str*) – ID.
- **ref** (*str*) – Reference base.
- **alt** (*str*) – Alternative base(s).
- **qual** (*str*) – Quality.
- **filter** (*str*) – Filter.
- **info** (*str*) – Additional information.
- **format** (*str*) – String with format specification.
- **speciesData** (*[str]*) – List with strings of the species data (e.g. 0/1:...).

- **ploidy** (*int*) – Ploidy (number of sets of chromosomes) of the sequenced individuals. Can be set with *set_ploidy()*.

get_alt_base_list()
Return alternative bases as a list.

get_base_ind(*iI*, *iC*)
Return the base of a specific individual.

Parameters

- **indiv** (*int*) – 0-based index of individual.
- **chrom** (*int*) – 0-based index of chromosome (for n-ploid individuals).

Return type character with nucleotide base.

get_info()
Return nucleotide base information string.

get_ref_base()
Return reference base.

Return type char

get_speciesData()
Return species data as a list.

•data[0][0] = data of first species/individual on chromatide A

•**data[0][1] = only set for non-haploids; data of first** species/individual on chromatide B

Sets data[i][j] to None if the base of individual *i* on chromosome *j* could not be read (e.g. it is not valid).

Return type matrix of integers

print_info()
Print nucleotide base information.

Print the stored single nucleotide base and related information from the VCF file.

purge()
Purge the data associated with this *NucBase*.

set_ploidy()
Set self.ploidy.

class libPoMo.vcf.**VCFSeq**
Store data retrieved from a VCF file.

Initialized with *open_seq()*.

Variables

- **name** (*str*) – Sequence name.
- **header** (*str*) – Sequence header.
- **speciesL** (*[str]*) – List with species / individuals.
- **nSpecies** (*int*) – Number of species / individuals.
- **baseL** (*[NucBase]*) – List with stored *NucBase* objects.
- **nBases** (*int*) – Number of *NucBase* objects stored.

append_nuc_base(*base*)
Append *base*, a given *NucBase*, to the VCFSeq object.

get_header_line_string(*indiv*)
Return a standard VCF File header string with individuals *indiv*.

get_nuc_base(*chrom*, *pos*)
Return base at position *pos* of chromosome *chrom*.

has_base(*chrom*, *pos*)
Return True (False) if base is (not) found.

Parameters

- **chrom** (*str*) – Chromosome name.
- **pos** (*int*) – 1-based position on *chrom*.

print_header_line(*indiv*)
Print a standard VCF File header with individuals *indiv*.

print_info(*maxB=50*, *printHeader=False*)
Print VCF sequence information.

Print vcf header, the total number of nucleotides and a maximum of *maxB* bases (defaults to 50). Only prints header if *printHeader* = True is given.

class libPoMo.vcf.**VCFStream**(*seqName*, *vcfFileObject*, *speciesList*, *firstBase*)
Store base data from a VCF file line per line.

It can be initialized with *init_seq()*. This class stores a single base retrieved from a VCF file and the file itself. It is used to parse through a VCF file line by line processing the bases without having to read the whole file at one.

Parameters

- **seqName** (*str*) – Name of the stream.
- **vcfFileObject** (*fo*) – File object associated with the stream.
- **speciesList** (*[str]*) – List with species / individuals.
- **firstBase** (NucBase) – First *NucBase* to be saved.

Variables

- **name** (*str*) – Name of the stream.
- **fo** (*fo*) – Stored VCF file object.
- **speciesL** (*[str]*) – List with species / individuals.
- **nSpecies** (*int*) – Number of species / individuals.
- **base** (*NusBase*) – Stored *NucBase*.

close()
Closes the linked file.

print_info()
Prints VCFStream information.

read_next_base()
Read the next base.

Return position of next base.

Raise a *ValueError* if no next base is found.

libPoMo.vcf.**check_fixed_field_header**(*ln*)
Check if the given line *ln* is the header of the fixed fields.

Sample header line:

```
#CHROM      POS      ID       REF      ALT      QUAL     FILTER  INFO     FORMAT␣
↪SpeciesL
```

libPoMo.vcf.**get_header_line_string**(*indiv*)
Return a standard VCF File header string with individuals *indiv*.

libPoMo.vcf.**get_indiv_from_field_header**(*ln*)
Return species from a fixed field header line *ln*.

Sample header line:

```
#CHROM      POS      ID       REF      ALT      QUAL     FILTER  INFO     FORMAT␣
↪SpeciesL
```

libPoMo.vcf.**get_nuc_base_from_line**(*ln*, *info=False*, *ploidy=None*)
Retrieve base data from a VCF file line *ln*.

Split a given VCF file line and returns a NucBase object. If *info* is set to False, only #CHROM, POS, REF, ALT and speciesData will be read.

Parameters

- **info** (*Bool*) – Determines if info is retrieved from *ln*.
- **ploidy** (*int*) – If ploidy is known and given, it is set.

libPoMo.vcf.**init_seq**(*VCFFileName*, *maxskip=100*, *name=None*)
Open a (gzipped) VCF4.1 file.

Try to open the given VCF file, checks if it is in VCF format. Initialize a *VCFStream* object that contains the first base.

Please close the associated file object with *VCFStream.close()* when you don't need it anymore.

Parameters

- **VCFFileName** (*str*) – Name of the VCF file.
- **maxskip** (*int*) – Only look *maxskip* lines for the start of the bases (defaults to 80).
- **name** (*str*) – Set the name of the sequence to *name*, otherwise set it to the filename.

libPoMo.vcf.**open_seq**(*VCFFileName*, *maxskip=100*, *name=None*)
Open a VCF4.1 file.

Try to open the given VCF file, checks if it is in VCF format and reads the bases(s). It returns an *VCFSeq* object that contains all the information.

Parameters

- **VCFFileName** (*str*) – Name of the VCF file.
- **maxskip** (*int*) – Only look *maxskip* lines for the start of the bases (defaults to 80).
- **name** (*str*) – Set the name of the sequence to *name*, otherwise set it to the filename.

libPoMo.vcf.**update_base**(*ln*, *base*, *info=True*)
Read line *ln* into base *base*.

Split a given VCF file line and returns a *NucBase* object. If *info* is set to False, only #CHROM, REF, ALT and speciesData will be read.

2.5 libPoMo.cf

This model provides functions to read, write and access files that are in counts format.

2.5.1 The Counts Format

This file format is used by PoMo and lists the base counts for every position.

It contains:

- 1 line that specifies the file as counts file and states the number of populations as well as the number of sites
- 1 headerline with tab separated sequence names
- N lines with counts of A, C, G and T bases at position n

It can contain:

- any number of lines that start with a #, these are treated as comments; There are no more comments allowed after the headerline.

```
COUNTSFILE      NPOP 5          NSITES N
CHROM           POS     Sheep           BlackSheep      RedSheep        Wolf            ␣
↪ RedWolf
1       s       0,0,1,0         0,0,1,0         0,0,1,0         0,0,5,0         0,0,0,1
1       s + 1   0,0,0,1         0,0,0,1         0,0,0,1         0,0,0,5         0,0,0,1
.
.
.
9       8373    0,0,0,1         1,0,0,0         0,1,0,0         0,1,4,0         0,0,1,0
.
.
.
Y       end     0,0,0,1         0,1,0,0         0,1,0,0         0,5,0,0         0,0,1,0
```

2.5.2 Convert to Counts Format

To convert a fasta reference file with SNP information from a variant call format (VCF) to counts format use the *CFWriter*. If you want to convert a multiple alignment fasta file, use the *CFWriter* together with the convenience function *write_cf_from_MFaStream()*.

Tabix index files need to be provided for all VCF files. They can be created from the terminal with $(tabix -p vcf "vcf-file.vcf.gz") if tabix is installed.

A code example is:

```
import import_libPoMo
import libPoMo.fasta as fa
import libPoMo.cf as cf

vcfFL = ["/path/to/vcf/file1", "/path/to/vcf/file2", "..."]

cfw = cf.CFWriter(vcfFL, "name-of-outfile")
mFaStr = fa.MFaStream("/path/to/fasta/reference")

cfw.write_HLn()
```

```
cf.write_cf_from_MFaStream(mFaStr, cfw)

cfw.close()
```

2.5.3 Objects

Classes:

- *CFStream*
- *CFWriter*, write a counts format file

Exception Classes:

- *NotACountsFormatFileError*
- *CountsFormatWriterError*
- *NoSynBase*

Functions:

- *interpret_cf_line()*, get data of a line in counts format
- *faseq_append_base_of_cfS()*, append CFStream line to FaSeq
- *cf_to_fasta()*, convert counts file to fasta file
- *write_cf_from_MFaStream()*, write counts file using the given MFaStream and CFWriter
- *fasta_to_cf()*, convert fasta to counts format

---

class libPoMo.cf.**CFStream**(*CFFileName*, *name=None*)

Store data of a CF file line per line.

Open a (gzipped) CF file. The file can be read line per line with *read_next_pos()*.

Parameters

- **CFFileName** (*str*) – Counts format file name to be read.
- **name** (*str*) – Optional; stream name, defaults to stripped filename.

Variables

- **name** (*str*) – Stream name.
- **chrom** (*str*) – Chromosome name.
- **pos** (*str*) – Positional string.
- **fo** (*fo*) – Fileobject.
- **indivL** (*[str]*) – List of names of individuals (populations).
- **countsL** (*[[int]]*) – Numpy array of nucleotide counts.
- **nIndiv** (*int*) – Number of individuals (populations).

read_next_pos()

Get next base.

Return position of next base. Raises *ValueError* if there is no next base.

Return type int

class libPoMo.cf.**CFWriter**(*vcfFileNameL*, *outFileName*, *splitChar='-'*, *mergeL=None*, *nameL=None*, *oneIndividual=False*)

Write a counts format file.

Save information that is needed to write a CF file and use this information to write a CF file. Initialize with a list of vcf file names and an output file name:

```
CFWriter([vcfFileNames], "output")
```

Tabix index files need to be provided for all VCF files. They can be created from the terminal with $(tabix -p vcf "vcf-file.vcf.gz") if tabix is installed.

Before the count file can be written, a reference sequence has to be specified. A single reference sequence can be set with *set_seq()*.

Write a header line to output:

```
self.write_HLn()
```

Write lines in counts format from 1-based positions *start* to *end* on chromosome *chrom* to output:

```
rg = sb.Region("chrom", start, end)
self.write_Rn(rg)
```

If you want to compare the SNPs of the VCF files to a multiple alingment fasta stream (*MFaStream*) consider the very convenient function *write_cf_from_MFaStream()*.

To determine the different populations present in the VCF files, the names of the individuals will be cropped at a specific char that can be set at initialization (standard value = '-'). It is also possible to collapse all individuals of determined VCF files to a single population (cf. mergeL and nameL).

The ploidity has to be set manually if it differs from 2.

Additional filters can be set before the counts file is written (e.g. only write synonymous sites).

Important: Remember to close the attached file objectsL with *close()*. If the CFWriter is not closed, the counts file is not usable because the first line is missing!

Parameters

- **vcfFileNameL** (*[str]*) – List with names of vcf files.
- **outFileName** (*str*) – Output file name.
- **verb** (*int*) – Optional; verbosity level.
- **splitChar** (*char*) – Optional; set the split character so that the individuals get sorted into the correct populations.
- **mergeL** (*[Boolean]*) – Optional; a list of truth values. If *mL[i]* is True, all individuals of *self.vcfL[i]* are treated as one population orspecies independent of their name. The respective counts are summed up. If *self.nL[i]* is given, the name of the summed sequence will be *self.nL[i]*. If not, the name of the first individual in *vcfL[i]* will be used.
- **nameL** (*[str]*) – Optional; a list of names. Cf. *self.mL*.
- **oneIndividual** (*Boolean*) – Optional; pick one individual out of each population.

Variables

- **refFN** (*str*) – Name of reference fasta file.
- **vcfL** (*[str]*) – List with names of vcf files.
- **outFN** (*str*) – Output file name.

- **v** (*int*) – Verbosity.
- **mL** (*[Boolean]*) – A list of truth values. If *mL[i]* is True, all individuals of *self.vcfL[i]* are treated as one population orspecies independent of their name. The respective counts are summed up. If *self.nL[i]* is given, the name of the summed sequence will be *self.nL[i]*. If not, the name of the first individual in *vcfL[i]* will be used.
- **nL** (*[str]*) – A list of names. Cf. *self.mL*.
- **nV** (*int*) – Number of vcf files.
- **vcfTfL** (*[fo]*) – List with *pysam.Tabixfile* objects. Filled by *self.__init_vcfTfL()* during initialization.
- **outFO** (*fo*) – File object of the outfile. Filled by *self.__init_outFO()* during initialization.
- **cD** – List with allele or base counts. The alleles of individuals from the same population are summed up. Hence, *self.cD[p]* gives the base counts of population *p* in the form: [0, 0, 0, 0]. Population *p* does not need to be the one from *self.vcfL[p]* because several populations might be present in one vcf file. *self.assM* connects the individual j from *self.vcfL[i]* such that *self.assM[i][j]* is *p*.
- **chrom** (*str*) – Name of the current chromosome. Set and updated by *write_Rn()*.
- **pos** (*int*) – Current position on chromosome. Set and updated by *write_Rn()*.
- **offset** (*int*) – Value that can be set with *set_offset()*, if the reference sequence does not start at the 1-based position 1 but at the 1-based position *offset*.
- **indM** – Matrix with individuals from vcf files. *self.indM[i]* is the list of individuals found in *self.vcfL[i]*.
- **nIndL** (*[int]*) – List with number of individuals in *self.vcfL[i]*.
- **assM** – Assignment matrix that connects the individuals from the vcf files to the correct *self.cD* index. Cf. *self.cD*
- **nPop** (*int*) – Number of different populations in count format output file (e.g. number of populations). Filled by *self.__init_assM()* during initialization.
- **refSeq** (*Seq*) – *Seq* object of the reference Sequence. This has to be set with *set_seq*.
- **ploidy** (*int*) – Ploidy of individuals in vcf files. This has to be set manually to the correct value for non-diploids!
- **splitCh** (*char*) – Character that is used to split the individual names.
- **onlySynonymous** (*Boolean*) – Only write 4-fold degenerate sites.
- **baseCounter** (*int*) – Counts the total number of bases.
- **__force** (*Boolean*) – If set to true, skip name checks.

add_base_to_sequence(*pop_id*, *base_char*, *double_fixed_sites=False*)

Adds the base given in *base_char* to the counts of population with id *pop_id*. If *double_fixed_sited* is true, fixed sites are counted twice. This makes sense, when heterozygotes are encoded with IUPAC codes.

close()

Write file type specifier, number of populations and number of sites to the beginning of the output file. Close fileobjects.

set_force(*val*)

Sets *self.__force* to *val*.

Parameters **val** (*Boolean*) –

set_offset(*offset*)
Set the offset of the sequence.

Parameters **offset** (*int*) – Value that can be set, if the reference sequence does not start at the 1-based position 1 but at the 1-based position *offset*.

set_ploidy(*ploidy*)
Set the ploidy.

In VCF files, usually the bases of all copies of the same chromosomes are given and separated by '/' or '|'. If the species is not diploid, this ploidy has to be set manually with this function.

set_seq(*seq*)
Set the reference sequence.

write_HLn()
Write the counts format header line to *self.outFN*.

write_Ln()
Write a line in counts format to *self.outFN*.

write_Rn(*rg*)
Write lines in counts format to *self.outFN*.

Parameters **rg** (Region) – *Region* object that determines the region that is covered.

exception libPoMo.cf.**CountsFormatWriterError**
General *CFWriter* object error.

exception libPoMo.cf.**NoSynBase**
Not a 4-fold degenerate site.

exception libPoMo.cf.**NotACountsFormatFileError**
CF file not valid.

libPoMo.cf.**cf_to_fasta**(*cfS*, *outname*, *consensus=False*)
Convert a *CFStream* to a fasta file.

Extracts the sequences of a counts file that has been initialized with an *CFStream*. The conversion starts at the line pointed to by the *CFStream*.

If more than one base is present at a single site, one base is sampled out of all present ones according to its abundance.

If consensus is set to True, the consensus sequence is extracted (e.g., no sampling but the bases with highest counts for each individual or population are chosen).

Parameters

- **cfS** (CFStream) – Counts format file stream.
- **outname** (*str*) – Fasta output file name.
- **consensus** (*Boolean*) – Optional; Extract consensus sequence? Defaults to False.

libPoMo.cf.**faseq_append_base_of_cfS**(*faS*, *cfS*, *consensus=False*)
Append a *CFStream* line to an *libPoMo.fasta.FaSeq*.

Randomly chooses bases for each position according to their abundance.

Parameters

- **faS** (FaSeq) – Fasta sequence to append base to.
- **cfS** (CFStream) – CFStream containing the base.

libPoMo.cf.**fasta_to_cf**(*fastaFN*, *countsFN*, *splitChar='-'*, *chromName='NA'*, *double_fixed_sites=False*)

Convert fasta to counts format.

The (aligned) sequences in the fasta file are read in and the data is written to a counts format file.

Sequence names are stripped at the first dash. If the strupped sequence name coincide, individuals are put into the same population.

E.g., homo_sapiens-XXX and homo_sapiens-YYY will be in the same population homo_sapiens.

Take care with large files, this uses a lot of memory.

The input as well as the output files can additionally be gzipped (indicated by a .gz file ending).

Variables **double_fixed_sites** (*bool*) – Set to true if heterozygotes are

encoded with IUPAC codes. Then, fixed sites will be counted twice so that the level of polymorphism stays correct.

libPoMo.cf.**interpret_cf_line**(*ln*)

Interpret a counts file line.

Return type is a tuple containing the chromosome name, the position and a list with nucleotide counts (cf. counts file).

Parameters **ln** (*str*) – Line in counts format.

Return type (str, int, [[int]])

libPoMo.cf.**weighted_choice**(*lst*)

Choose element in integer list according to its value.

E.g., in [1,10], the second element will be chosen 10 times as often as the first one. Returns the index of the chosen element.

Variables **lst** (*[int]*) – List of integers.

Return type int

libPoMo.cf.**write_cf_from_MFaStream**(*refMFaStr*, *cfWr*)

Write counts file using the given MFaStream and CFWriter.

Write the counts format file using the first sequences of all alignments in the MFaStream. The sequences are automatically reversed and complemented if this is needed (indicated in the header line). This is very useful if you e.g. want to compare the VCF files to a CCDC alignment.

Parameters

- **refMFaStr** (*FMaStream*) – The reference *MFaStream*.
- **cfWf** (CFWriter) – The *CFWriter* object that contains the VCF files.

libPoMo.cf.**write_cf_from_gp_stream**(*gp_stream*, *cfWr*)

Write counts file using a given GP stream with reference and CFWriter.

Write the counts format file using all genes in the GP stream. The sequences are automatically reversed and complemented if this is needed.

Parameters

- **gp_stream** (*GPStream*) – The GP stream and reference GPStream.
- **cfWf** (CFWriter) – The *CFWriter* object that contains the VCF files.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

l

O

P

R

S

T

U

V

W