PPP Documentation

Release 0.1.13

PPP Development Team

Feb 11, 2022

Contents:

The Popgen Pipeline Platform (PPP) is a software platform with the goal of reducing the computational expertise required for conducting population genomic analyses. The PPP was designed as a collection of scripts that facilitate common population genomic workflows in a consistent and standardized environment. Functions were developed to encompass entire workflows, including: input preparation, file format conversion, various population genomic analyses, and output generation. By facilitating entire workflows, the PPP offers several benefits to prospective end users - it reduces the need of redundant in-house software and scripts that would require development time and may be error-prone, or incorrect, depending on the expertise of the investigator. The platform has also been developed with reproducibility and extensibility of analyses in mind.

Please Note: This documentation is currently being developed and will be updated freqeuntly in the coming days

CHAPTER 1

Introduction

The Popgen Pipeline Platform (PPP) was written using the Python programming language and designed to operate using Python 3.7. In comparison to a fixed pipeline, the PPP was designed as a collection of modular functions that may combined to generate a wide variety of analyses and pipelines.

For simplicity, PPP functions are separated into four categories: # **Core functions**: Frequently used methods and procedures in population genomic pipelines (e.g. phasing, filtering, four-gamete test, etc.). # **Input file** generators: Input generators for creating the necessary input for population genomic analysis (e.g. generating input for IMa3, TreeMix, G-PhoCS, etc.) # **Analyses**: Common population genomic analyses (e.g. isolation and migration, admixture, linkage disequilibrium, etc.) # **Utilities**: Simple file-specific procedures often required in population genomic pipelines

For details on specific functions, please see the documentation on each section.

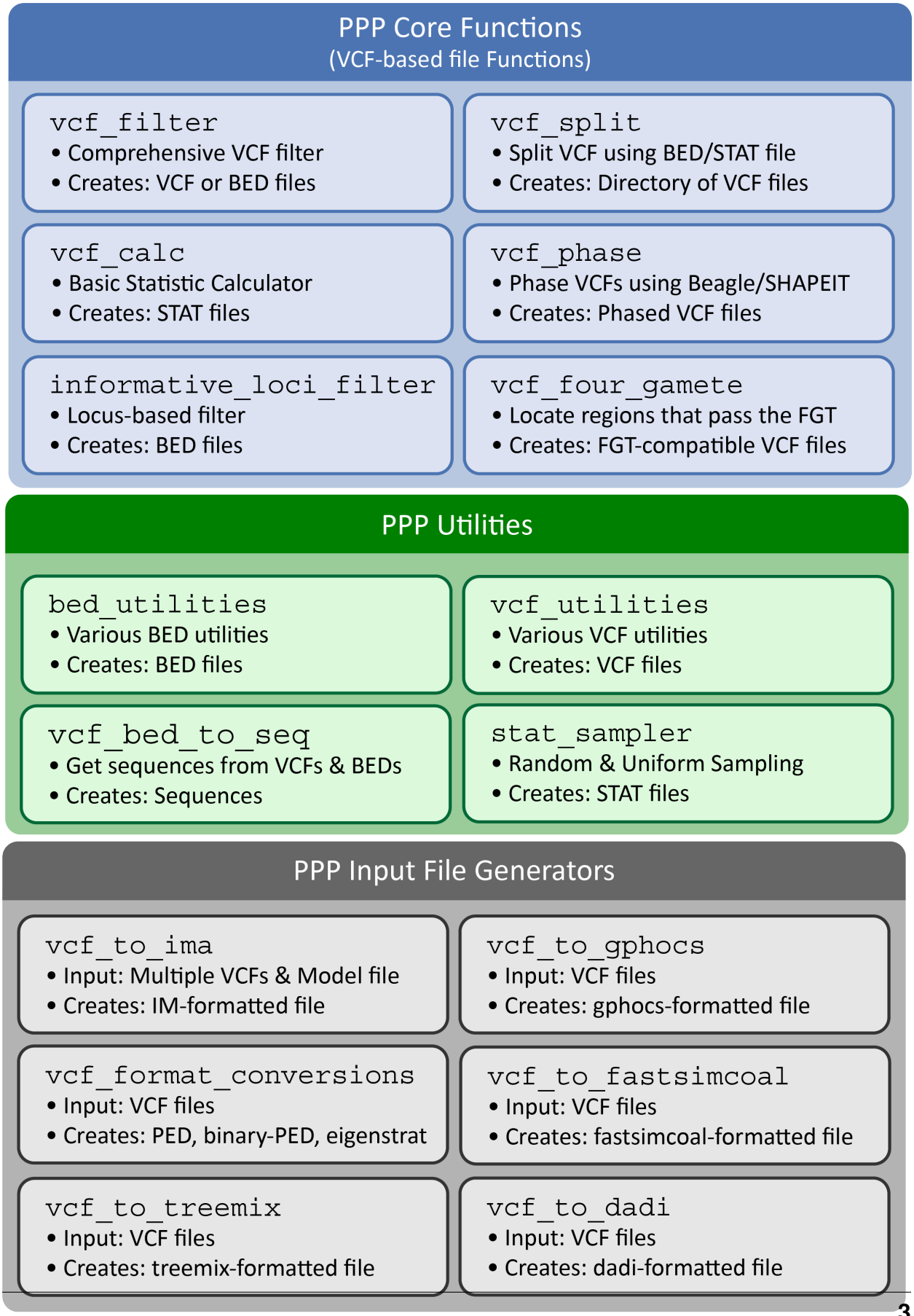

PPP Core Functions
(VCF-based file Functions)
vcf_filter
• Comprehensive VCF filter
• Creates: VCF or BED files
vcf_split
• Split VCF using BED/STAT file
• Creates: Directory of VCF files
vcf_calc
• Basic Statistic Calculator
• Creates: STAT files
vcf_phase
• Phase VCFs using Beagle/SHAPEIT
• Creates: Phased VCF files
informative_loci_filter
• Locus-based filter
• Creates: BED files
vcf_four_gamete
• Locate regions that pass the FGT
• Creates: FGT-compatible VCF files
PPP Utilities
bed_utilities
• Various BED utilities
• Creates: BED files
vcf_utilities
• Various VCF utilities
• Creates: VCF files
vcf_bed_to_seq
• Get sequences from VCFs & BEDs
• Creates: Sequences
stat_sampler
• Random & Uniform Sampling
• Creates: STAT files
PPP Input File Generators
vcf_to_ima
• Input: Multiple VCFs & Model file
• Creates: IM-formatted file
vcf_to_gphocs
• Input: VCF files
• Creates: gphocs-formatted file
vcf_format_conversions
• Input: VCF files
• Creates: PED, binary-PED, eigenstrat
vcf_to_fastsimcoal
• Input: VCF files
• Creates: fastsimcoal-formatted file
vcf_to_treemix
• Input: VCF files
• Creates: treemix-formatted file
vcf_to_dadi
• Input: VCF files
• Creates: dadi-formatted file
PPP Analyses

Figure 1: Structure of the PPP

1.1 Creating Pipelines

Most PPP-based pipelines are expected to primarily consist of core functions. To simplify development, all core functions were designed to operate using VCF-based files. The VCF format was selected due to the frequent support for the format among publicly available datasets and population genomics software. At present, pipelines may be generated in one of two methods: i) calling each function by command-line or ii) calling the function within a script, such as a jupyter notebook. Example usage of both methods may be found within examples.

CHAPTER 2

Installation

2.1 From PyPi

The PPP can also be easily installed via the PyPi repository via pip:

```
pip install py-popgen
```

2.2 From Conda

The PPP has conda packages available for python versions 3.6 and 3.7. To install in a clean environment, run the following:

```
conda create -n py-popgen python=3.7.7
conda activate py-popgen
conda install -c jaredgk -c bioconda py-popgen
```

2.3 From Source

The most current version of the PPP can be installed by obtaining the source code from the PPP GitHub repository. This can be done with:

```
git clone https://github.com/jaredgk/PPP
```

To install the local repository copy and allow edits to the source code to be included with imports without any additional steps, run the following commands:

```
cd PPP
pip install -e .
```

To install the repository without pip, run the following (note that any modifications to the source code will not be used at runtime unless the setup command is run again):

```
cd PPP
python setup.py install
```

2.4 Dependencies

If installing PPP from source, multiple python and non-python depencencies must also be installed.

2.4.1 Python Dependencies

The PPP requries a number of python libraries, including:

- The SciPy Ecosystem (i.e. numpy, scipy, pandas, matplotlib, etc.)
- Pysam
- Biopython
- Cython
- rpy2

We recommend users install and maintain these libraries using either pip or Anaconda 3.

2.4.2 Other Dependencies

The PPP also requries a number of executables to be installed, including:

- VCFtools
- BCFtools, Samtools, and HTSlib
- plink 1.9
- plink 2.0

- SHAPEIT
- Beagle 5.0
- Picard

Please note that VCFtools, BCFtools, Samtools, HTSlib, plink 1.9, plink 2.0, and SHAPEIT may be installed using Anaconda 3.

CHAPTER 3

Usage and Examples

3.1 Command-line Usage and Examples

PPP functions may be called at the command-line as shown in this example:

```
vcf_filter.py --vcf examples/files/merged_chr1_10000.vcf.gz --filter-
↪only-biallelic --out-format bcf
```

Details on the usage and arguments of each function may be found within the relevant documentation. In addition, all files specified within these examples may be found within the **examples/files** directory of the PPP repository.

3.2 Module Usage and Examples

PPP functions may also be imported from the **pgpipe** module for use within a python script or a Jupyter Notebook as shown in this example:

```
import pgpipe.vcf_filter as vcf_filter

vcf_filter.run(vcf = 'examples/files/merged_chr1_10000.vcf.gz', filter_
↪only_biallelic = True, out_format = 'bcf')
```

In comparison to calling functions at the command-line, imported functions require:

- The pgpipe module must be imported

- Each function is called using .run(): **vcf_filter.run() or pgpipe.vcf_filter.run()**
- The use of underscores within arguments rather than dashes: **--out-format vs. out_format**
- Setting the value to True when arguments do not require a value: **--filter-only-biallelic vs. filter_only_biallelic = True**

3.2.1 Example Jupyter Notebooks

We have included two example notebooks within the **examples/jupyter** directory of the PPP repository.

Example Jupyter Pipleine

[1]:
```
import sys
import os
import subprocess

from pgpipe import four_gamete, vcf_split_pysam, vcf_to_ima, vcf_filter,
 ↪ vcf_calc, vcf_sampler, vcf_phase, stat_sampler, vcf_split
from pgpipe.logging_module import initLogger
from pgpipe.informative_loci_filter import filter_bed_regions
from pgpipe.subtract_bed import filter_stat
import pysam

print ("Imports complete")
```

```
Imports complete
```

Setting Filepaths

The required input files for a PPP run are: - A genome VCF of the target populations (plus a tabix index if bgzipped) - A population model file

The population model file is a JSON-formatted file that defines population names and the individuals from the VCF that belong to each population. This file can be created using the model_creator function, or by creating it manually by using an example model file as a template.

For region filtering, the following should be provided: - Name for target region file - Name for final selected region file - File with genic regions/regions to be excluded from analysis (optional) - File with regions to be selected from for analysis (optional)

There are two methods for obtaining target regions for subsampling and analysis, with some level of interoperability: using a statistic file to sample regions, or use a file with target regions to randomly select regions with enough sites to be valid in an IM analysis. To generate a statistic file, use

vcf_calc to read over the input VCF file, then use stat_sampler to select either a random or uniform distribution of these regions given their statistic value. These stat files can be filtered with a genic region file, available from UCSC, using the subtract_bed function. This method is implemented in this notebook, with the genic region filtering offered as an optional cell.

An additional method of region selection, done without statistics, can be done by downloading region files for genic regions, and optionally for STR regions and regions with missing data. If one wants to find a set of target regions that are intergenic and outside of STR regions, download the corresponding files from UCSC for your species and use invert_bed_region to 'invert' the files, optionally selecting only regions outside of a set number of bases from the regions in the file with the –window option. If files don't have typical BED column order, use the –bed-column-index option to provide a comma-separated string with the 0-based index of the start, end, and chromosome column. (For normal BED files, this would be '1,2,0') The get_nonmissing_chunks function can scan the input VCF and find regions with no missing data. All of these can be combined with bedtools to select regions that overlap with all three possible files.

Whichever method used, the target region file should be run through the informative_filter function to check the VCF file has enough biallelic, informative SNPs (two or more of each of two alleles) to have a good chance of passing the four-gamete test (which requires regions with at least two SNPs). An informative count of 5 will usually allow for this, but if you have limited regions this threshhold can be lowered to 3.

In addition to these files, additional functionality such as CpG filtering and comprehensive logging can be used by providing: - A reference FASTA (for CpG filtering, can be bgzipped or unzipped but requires indexing w/faidx) - A log filename

This section also is used to set up the directory structure for data files, a working directory, a directory for VCF region files, a target number of loci, and names for various stages of loci VCF files (phasing, four-gamete testing, potentially filtering individuals with missing data).

[2]:
```
#Set up directories and filepaths, run on all restarts
work_dir='/home/jared/workspace/projects_ppp/notebook_sample/'
#data_dir='/media/ccgg/ppp_sample_data/'
data_dir=work_dir
vcf_dir = work_dir+'vcfs/'


main_vcf_name = data_dir+'pan_chr20.vcf.gz'
filtered_vcf_name = data_dir+'pan_chr20_filtered.vcf.gz'
stat_file_name = work_dir+'fst_regions.bed'
model_file = data_dir+'great_ape.model'
int_bed_file = work_dir+'regions_for_sampling.bed'
target_loci_file = work_dir+'target_loci.bed'
ima_input_file = work_dir+'test_run_input.ima.u'
#subsamp_bed_file = work_dir+'great_ape_genome2/5k_sample.bed'
logfile = '/home/jared/testpppj.log'
```

(continues on next page)

(continued from previous page)

```
loci=200

region_files = [vcf_dir+'Sampled_nonmissing/Sample_'+str(i)+'.vcf' for␣
↪i in range(loci)]
phased_files = [vcf_dir+'Phased/phased_'+str(i)+'.vcf' for i in␣
↪range(loci)]
fourg_files = [vcf_dir+'four_gamete/Sample_'+str(i)+'.vcf' for i in␣
↪range(loci)]
passed_files = []
```

[12]:
```
#Set up directory structure, only needs to be run once
if not os.path.exists(vcf_dir):
    os.makedirs(vcf_dir)
    os.makedirs(vcf_dir+'four_gamete/')
    os.makedirs(vcf_dir+'Sampled_nonmissing/')
    os.makedirs(vcf_dir+'Phased/')
```

The vcf_filter step will filter the original VCF according to many conditions, including: - Non-biallelic sites (–filter-min-alleles and –filter-max-alleles) - Sites with missing data (–filter-max-missing) - Indels - Sites on non-autosomal chromosomes (–filter-exclude-chr) - Individuals not named in model file (use –model-file to input model file, with –model if multiple models included in file)

[4]:
```
#Creates VCF filtered for no missing data and biallelic sites
vcf_filter.run(['--vcf', main_vcf_name, '--filter-max-missing', '1.0',
↪'--model-file',model_file,
                '--model','2Pop', '--filter-min-alleles', '2', '--
↪filter-max-alleles', '2', '--out-format',
                'vcf.gz', '--out', filtered_vcf_name, '--filter-exclude-
↪chr', 'chrX', 'chrY', '--overwrite'])

pysam.tabix_index(filtered_vcf_name,preset='vcf')
print("Filtering complete")
```

```
Filtering complete
```

The vcf_calc step will, for every 10kb window in the genome, calculate Fst given populations from the model file. Statistics that can be filtered over currently include: - Windowed pi - Tajima's D - Fst

[3]:
```
#Calculates f_st statistics across genome
vcf_calc.run(['--vcf', filtered_vcf_name, '--out', stat_file_name,
              '--calc-statistic', 'windowed-weir-fst', '--model', '2Pop
↪', '--statistic-window-size',
              '10000', '--statistic-window-step', '10000', '--model-
↪file', model_file, '--overwrite'])
```

(continues on next page)

(continued from previous page)

```
print("Stat calculation complete")
```

```
Stat calculation complete
```

Using the informative site filter, the regions produced by the statistics generation can be checked for whether they contain enough informative sites to pass the four-gamete test. Additional filtering can be done here if it hasn't been done before.

[6]:
```
#Selects subset of regions for fast sampling
filter_bed_regions(['--vcf',filtered_vcf_name,'--bed',stat_file_name,
                    '--remove-indels','--minsites','3','--keep-full-
↪line','--out',int_bed_file,
                    '--randcount','5000','--remove-multi'])
print("BED regions selected")
```

```
BED regions selected
```

```
Only 2848 of 5000 regions found
```

(Optional) If a file with genic regions is provided, statistic windows that overlap those regions can be removed from potential loci. The window option can be used to extend exclusion regions by a set number of base pairs up AND downstream of regions in the filter file. Zero-ho indicates that files use a zero-based, half open interval representation, as opposed to the general 1-based, closed region format.

[3]:
```
int_bed_file2 = work_dir+'regions_for_sampling_nogenes.bed'
gene_file = work_dir+'hg18_chr22_genes.bed'
filter_stat(['--stat-file',int_bed_file,'--filter-file',gene_file,'--
↪window','10000','--zero-ho',
             '--out',int_bed_file2])
int_bed_file = int_bed_file2
```

```
WARNING:root:1773 of 2848 regions selected as non-overlapping
```

The statistic file can be filtered in one of two ways: randomly or uniformly. A random sample (which conceptually doesn't require a statistic) will select sample-size numer of loci for analysis, while a uniform sample will attempt to create a uniform distribution of the chosen statistic. The samples will be placed into a set number of bins, which must be divisible by the number of target loci.

[4]:
```
stat_sampler.run(['--statistic-file',int_bed_file,'--out',target_loci_
↪file,'--sampling-scheme','uniform',
                  '--uniform-bins','5','--sample-size',str(loci),'--calc-
↪statistic','windowed-weir-fst','--overwrite'])
```

This function creates loci VCF files from the full-genome VCF, while optionally doing additional filtering

[4]:
```
#Uniformly sample regions for subset of 200 loci
#vcf_sampler.run(['--vcf', filtered_vcf_name, '--statistic-file',
#                 target_loci_file, '--out-format', 'vcf', '--calc-
 ↪statistic', 'windowed-weir-fst',
#                 '--sampling-scheme', 'uniform', '--uniform-bins', '5',
 ↪ '--out-dir',
#                 work_dir + 'great_ape_genome2/Sample_Files', '--
 ↪overwrite'])
#vcf_split.run(['--vcf',filtered_vcf_name,'--split-method','statistic-
 ↪file','--out-format','vcf','--out-prefix',
#              vcf_dir+'Sampled_nonmissing/Sample','--split-file',
 ↪target_loci_file])
vcf_split_pysam.vcf_region_write([filtered_vcf_name,'--bed',target_
 ↪loci_file,'--out-prefix',vcf_dir+'Sampled_nonmissing/Sample_',
                                   '--remove-indels','--remove-multi','--
 ↪bed-column-index','2,3,1',
                                    '--informative-count','2'])

print("Sampling complete")
```

```
Sampling complete
```

Each locus must be prepared for IM analysis, which involves finding a subregion of each locus that passes the four-gamete test. The four-gamete test is passed if all pairs of alleles in a region have less than four gametes among them. For example, if two SNPs are A/C and G/T, there are four possible gamete haplotypes among them: AG, AT, CG, and CT. If haplotypes with all four of these are present, this indicates that there must have been a recombination event between them at some point in the sampled populations. This violates the IM model, so these regions would fail the test. The four-gamete code as implemented will compute all regions that pass the four-gamete test in a locus VCF file, then select a region either at random or with the largest number of informative sites. A minimum number of informative sites can be set, which defaults to two.

Before the four-gamete test, the locus VCF files need to be phased. This pipeline provides two phasing programs, beagle and shapeit.

[5]:
```
#Phase locus
for i in range(loci):
    vcf_phase.run(['--vcf',region_files[i],'--phase-algorithm','shapeit
 ↪','--out',
                   phased_files[i],'--out-format','vcf','--overwrite'])
print ("Phasing done")
```

```
Phasing done
```

Once phasing is done, each file must be filtered through the four-gamete test. The four-gamete test is a method for determining whether or not there has been recombination between a pair of variants. To do this, all individuals have haplotypes defined as the variants at the two sites. Given two snps

with ref/alt alleles A/G and C/T, if individuals in this sample have haplotypes AC, AT, and GT, it is possible that there has been no recombination between these alleles. If an additional individual has the GC haplotype, this means that a recombination event must have taken place between the sites. This function will return a subregion of the region in the contained VCF that passes the four-gamete test with at least two informative (ac>1) SNPs. If no valid region is found, no VCF is created and the region is skipped for downstream analysis.

[6]:
```
#Subsample locus for four-gamete compatible interval, if no subregion␣
↪returned, do not use VCF
passed_files = []
for i in range(loci):
    ret = four_gamete.sample_fourgametetest_intervals(['--vcfs',␣
↪phased_files[i], '--out',
                                                       fourg_files[i],
↪'--4gcompat', '--reti', '--right',
                                                       '--numinf', '2
↪'])
    if ret[0] is not None:
        passed_files.append(fourg_files[i])
print ("Four gamete regions selected for %d loci"%(len(passed_files)))
```

```
Four gamete regions selected for 199 loci
```

This converts the files that pass the four-gamete test into a single IMa input file. Required arguments are a list of VCF files, and a model file. Filtering options are also available if unwanted sites haven't been filtered out at a previous step.

[7]:
```
#Create IMa input file
ima_args = ['--vcfs']
ima_args.extend(passed_files)
ima_args.extend(['--model-file', model_file, '--model','2Pop','--out',␣
↪work_dir + 'ima_all_loci.ima.u'])

vcf_to_ima.vcf_to_ima(ima_args)
print ("IMa input created")
```

```
IMa input created
```

If desired, this block will run admixture to determine the population assignments of the various populations in the input VCF files. The plot will indicate, for each individual, an estimate of how much of their ancestry comes from the populations determined by clustering in admixture.

[ ]:
```
#Admixture analysis, optional
from pgpipe import convert, admixture, graph_plotter
phased_string = ' '.join(phased_files)
loci_vcf = vcf_dir+'Phased/phased_merged.vcf.gz'
concatcall = subprocess.Popen('vcf-concat '+phased_string+ ' | bgzip -
↪c > '+loci_vcf, shell=True,stdout=subprocess.PIPE)
```

(continues on next page)

(continued from previous page)

```
temp_out, temp_err = concatcall.communicate()
convert.run(['--vcf',loci_vcf,'--out-format','binary-ped','--out-prefix
↪',vcf_dir+'great_ape','--overwrite'])
admixture.run(['--file',vcf_dir+'great_ape.bed','--pop','2'])
graph_plotter.bar_plot(vcf_dir+'great_ape.2.Q')
print ("Plots created")
```

[ ]:

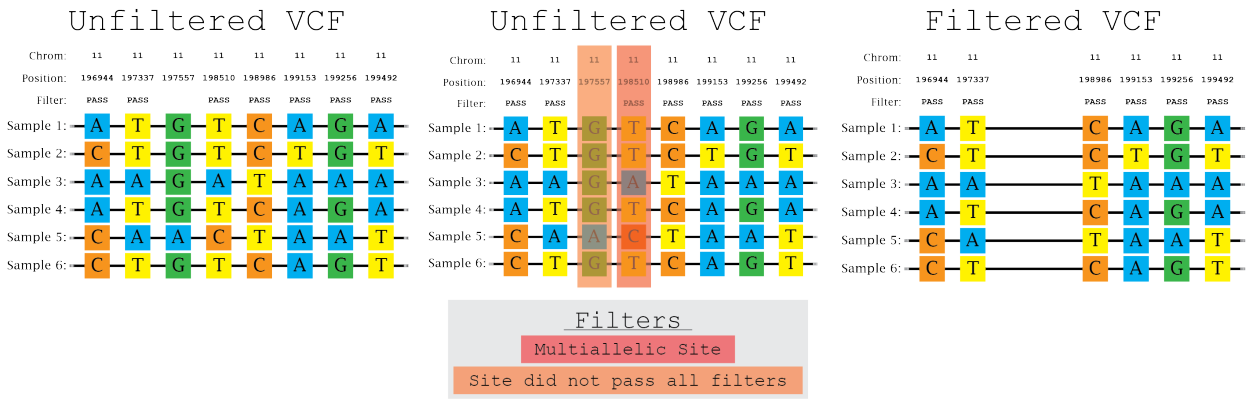
CHAPTER 4

PPP Core Functions

The functions below were developed to perform many of the core operations typically used in population genetic analyses. Each of these functions were designed to perform a single operation (i.e. filtering, phasing, etc.).

4.1 vcf_filter.py: VCF Filter Function

Depending on the analysis being conducted, a number of variant sites and/or samples may be unsuitable and must be removed. Given an unfiltered VCF and the desired filters, vcf_filter will apply the filters and produce a filtered VCF. Filters may be used independently or combined as needed. In addition, a number of the filters are seperated into two types: include (include/keep all relevant variant sites or samples) and exclude (exclude/remove all relevant variant sites or samples).



In this illustration of the filtering process (within a locus of interest), variant sites were kept only if

they: i) were biallelic and ii) passed all filters. These requirements resulted in the removal of two variant sites (i.e. 197557 and 198510) within the given locus.

4.1.1 Command-line Usage

The VCF file filter may be called using the following command:

```
vcf_filter.py
```

Example usage

Command-line to create a BCF with only biallelic sites:

```
vcf_filter.py --vcf examples/files/merged_chr1_10000.vcf.gz --filter-
↪only-biallelic --out-format bcf
```

Command-line to only include variants on chr1 from 1 to 1509546:

```
vcf_filter.py --vcf examples/files/merged_chr1_10000.bcf --filter-
↪include-pos chr1:1-1509546
```

Command-line to remove indels and ouput a BCF file:

```
vcf_filter.py --vcf examples/files/merged_chr1_10000.indels.vcf.gz --
↪filter-exclude-indels --out-format bcf
```

4.1.2 Dependencies

- BCFtools

4.1.3 Input Command-line Arguments

--vcf <*input_filename*> Argument used to define the filename of the VCF file to be filtered.

--model-file <*model_filename*> Argument used to define the model file. Please note that this argument cannot be used with the individual-based filters.

--model <*model_str*> Argument used to define the model (i.e. the individual(s) to include). Please note that this argument cannot be used with the individual-based filters.

4.1.4 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--out-format <*vcf, vcf.gz, bcf, bed, sites*> Argument used to define the desired output format. Formats include: uncompressed VCF (vcf); compressed VCF (vcf.gz) [default]; BCF (bcf); variants in bed format; or variants in sites format.

--overwrite Argument used to define if previous output should be overwritten.

4.1.5 Filter Command-line Arguments

The filtering arguments below are roughly seperated into catagoires. Please not that mulitple filters are seperated into two opposing function types **include** and **exclude**.

Individual-Based Arguments

Please note that all individual-based arguments are not compatible with either the **--model** or **--model-file** command-line arguments.

--filter-include-indv <*indv_str*> <*indv1_str, indv2_str, etc.*> Argument used to define the individual(s) to include. This argument may be used multiple times if desired.

--filter-exclude-indv <*indv_str*> <*indv1_str, indv2_str, etc.*> Argument used to define the individual(s) to exclude. This argument may be used multiple times if desired.

--filter-include-indv-file <*indv_filename*> Argument used to define a file of individuals to include.

--filter-exclude-indv-file <*indv_filename*> Argument used to define a file of individuals to exclude.

Allele/Genotype-Based Arguments

--filter-only-biallelic Argument used to only include variants that are biallelic.

--filter-min-alleles *min_int* Argument used to include variants with a number of allele >= to the given number.

--filter-max-alleles *max_int* Argument used to include variants with a number of allele <= to the given number.

--filter-maf-min *maf_proportion* Argument used to include variants with equal or greater MAF values.

--filter-maf-max *maf_proportion* Argument used to include variants with equal or lesser MAF values.

--filter-mac-min *mac_int* Argument used to include variants with equal or greater MAC values.

--filter-mac-max *mac_int* Argument used to include variants with equal or lesser MAC values.

--filter-include-indels Argument used to include variants if they contain an insertion or a deletion.

--filter-exclude-indels Argument used to exclude variants if they contain an insertion or a deletion.

--filter-include-snps Argument used to include variants if they contain a SNP.

--filter-exclude-snps Argument used to exclude variants if they contain a SNP.

--filter-include-snp *<rs#> <rs#1, rs#2, etc.>* Argument used to include SNP(s) with the matching ID. This argument may be used multiple times if desired.

--filter-exclude-snp *<rs#> <rs#1, rs#2, etc.>* Argument used to exclude SNP(s) with the matching ID. This argument may be used multiple times if desired.

--filter-include-snp-file *<snp_filename>* Argument used to define a file of SNP IDs to include.

--filter-exclude-snp-file *<snp_filename>* Argument used to define a file of SNP IDs to exclude.

--filter-max-missing *proportion_float* Argument used to filter positions by their proportion of missing data, a value of 0.0 allows for no missing whereas a value of 1.0 ignores missing data.

--filter-max-missing-count *count_int* Argument used to filter positions by the number of samples with missing data, a value of 0 allows for no samples to have missing data.

Position-Based Arguments

--filter-include-pos *<chr, chr:pos, chr:start-end, chr:start->* Argument used to include matching positions. May be used to include: an entire chromosome (i.e. chr); a single position (i.e. chr:pos); a chromosomal locus (i.e. chr:start-end); or a chromosomal span (i.e. chr:start-/chr:0-end). This argument may be used multiple times if desired.

--filter-exclude-pos *<chr, chr:pos, chr:start-end, chr:start->* Argument used to exclude matching positions. May be used to exclude: an entire chromosome (i.e. chr); a single position (i.e. chr:pos); a chromosomal locus (i.e. chr:start-end); or a chromosomal span (i.e. chr:start-/chr:0-end). This argument may be used multiple times if desired.

--filter-include-pos-file *<position_filename>* Argument used to define a file of positions to include within a tsv file (chromosome and position).

--filter-exclude-pos-file *<position_filename>* Argument used to define a file of positions to exclude within a tsv file (chromosome and position).

--filter-include-bed *<position_bed_filename>* Argument used to define a BED file of positions to include. Please note that filename must end in .bed.

--filter-exclude-bed *<position_bed_filename>* Argument used to define a BED file of positions to exclude. Please note that filename must end in .bed.

Flag-Based Arguments

--filter-include-passed Argument used to include positions with the 'PASS' filter flag.

--filter-exclude-passed Argument used to exclude positions with the 'PASS' filter flag.

--filter-include-filtered *<filter_flag>* Argument used to include positions with the specified filter flag.

--filter-exclude-filtered *<filter_flag>* Argument used to exclude positions with the specified filter flag.

4.1.6 Other Command-line Arguments

--force-samples Argument used to ignore the error rasied when a sample that does not exist within the input VCF.

4.2 vcf_calc.py: VCF Statistic Calculator Function

Automates the calculation of site/windowed fixation index (Fst), Tajima's D, nucleotide diversity (Pi), allele frequency, and heterozygosity using VCFTools. If no statistic is specified, windowed Fst is used by default.

4.2.1 Command-line Usage

The VCF statistic calculator may be called using the following command:

```
vcf_calc.py
```

Example usage

Command-line to calculate Tajima's D:

```
vcf_calc.py --vcf examples/files/merged_chr1_10000.vcf.gz --calc-
↪statistic TajimaD --statistic-window-size 10000
```

Command-line to calculate windowed Fst on the two populations within the model *2Pop*:

```
vcf_calc.py --vcf examples/files/merged_chr1_10000.vcf.gz --model-file␣
↪examples/files/input.model --model 2Pop --calc-statistic windowed-
↪weir-fst --statistic-window-size 10000 --statistic-window-step 10000
```

4.2.2 Dependencies

- VCFtools

4.2.3 Input Command-line Arguments

--vcf <*input_filename*> Argument used to define the filename of the VCF file for calculations.

--model-file <*model_filename*> Argument used to define the model file. Please note that this argument cannot be used with the **--pop-file** argument or individual-based filters.

--model <*model_str*> Argument used to define the model (i.e. the individual(s) to include and/or the populations for relevant statistics). May be used with any statistic. Please note that this argument cannot be used with **--pop-file** argument or the individual-based filters.

4.2.4 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**. Cannot be used if multiple output files are created.

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--out-dir <*output_dir_name*> Argument used to define the output directory. Only used if 3+ populations are specified.

--overwrite Argument used to define if previous output should be overwritten.

4.2.5 Statistic Command-line Specification

--calc-statistic <*weir-fst, windowed-weir-fst, TajimaD, site-pi, window-pi, freq, het-fit, het-fis, hardy-weinberg*> Argument used to define the statistic to be calculated. Site Fst (weir-fst), windowed Fst (windowed-weir-fst), Tajima's D (TajimaD), site nucleotide diversity (site-pi), windowed nucleotide diversity (window-pi), allele frequency (freq), Fit (het-fit), Fis (het-fis), and the hardy-weinberg equilibrium (hardy-weinberg).

Models with 3+ populations

If a model is specified with 3 or more populations, the following statistics will result in the creation of an output directory - see **--out-dir** - of pairwise comparisons: *weir-fst*, *windowed-weir-fst*, *site-pi*, *window-pi*.

Statistic Command-line Requirements and Options

It should be noted that some of the statistics in the VCF calculator require additional arguments (i.e. **--pop-file**, **--statistic-window-size**, **--statistic-window-step**). These statistics may be found below with their additional requirements and optional arguments. If a statistic is not given, only the statistic specification (i.e. **--calc-statistic**) is required.

--calc-statistic *weir-fst* Requires: **--pop-file/--model**.

--calc-statistic *windowed-weir-fst* Requires: **--pop-file/--model** and **--statistic-window-size**. If **--statistic-window-step** is not given, it will default to the value of **--statistic-window-size**.

--calc-statistic *TajimaD*

Requires: **--statistic-window-size**

--calc-statistic *site-pi* Optional: **--pop-file/--model**.

--calc-statistic *windowed-pi* Requires: **--statistic-window-size**. . If **--statistic-window-step** is not given, it will default to the value of **--statistic-window-size**. Optional: **--pop-file/--model**.

--calc-statistic *het-fis* Requires: **--pop-file/--model**.

Additional Statistic Command-line Arguments

--statistic-window-size <*size_int*> Defines the statistic window size. Not usable with all statistics.

--statistic-window-step <*step_int*> Defines the statistic window step size. Not usable with all statistics.

--pop-file <*pop_filename*> Population file. This argument may be used multiple times if desired. Please note the this argument is not compatible with either the **--model** or **--model-file** command-line arguments.

4.2.6 Filter Command-line Arguments

If using an unfiltered VCF file (e.g. reduce the creation of unnecessary large files) the VCF calculator is able to use either a kept or removed sites/BED file and the individual-based paramemeters.

Individual-Based Arguments

Please note that all individual-based arguments are not compatible with either the **--model** or **--model-file** command-line arguments.

--filter-include-indv <*indv_str*> <*indv1_str, indv2_str, etc.*> Argument used to define the individual(s) to include. This argument may be used multiple times if desired.

--filter-exclude-indv <*indv_str*> <*indv1_str, indv2_str, etc.*> Argument used to define the individual(s) to exclude. This argument may be used multiple times if desired.

--filter-include-indv-file <*indv_filename*> Argument used to define a file of individuals to include.

--filter-exclude-indv-file <*indv_filename*> Argument used to define a file of individuals to exclude.

Position-Based Arguments

--filter-include-positions <*position_filename*> Argument used to define a file of positions to include within a tsv file (chromosome and position).

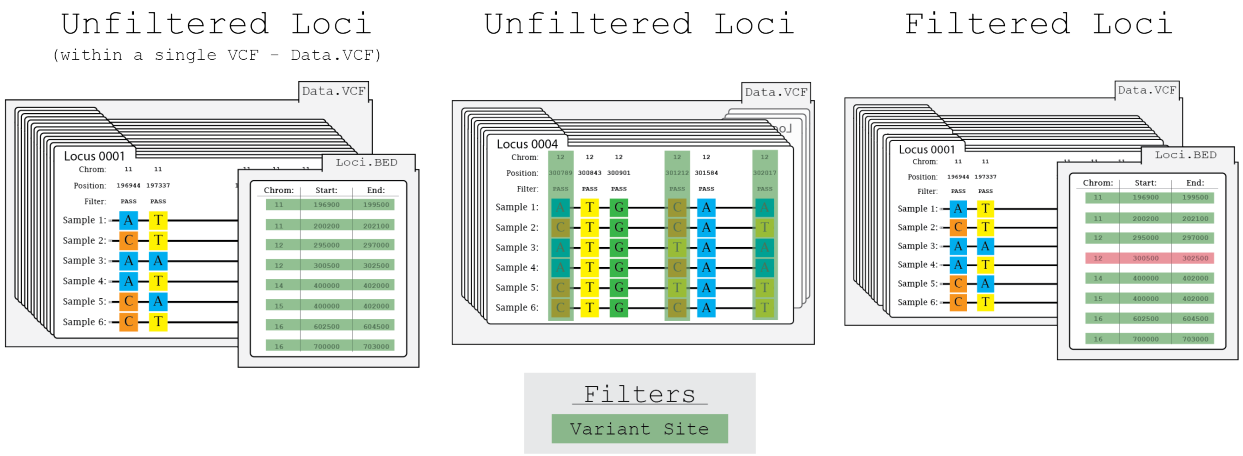
--filter-exclude-positions <*position_filename*> Argument used to define a file of positions to exclude within a tsv file (chromosome and position).

--filter-include-bed <*position_bed_filename*> Argument used to define a BED file of positions to include.

--filter-exclude-bed <*position_bed_filename*> Argument used to define a BED file of positions to exclude.

4.3 informative_loci_filter.py: Informative Loci Filter

This function checks to make sure that a locus has enough sites to be considered informative in either the four-gamete test or an IM run. Given a BED file and a VCF file, informative_loci_filter will find regions in the VCF that have a specified number of variant sites.



In this illustration of the locus filtering process, locus_0004 is removed due to only having three variant sites (highlighted in green) when the threshold is set to four.

Because many variants are not considered useful in these situations, filters are provided for removing sites with missing data, non-biallelic sites, indels, CpGs, and singletons from determining if there are a sufficient number of sites in the region. Output is either a BED file of a set number of random regions that pass the criteria, or a file with all regions that pass. It can also remove regions that are below a minimum specified length. If a model file is specified, only the individuals in the selected population will be considered for singleton and missing data filters.

4.3.1 Input Arguments

--vcf <*vcf_name*> Name of input VCF file

--bed <*bed_file*> Name of input BED file

4.3.2 Output Arguments

--out <*out_name*> Name of output BED file

--randcount <*number_of_regions*> If set, will output set number of regions randomly selected from those that pass the criteria. Default behavior is to output all regions that match criteria.

4.3.3 Filtering Arguments

--remove-multi Do not count tri-allelic+ sites toward number of valid variants in a region

--remove-missing <*max_misscount*> Do not count sites with more than max_misscount missing individuals. Default of -1 indicates all sites are included, 0 indicates sites with any missing data are not counted..

--remove-indels

Do not count indels toward number of valid variants in a region

--parsecpg <*fasta_reference_filename*> Optional argument that if set, will detect whether or not variants are CpGs. A check is made to make sure the positions in the FASTA line up with the correct variant reference allele.

--informative-count <*minimum_allele_count*> Minimum number of haplotypes with both alleles at a site. Default is 2, meaning there must be two of each of reference and alternate allele in the target individuals. Can be set to 1 to filter out invariant sites.

--minsites <*min_sites*> Minimum number of variants required for a region to pass the filtering criteria. Variants that match specified arguments will not be counted towards this total.

--min-length <*min_length*> Minimum base length of region for region to be considered.

4.3.4 Region Arguments

--bed-column-index <*start_idx*>,*<end_idx>*,*<chrom_idx>* Comma-separated string of the zero-based indices of the start, end, and chromosome columns in the input file, so the file doesn't need to be reformatted. Default for a regular BED file is 1,2,0.

--oneidx-start If set, indicates input BED regions are formatted as one-indexed, closed intervals, as opposed to the BED default of zero-based, half-open intervals. For example, the first million bases on a chromosome would be:

Zero-based, half-open: 0,1000000 One-based, closed: 1,1000000

--pad <*pad_length*> If set, regions in input file will be extended by pad_length bases on both sides.

--keep-full-line If set, regions output will be the same line as was present in the input file. Default behavior is to output start/end/chrom columns in that order, without any other data.

4.3.5 Model Arguments

--model-file <*model_filename*> Name of model file that contains individuals to be considered for filtering.

--model <*model_name*> If model file contains more than one model, name of model to be used.

4.3.6 Other Arguments

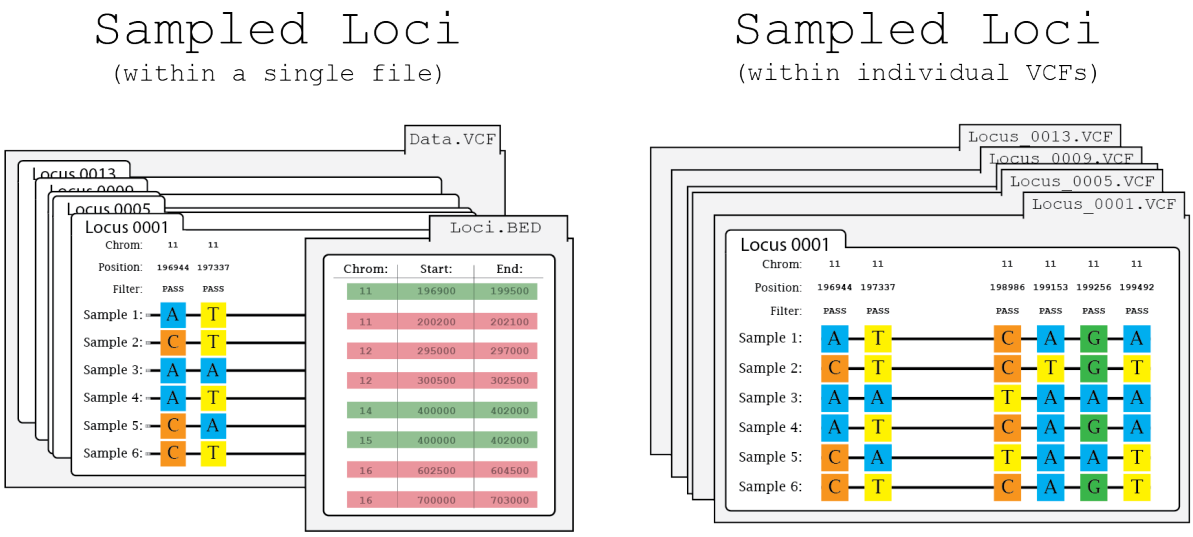
--no-sorting Will output regions in order they were in input file. Default behavior is to sort regions before filtering.

--tbi <*tabix_index*> If input VCF is compressed and tabix file is not default, provide the tabix filename here.

--no-xy Removes regions on X/Y chromosomes from consideration.

4.4 vcf_split.py: VCF Split Function

As a single VCF may include the variant sites of multiple loci, it is often necessary to seperate the loci from the VCF. Given a VCF file and a file of loci (i.e. BED or PPP-created statistic file), vcf_split will generate a VCF for each locus.



In this illustration of the splitting process, Data.VCF includes variant sites associated with a discrete set of loci (i.e. Locus_0001 - Locus_0013). Once split, a single file (e.g. Locus_0001.VCF) will only contain the variant sites associated with that locus.

4.4.1 Command-line Usage

The VCF splitter may be called using the following command:

```
vcf_split.py
```

Example usage

Command-line to split using a statistic file:

```
vcf_split.py --vcf examples/files/merged_chr1_10000.vcf.gz --split-
↪file examples/files/sampled.windowed.weir.fst.tsv --split-method␣
↪statistic-file --model-file examples/files/input.model --model 2Pop
```

4.4.2 Dependencies

- BCFtools

4.4.3 Input Command-line Arguments

--vcf <*input_filename*> Argument used to define the filename of the VCF file to be split.

--split-file <*split_filename*> Argument used to define the file to be split

--model-file <*model_filename*> Argument used to define the model file. Please note that this argument cannot be used with the **--pop-file** argument or individual-based filters.

--model <*model_str*> Argument used to define the model (i.e. the individual(s) to include and/or the populations for relevant statistics). May be used with any statistic. Please note that this argument cannot be used with **--pop-file** argument or the individual-based filters.

4.4.4 Output Command-line Arguments

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--out-format <*vcf, vcf.gz, bcf*> Argument used to define the desired output format. Formats include: uncompressed VCF (vcf); compressed VCF (vcf.gz) [default]; and BCF (bcf).

--out-dir <*output_dir_name*> Argument used to define the output directory.

--overwrite Argument used to define if previous output should be overwritten.

4.4.5 Split Command-line Arguments

--split-method <*statistic-file, bed*> Argument used to define the splitting method. Users may spilit using either a statistic-file (statistic-file) from VCF Calc (or other methods) or a BED file (bed).

--statistic-window-size <*statistic_window_int*> Argument used to define the size of window calculations. This argument is only required if the BIN_END column is absent within the file.

--no-window-correction Argument used to define if a window should not be corrected to avoid an overlap of a single position (i.e. 100-200/200-300 vs. 100-199/200-299).

4.4.6 Filter Command-line Arguments

If using an unfiltered VCF file (e.g. reduce the creation of unnecessary large files) the VCF calculator is able to use either a kept or removed sites/BED file and the individual-based paramemeters.

Individual-Based Arguments

Please note that all individual-based arguments are not compatible with either the **--model** or **--model-file** command-line arguments.

--filter-include-indv <*indv_str*> <*indv1_str, indv2_str, etc.*> Argument used to define the individual(s) to include. This argument may be used multiple times if desired.

--filter-exclude-indv <*indv_str*> <*indv1_str, indv2_str, etc.*> Argument used to define the individual(s) to exclude. This argument may be used multiple times if desired.

--filter-include-indv-file <*indv_filename*> Argument used to define a file of individuals to include.

--filter-exclude-indv-file <*indv_filename*> Argument used to define a file of individuals to exclude.

Position-Based Arguments

--filter-include-positions <*position_filename*> Argument used to define a file of positions to include within a tsv file (chromosome and position).

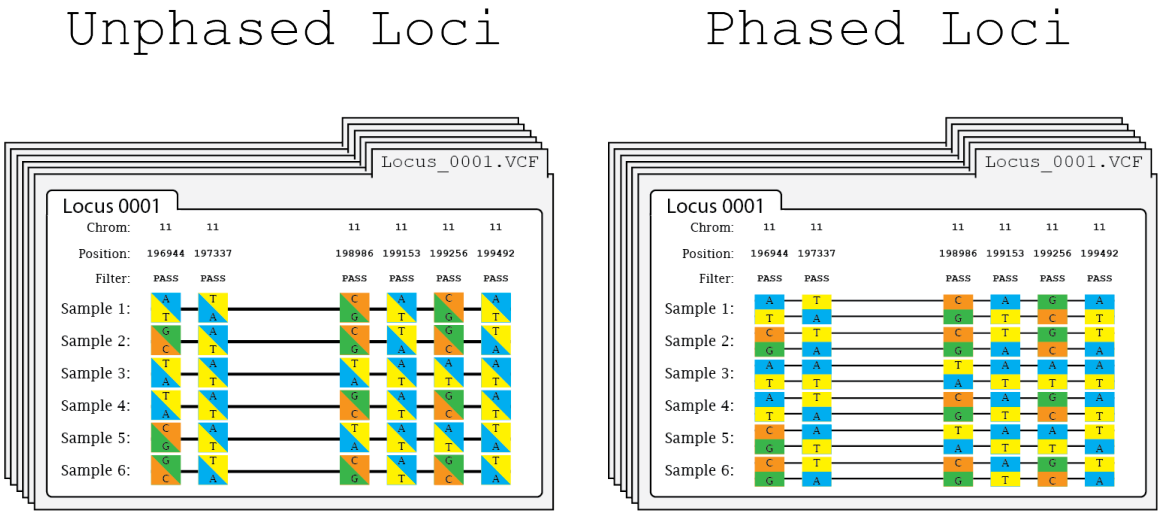
--filter-exclude-positions <*position_filename*> Argument used to define a file of positions to exclude within a tsv file (chromosome and position).

--filter-include-bed <*position_bed_filename*> Argument used to define a BED file of positions to include.

--filter-exclude-bed <*position_bed_filename*> Argument used to define a BED file of positions to exclude.

4.5 vcf_phase.py: VCF Phase Function

Phasing is an essental and frequently used process in population genetic analyses. Given an unphased VCF file and a selected phasing algorithm, vcf_phase will produce a phased VCF. Phasing may be configured using various general options (e.g. specifying Ne, including a genetic map) or algorithm-specific options (e.g. including a compatible reference panel) as needed.



In this illustration of the phasing process, unphased variants (alleles divided diagonally) are converted into an estimated haplotypes (alleles divided horizontally and on seperate strands).

4.5.1 Command-line Usage

The VCF file phaser may be called using the following command:

```
vcf_phase.py
```

Example usage

Command-line to phase a VCF using Beagle:

```
vcf_phase.py --vcf examples/files/merged_chr1_10000.unphased.vcf.gz --
↪phase-algorithm beagle
```

Command-line to phase a VCF using SHAPEIT:

```
vcf_phase.py --vcf examples/files/merged_chr1_10000.unphased.vcf.gz --
↪phase-algorithm shapeit
```

4.5.2 Dependencies

- SHAPEIT
- Beagle 5.0
- BCFtools

- plink 2.0

4.5.3 Input Command-line Arguments

--vcf <*input_filename*> Argument used to define the filename of the VCF file to be phased.

--model-file <*model_filename*> Argument used to define the model file. Please note that this argument cannot be used with the **--pop-file** argument or individual-based filters.

--model <*model_str*> Argument used to define the model (i.e. the individual(s) to include and/or the populations for relevant statistics). May be used with any statistic. Please note that this argument cannot be used with **--pop-file** argument or the individual-based filters.

4.5.4 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**.

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--out-format <*vcf, vcf.gz, bcf*> Argument used to define the desired output format. Formats include: uncompressed VCF (vcf); compressed VCF (vcf.gz) [default]; and BCF (bcf).

--overwrite Argument used to define if previous output should be overwritten.

4.5.5 Phasing Command-line Arguments

--phase-algorithm <*beagle, shapeit*> Argument used to define the phasing algorithm. BEAGLE 5.0 (beagle) [default] and SHAPEIT (shapeit). Please note: Both algorithms possess algorithm-specific arguments that may be found in their respective sections.

--Ne <*Ne_int*> Argument used to define the effective population size.

--genetic-map <*genetic_map_filename*> Argument used to define a genetic map file.

--phase-chr <*chr*> Argument used to define a single chromosome to phase.

--phase-from-bp Argument used to define the lower bound of positions to include. May only be used with a single chromosome.

--phase-to-bp Argument used to define the upper bound of positions to include. May only be used with a single chromosome.

--random-seed <*seed_int*> Argument used to define the seed value for the random number generator.

SHAPEIT Phasing Command-line Arguments

--shapeit-ref <*ref_haps*> <*ref_legend*> <*ref_sample*> Argument used to define a reference panel. Three files are required: the reference haplotypes (.haps), the snp map (.legend), and the individual information (.sample)

--shapeit-burn-iter <*iteration_int*> Argument used to define the number of burn-in iterations.

--shapeit-prune-iter <*iteration_int*> Argument used to define the number of pruning iterations.

--shapeit-main-iter <*iteration_int*> Argument used to define the number of main iterations.

--shapeit-states <*state_int*> Argument used to define the number of conditioning states for haplotype estimation.

--shapeit-window <*Mb_float*> Argument used to define the model window size in Mb.

--shapeit-force Argument used to diable the missing data error (i.e. --force). Use at your own risk.

BEAGLE Phasing Command-line Arguments

--beagle-ref <*ref_vcf, ref_bref3*> Argument used to define a reference panel VCF or bref3.

--beagle-burn-iter <*iteration_int*> Argument used to define the number of burn-in iterations.

--beagle-iter <*iteration_int*> Argument used to define the number of main iterations

--beagle-states <*state_int*> Argument used to define the number of model states for genotype estimation.

--beagle-error <*probability*> Argument used to define the HMM allele mismatch probability.

--beagle-window <*cM_float*> Argument used to define the sliding window size in cM.

--beagle-overlap <*cM_float*> Argument used to define the overlap between neighboring windows in cM.

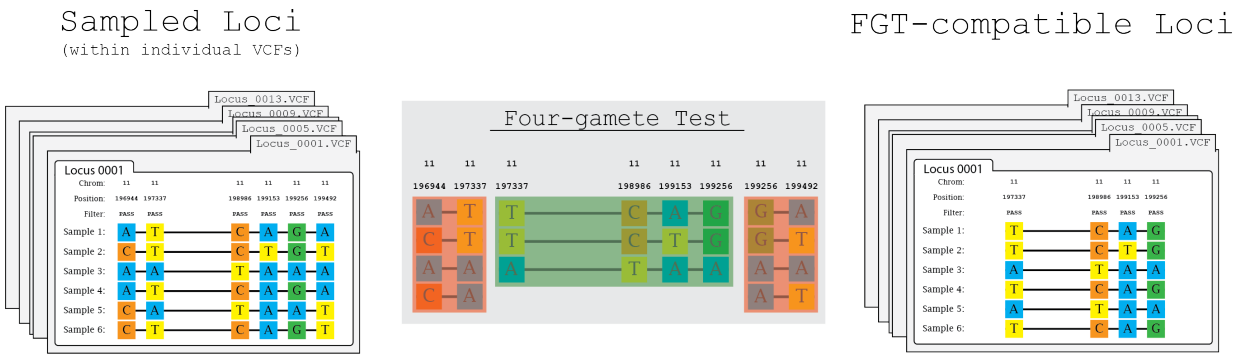
--beagle-step <*cM_float*> Argument used to define the step length in cM used for identifying short IBS segments.

--beagle-nsteps <*windows_int*> Argument used to define the number of consecutive **--beagle-steps** used for identifying long IBS segments.

--beagle-path <*path*> Argument used to define the path to locate beagle.jar.

4.6 vcf_four_gamete.py: Four Gamete Test Function

The four-gamete test is a method for determining whether or not there has been recombination between a pair of variants. To do this, all individuals must have haplotypes defined as the variants at the two sites.



In this illustration of four-gamete test, the haplotypes of the samples from 197337 to 199256 (highlighted in green) pass the four-gamete test. In comparison, the haplotypes from 196944 to 197337 and from 199256 to 199492 (highlighted in red) both fail the four-gamete test as all possible haplotypes are observed.

Given phased input with individual variants over a region of the genome, four_gamete generates an interval within those variants that passes the four-gamete filtering criteria, then return either that interval or an output file with variants in that interval.

Common usage for this function is to input a VCF file that contains variants for individuals at a single locus, with output returned being a VCF that contains a subsample of these variants. A full VCF can be used with --vcfreg, where the second argument is a BED file with one or more regions, output will be either a VCF for four-gamete passing regions or a new BED file with the truncated regions.

4.6.1 Input Arguments

--vcfs <*input_vcf_1*>...*<input_vcf_n>* Input name of one or more VCF files, where each VCF represents a locus.

--vcfreg <*input_vcf*> <*BED file*> Input name of VCF file containing genome data and name of BED file with regions to be analyzed.

4.6.2 Output Aguments

--out <*output_filename*> Name for output file.

--out-prefix <*ouput_prefix*> If multiple files are output, this option is required to set a prefix for the output files.

4.6.3 Interval Arguments

--numinf <*minimum informative site count*> Region returned must have at least n informative sites, defaults to 1

--hk If set, returns intervals with at least one recombination event instead of regions with no recombination.

--reti This script will generate a list of valid regions with no recombination. Selecting this option will return a single interval as specified by other arguments

--retl Returns all valid intervals, either as a list of intervals or multiple output files

4.6.4 Single Returned Region Arguments

Select one of: **--rani**

Returns random interval (default)

--ranb Returns random interval, with probability of interval proportional to interval length

--left Return first interval with enough informative sites

--right Return last interval with enough informative sites

--maxlen Return interval with most informative sites

4.6.5 Other Arguments

--remove-multiallele Removes multi-alleleic sites from analysis

--include-missing Include sites with missing data in analysis

--ovlps Extend region to include non-informative variants between an edge variant and a variant that breaks the four-gamete criteria

--ovlpi Include informative variants from overlapping regions

CHAPTER 5

PPP Input File Generators

For several programs that implement analyses of population genomic variation, PPP provides scripts to generate input files from VCF files.

5.1 vcf_format_conversions.py: VCF to Plink/EIGENSTRAT

Automates various simple file conversions. Currently the function is capable of converting between VCF-based formats (i.e. VCF, compressed-VCF, and BCF) and PLINK-based formats (i.e. PED and Binary-PED). Additional formats will be added as needed.

5.1.1 Input Command-line Arguments

--vcf <*vcf_filename*> Argument used to define the filename of the VCF file.

--vcf-fid <*fid_str*> Argument used to define the family ID for all VCF samples.

--ped-prefix <*ped_prefix*> Argument used to define the filename prefix of both PED and MAP files.

--ped <*ped_filename*> Argument used to define the filename of the PED file. Called alongside **--map**.

--map <*map_filename*> Argument used to define the filename of the MAP file. Called alongside **--ped**.

--binary-ped-prefix <*binary_ped_prefix*> Argument used to define the filename prefix of the Binary-PED, FAM, and BIM files.

--binary-ped <*ped_filename*> Argument used to define the filename of the Binary-PED (i.e. BED) file. Called alongside **--fam** and **--bim**.

--fam <*fam_filename*> Argument used to define the filename of the FAM file. Called alongside **--binary-ped** and **--bim**.

--bim <*bim_filename*> Argument used to define the filename of the BIM file. Called alongside **--binary-ped** and **--fam**.

5.1.2 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**.

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension).

--out-format <*vcf, vcf.gz, bcf, ped, ped-12, binary-ped, eigenstrat*> Argument used to define the desired output format. Formats include: uncompressed VCF (vcf); compressed VCF (vcf.gz); BCF (bcf); PLINK text file (ped); PLINK "12" coded text file (ped-12); binary PLINK file (binary-ped); and eigenstrat file (eigenstrat).

--overwrite Argument used to define if previous output should be overwritten.

5.1.3 Other Command-line Arguments

--delete-original Argument used to define that the original file should be deleted once converted.

--threads <*thread_int*> Argument used to define the number of threads. This argument is currently only supported by conversions to/from PED and Binary-PED.

5.2 vcf_to_ima.py: VCF to IMa Conversion Function

Create IMa input file from four-gamete filtered VCF files.

An IM analysis requires an IM-formatted input file that contains multiple loci, where each locus has sequence information for variant sites as determined by multiple individuals (which are grouped by population). To produce this file, this script takes either a VCF file with a BED file that indicates loci to be used or a VCF file per locus. A model file will be used to split the samples in the VCF(s) into populations.

For each locus, a header line is created that contains several pieces of information, including number of individuals per population at this locus, number of sites in sequences provided, mutation model,

inheritance scalar, and mutation rate per year at locus (over locus, not per base pair). Population sizes and sequence ordering are handled internally, the inheritance scalar and mutation rate can be set via commandline.

Loci provided as input must pass the four-gamete filtering criteria. In addition, if filtering has not been previously done options are available to filter out indels (default), multiallelic sites, and CpGs (with reference genome). Sites with missing data can either be filtered by dropping individuals missing data from a locus from analysis at that locus, or by replacing the missing site with a reference allele.

5.2.1 Input Arguments

--vcf <*input_filename*> Filename for input VCF if using BED file with locus information

--vcfs <*vcf_filename_1*>...*<vcf_filename_n>* One or multiple VCF input filenames where each file contains sequences for a single locus. A file with lines corresponding to filenames can be provided with **--vcfs** @<*vcf_filelist*>

--model-file <*model_filename*> Filename of model file.

--model <*model name*> If model file contains multiple models, use this argument to specify name of population to use.

--reference-fasta <*reference_filename*> Filename for reference FASTA file. File can be uncompressed or bg-zipped, but must be indexed with faidx. When option is specified, default options are to include sequence in output loci but not filter for CpGs (use --parse-cpg)

--bed <*bed_filename*> Filename for BED file specifying loci if only one VCF is provided. Can be used with multiple VCFs if line count aligns, used for getting correct locus length.

5.2.2 Output Arguments

--out <*out_filename*> Output filename.

5.2.3 Model Options

--mutrate <*mutation rate*> Set mutation rate per base pair (default is 1e-9). This value is multiplied by locus length to get mutation rate per locus.

--inheritance-scalar <*scalar*> Sets inheritance scalar for all loci. Default behavior is to set scalar to 1 for non-X/Y/MT chromosomes, .75 for 'X' and 'chrX', and .25 for 'y', 'chrY', 'MT', and 'chrMT'.

5.2.4 Filtering Options

--remove-multiallele Set all multiallelic sites to be reference.

--drop-missing-sites <*individual_count*> Drops all sites where more than 'individual_count' individuals are missing data. Default is -1 (no dropping), and 0 will drop all sites missing data and replace them with the reference allele.

--drop-missing-inds If set, if an individual is missing data at a locus, that individual will not be included at that locus and population counts for that locus will be adjusted.

--remove-cpg Requires --reference-fasta. If set, will replace CpG sites with reference allele at site, setting them as invariant.

5.2.5 Other Options

--oneidx-start If set, indicates input BED regions are formatted as one-indexed, closed intervals, as opposed to the BED default of zero-based, half-open intervals. For example, the first million bases on a chromosome would be:

Zero-based, half-open: 0,1000000 One-based, closed: 1,1000000

--bed-column-index <*start_col,end_col,chrom_col*> Comma-separated list of zero-based indexes of start, end, and chromosome name columns in input BED file. Default value for traditionally structured BED is 1,2,0

--noseq If set, and --reference-fasta is provided, will not output invariant sites to IM file.

5.3 vcf_to_treemix.py: VCF to treemix Conversion Function

The treemix program was developed by Pickrell and Prichard (2012) to estimate phylogeny and admixture for closely related populations.

Pickrell JK, Pritchard JK (2012) Inference of Population Splits and Mixtures from Genome-Wide Allele Frequency Data. PLOS Genetics 8(11): e1002967.

The program can make use of very large numbers of SNPs.

vcf_to_treemix.py will generate a treemix input file from a vcf file.

If run using the --bed-file and --kblock options, the resulting treemix file can be run using the 'linkage disequilibrium' (-k) option. Under this option each block of kblock SNPs are treated as a linked group and different groups are treated as unlinked.

5.3.1 Required Arguments

--vcf <*input_vcf_filename*> The name of the vcf file. This can be a bgzipped vcf file. .

--model-file <*model_file_name*> The name of a PPP model file.

--modelname <*model_name*> The name of a model in the model file. The treemix file to be generated will contain the allele counts for each SNP in each of the populations. The treemix run will estimate the phylogeny for the populations in the model.

--out <*outpuf_file_name*> The name of the treemix file to be generated. The file is bgzipped and '.gz' is added to the end of the name

5.3.2 Optional Aguments

--bed-file <*BED_file_name*> The BED file is a sorted UCSC-style bedfile containing chromosome locations of the SNPs to be included in the output file. The BED file has no header. The first column is the chromosome name (this must match the chromosome name in the vcf file). The second column is start position (0-based, open interval) The third column is end position (closed interval). Any other columns are ignored.

If used with --kblock, each of the BED file regions is used to generate one block of SNPs

--kblock <*k_block_size*> Used with --bed-file, for using treemix runtime option -k. k is the number of SNPs in a block in the treemix file. If the actual number of SNPs in a BED file interval is less than kblock, then additional invariant rows are added to the treemix file so the total numbers of rows for that and every block is equal to kblock. k is set to 1000 by default. It needs to be increased only when one or more BED file regions have more than k snps.

5.3.3 Example usage

Example command-lines:

```
vcf_to_treemix.py -h
```

```
   vcf_to_treemix.py --vcf pan_example.vcf.gz --model-file panmodels.
↪model --modelname 4Pop --out vcf_treemixtest1 --bed-file pan_example_
↪regions.bed --kblock 1000

.. code-block:: bash

   vcf_to_treemix.py --vcf pan_example.vcf.gz --model-file panmodels.
↪model --modelname 4Pop --out vcf_treemixtest2
```

5.4 vcf_to_gphocs.py: VCF to GPhocs Conversion Function

Generates an input sequence file for the G-Phocs program from a vcf file and a fasta reference file.

G-Phocs can estimate the phylogenetic and demographic history of a set of genomes, each sampled at a large number of genomic regions or loci.

Gronau I, Hubisz MJ, Gulko B, Danko CG, Siepel A. Bayesian inference of ancient human demography from individual genome sequences. Nature Genetics 43 1031-1034. 2011

https://github.com/gphocs-dev/G-PhoCS/blob/master/GPhoCS_Manual.pdf

5.4.1 Required Arguments

--vcf <*input_vcf_filename*> The name of the vcf file. This can be a bgzipped vcf file. .

--model-file <*model_file_name*> The name of a PPP model file.

--model <*model_name*> The name of a model in the model file. The treemix file to be generated will contain the allele counts for each SNP in each of the populations. The treemix run will estimate the phylogeny for the populations in the model.

--bed-file <*BED_file_name*> The Bed file specifies the regions of the vcf file to be sampled. Each row of the BED file (region) correspondes to one locus in the G-Phocs sequence file.

The BED file is a sorted UCSC-style bedfile containing chromosome locations of the SNPs to be included in the output files. The BED file has no header. The first column is the chromosome name (this must match the chromosome name in the vcf file). The second column is start position (0-based, open interval) The third column is end position (closed interval). Any other columns are ignored.

--out <*output file name*> Specifies the complete output filename.

--reference <*reference fasta file*> The reference genome fasta file is required in order to generate full sequences from the SNP data in the vcf file.

5.4.2 Optional Aguments

--diploid <*True (default)/False*> By default G-Phocs works with a single sequence for each individual, where heterozygous positions are shown using IUPAC ambiguity codes. If this option is False, then only the first sequence of each individual is returned and heterozygous positions are not shown.

--nloci <*number of loci*> By default the output file will contain as many loci as there are regions in the BED file. With this option, the first nloci regions will be used.

5.4.3 Example usage

Example command-lines:

```
vcf_to_gphocs.py -h
```

```
vcf_to_gphocs.py --vcf pan_example.vcf.gz --reference pan_example_ref.
↪fa --model-file panmodels.model --modelname 4Pop" --bed-file pan_
↪example_regions.bed --outvcf_gphocs_test.out
```

5.5 vcf_to_fastsimcoal.py: VCF to fastsimcoal Conversion Function

Generates Site Frequency Spectrum (SFS) files for fastsimcoal based on instructions in fastsimcoal ver 2.6 manual.

Generates one-dimensional (1D), two-dimensional (2D) and multidimensional SFS files

All generated SFS files are contained in a zip file archive.

Excoffier, L. and M. Foll. 2011. fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios. Bioinformatics 27: 1332-1334.

5.5.1 Required Arguments

--vcf <*input_vcf_filename*> The name of the vcf file. This can be a bgzipped vcf file. .

--model-file <*model_file_name*> The name of a PPP model file.

--modelname <*model_name*> The name of a model in the model file. The treemix file to be generated will contain the allele counts for each SNP in each of the populations. The treemix run will estimate the phylogeny for the populations in the model.

--dim <*dimension file type signifiers*> One or more of '1', '2', or 'm', for 1D, 2D or multidimensional output files.

For 1D files:

- the filename suffix is _DAFpop#.obs for an array of derived allele counts. where '#' is replaced by the population number
- the filename suffix is _MAFpop#.obs for an array of minor allele counts.

For 2D files:

- the filename suffix is _jointDAFpop#_&.obs for an array of derived allele counts. where # and & are population numbers, and # is larger than &

- the filename suffix is _jointDAFpop#_&.obs for an array of minor allele counts.

For a multidimensional file:

- the filename suffix is _DSFS.obs for an array of derived allele counts.
- the filename suffix is _MSFS.obs for an array of minor allele counts.

5.5.2 Optional Aguments

--basename <*name of outpuf file prefix*> This is used to specify the prefix of the output files and the prefix of the zip file archive. The default is "ppp_fsc" in the same folder as the vcf file

--bed-file <*BED_file_name*> The BED file is a sorted UCSC-style bedfile containing chromosome locations of the SNPs to be included in the output files. The BED file has no header. The first column is the chromosome name (this must match the chromosome name in the vcf file). The second column is start position (0-based, open interval) The third column is end position (closed interval). Any other columns are ignored.

--outgroup_fasta <*name of alternative reference sequence*> This option is used to specify the name of a fasta file to use as an alternative reference to that used for the vcf file.

This fasta file must have been properly aligned to the reference used in the vcf file.

This option can be useful, for example, if an ancestral or outgroup reference is available that more accurately identifies the ancestral (and thus derived) allele at each SNP than does the reference used to make the vcf file.

--downsamplesizes <*down sample sizes*> A sequence of integers, one for each of the populations in the model in the same order as populations listed in the model. The values specify the down sampling to be used for each respective population. For a population with k>=1 diploid individuals (2k>=2 genomes) in the model, the downsample count d must be 2<=d<=2k.

--folded <*True/False*> The folded option indicates that the folded sfs should be returned. If folded is False (default) the sfs reports the count of the derived allele. If True, the sfs reports of the count of the minor (less frequent) allele.

--randomsnpprop <*floating point value between 0 and 1*> This option can be used to randomly sample a subset of SNPs. The default is to sample all biallelic SNPs.

--seed <*integer*> This is used with --randomsnpprop as the seed for the random number generator.

5.5.3 Example usage

Example command-lines:

```
vcf_to_fastsimcoal.py -h
```

```
vcf_to_fastsimcoal.py --vcf pan_example2.vcf.gz --model-file panmodels.
↪model --modelname 5Pop --downsamplesizes 3 3 3 4 2  --basename vcf_
↪fsc2 --folded --dim 1 2 m  --outgroup-fasta chr22_pan_example2_ref.fa
```

5.6 vcf_to_dadi.py: VCF to dadi Conversion Function

Generates a dadi snp file from a vcf file.

The dadi snp file format is described in the dadi manual

https://dadi.readthedocs.io/en/latest/user-guide/importing-data/#snp-data-format

Gutenkunst RN, Hernandez RD, Williams SH, Bustamante CD (2009) Inferring the joint demographic history of multiple populations from multidimensional SNP frequency data. PLoS Genet 5: e1000695. DOI: 10.1371/journal.pgen.1000695

5.6.1 Required Arguments

--vcf <*input_vcf_filename*> The name of the vcf file. This can be a bgzipped vcf file. .

--model-file <*model_file_name*> The name of a PPP model file.

--modelname <*model_name*> The name of a model in the model file. The treemix file to be generated will contain the allele counts for each SNP in each of the populations. The treemix run will estimate the phylogeny for the populations in the model.

--out <*output file name*> Specifies the complete output filename.

5.6.2 Optional Aguments

--bed-file <*BED_file_name*> The BED file is a sorted UCSC-style bedfile containing chromosome locations of the SNPs to be included in the output files. The BED file has no header. The first column is the chromosome name (this must match the chromosome name in the vcf file). The second column is start position (0-based, open interval) The third column is end position (closed interval). Any other columns are ignored.

--outgroup_fasta <*name of alternative reference sequence*> This option is used to specify the name of a fasta file to use as an alternative reference to that used for the vcf file.

This fasta file must have been properly aligned to the reference used in the vcf file.

This option can be useful, for example, if an ancestral or outgroup reference is available that more accurately identifies the ancestral (and thus derived) allele at each SNP than does the reference used to make the vcf file.

--comment <*comment text*> Comment text to be included in the header of the output file.

5.6.3 Example usage

Example command-lines:

```
vcf_to_dadi.py -h
```

```
vcf_to_dadi.py --vcf  pan_example.vcf.gz --model-file  panmodels.model␣
↪ --modelname 4Pop  --out vcf_dadisnp_bedfile_test.out  --comment␣
↪testing bedfile --bed-file pan_example_regions.bed
```

```
vcf_to_dadi.py --vcf  pan_example2.vcf.gz --model-file  panmodels.
↪model  --modelname 4Pop --out  vcf_dadisnp_test.out  --comment␣
↪testing comment
```

```
vcf_to_dadi.py --vcf pan_example2.vcf.gz --model-file panmodels.model  -
↪-modelname 4Pop --out vcf_dadisnp_fasta_test.out  --comment testing␣
↪outgroup-fasta --outgroup-fasta  chr22_pan_example2_ref.fa
```

CHAPTER 6

PPP Analyses

These functions were developed to perform the actual population genetic analyses for the PPP. Please note that each functions was designed to use a specific file format and the use of a conversion function may be required. See each function for more details.

6.1 eigenstrat_fstats.py: F-statistics Analysis

Automates the calculation of multiple admixture statistics, including: Patterson's D, F4 statistic, F4-ratio statistic, and F3 statistic.

6.1.1 Command-line Usage

The admixture statistics automater may be called using the following command:

```
eigenstrat_fstats.py
```

Example usage

Command-line to calculate Patterson's D:

```
eigenstrat_fstats.py --eigenstrat-prefix snps --calc-admix-statistic D -
↪-admix-w-pop French --admix-x-pop Yoruba --admix-y-pop Vindija --
↪admix-z-pop Chimp
```

Command-line to calculate the F4-ratio:

```
eigenstrat_fstats.py --eigenstrat-prefix snps --calc-admix-statistic F4-
↪ratio --admix-a-pop Altai --admix-b-pop Vindija --admix-c-pop Yoruba -
↪-admix-x-pop French --admix-o-pop Chimp
```

6.1.2 Dependencies

- AdmixTools
- admixr

6.1.3 Input Command-line Arguments

--eigenstrat-prefix <*input_prefix*> Argument used to define the filename prefix shared by the genotype file (.geno), the individual file (.ind), and the SNP file (.snp). Should not be used alongside the specific file arguments (e.g. --geno).

--geno <*geno_filename*> Argument used to define the filename of the eigenstrat genotype file (.geno). Must be called alongside --ind and --snp. Cannot be called alongside --eigenstrat-prefix.

--ind <*ind_filename*> Argument used to define the filename of the eigenstrat individual file (.ind). Must be called alongside --geno and --snp. Cannot be called alongside --eigenstrat-prefix.

--snp <*snp_filename*> Argument used to define the filename of the eigenstrat SNP file (.snp). Must be called alongside --geno and --ind. Cannot be called alongside --eigenstrat-prefix.

--model-file <*model_filename*> Argument used to define the model file. Please note that this argument cannot be used with the individual-based filters.

--model <*model_str*> Argument used to define the model (i.e. the individual(s) to include and/or the populations for relevant statistics). May be used with any statistic. Please note that this argument cannot be used with **--pop-file** argument or the individual-based filters.

6.1.4 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**. Cannot be used if multiple output files are created.

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--overwrite Argument used to define if previous output should be overwritten.

6.1.5 Statistic Command-line Specification

--calc-admix-statistic <*D, F4, F4-ratio, F3*> Argument used to define the admix statistic to be calculated. Patterson's D (D), F4 statistic (F4), F4-ratio statistic (F4-ratio), and F3 statistic (F3). See below for details on the arguments requried by each statistic .

Statistic Command-line Requirements

It should be noted that each admix statistic has a specific set of population labels arguments. These labels are used to specify a representive population. For instance, the argument '--admix-w-pop CEU' will replace the W label of Patterson's D and the F4 statistic with the CEU population. These arguments may be found in the next section.

--calc-admix-statistic *D* Requires: **--admix-w-pop/--admix-w-pop-file**, **--admix-x-pop/--admix-x-pop-file**, **--admix-y-pop/--admix-y-pop-file**, and **--admix-z-pop/--admix-z-pop-file**.

--calc-admix-statistic *F4* Requires: **--admix-w-pop/--admix-w-pop-file**, **--admix-x-pop/--admix-x-pop-file**, **--admix-y-pop/--admix-y-pop-file**, and **--admix-z-pop/--admix-z-pop-file**.

--calc-admix-statistic *F4-ratio* Requires: **--admix-a-pop/--admix-a-pop-file**, **--admix-b-pop/--admix-b-pop-file**, **--admix-c-pop/--admix-c-pop-file**, **--admix-x-pop/--admix-x-pop-file**, and **--admix-o-pop/--admix-o-pop-file**.

--calc-admix-statistic *F3* Requires: **--admix-a-pop/--admix-a-pop-file**, **--admix-b-pop/--admix-b-pop-file**, and **--admix-c-pop/--admix-c-pop-file**.

Additional Statistic Command-line Arguments

--admix-w-pop <*w_pop_str*> <*w_pop1_str, w_pop2_str, etc.*> Argument used to define the population(s) to represent W in the supported admixure statistic. This argument may be used multiple times if desired. If multiple populations the statistic will be repeated until each population has represented W.

--admix-w-pop-file <*w_pop_filename*> Argument used to define a file of population(s) to represent W in the supported admixure statistic. If multiple populations the statistic will be repeated until each population has represented W.

--admix-x-pop <*x_pop_str*> <*x_pop1_str, x_pop2_str, etc.*> Argument used to define the population(s) to represent X in the supported admixure statistic. This argument may be used multiple times if desired. If multiple populations the statistic will be repeated until each population has represented X.

--admix-x-pop-file <*x_pop_filename*> Argument used to define a file of population(s) to represent X in the supported admixure statistic. If multiple populations the statistic will be repeated until each population has represented X.

--admix-y-pop <*y_pop_str*> <*y_pop1_str, y_pop2_str, etc.*> Argument used to define the population(s) to represent Y in the supported admixure statistic. This argument may be used multiple times if desired. If multiple populations the statistic will be repeated until each population has represented Y.

--admix-y-pop-file <*y_pop_filename*> Argument used to define a file of population(s) to represent Y in the supported admixure statistic. If multiple populations the statistic will be repeated until each population has represented Y.

--admix-z-pop <*z_pop_str*> <*z_pop1_str, z_pop2_str, etc.*> Argument used to define the population(s) to represent Z in the supported admixure statistic. This argument may be used multiple times if desired. If multiple populations the statistic will be repeated until each population has represented Z.

--admix-z-pop-file <*z_pop_filename*> Argument used to define a file of population(s) to represent Z in the supported admixure statistic. If multiple populations the statistic will be repeated until each population has represented Z.

--admix-a-pop <*a_pop_str*> <*a_pop1_str, a_pop2_str, etc.*> Argument used to define the population(s) to represent A in the supported admixure statistic. This argument may be used multiple times if desired. If multiple populations the statistic will be repeated until each population has represented A.

--admix-a-pop-file <*a_pop_filename*> Argument used to define a file of population(s) to represent A in the supported admixure statistic. If multiple populations the statistic will be repeated until each population has represented A.

--admix-b-pop <*b_pop_str*> <*b_pop1_str, b_pop2_str, etc.*> Argument used to define the population(s) to represent B in the supported admixure statistic. This argument may be used multiple times if desired. If multiple populations the statistic will be repeated until each population has represented B.

--admix-b-pop-file <*b_pop_filename*> Argument used to define a file of population(s) to represent B in the supported admixure statistic. If multiple populations the statistic will be repeated until each population has represented B.

--admix-c-pop <*c_pop_str*> <*c_pop1_str, c_pop2_str, etc.*> Argument used to define the population(s) to represent C in the supported admixure statistic. This argument may be used multiple times if desired. If multiple populations the statistic will be repeated until each population has represented C.

--admix-c-pop-file <*c_pop_filename*> Argument used to define a file of population(s) to represent C in the supported admixure statistic. If multiple populations the statistic will be repeated until each population has represented C.

6.2 admixture.py: Admixture Analysis

Automates the estimation of individual ancestries using Admixture. The functions allows for input as: i) Binary-PED files or ii) PED 12-formatted files. The function is also capable of configuring the optional arguments of Admixture.

6.2.1 Command-line Usage

The admixture automater may be called using the following command:

```
admixture.py
```

Example usage

Estimating individual ancestries for each sample within *hapmap3.bed* for three ancestral populations.

```
admixture.py --binary-ped-prefix hapmap3 --pop 3
```

6.2.2 Dependencies

- Admixture

6.2.3 Input Command-line Arguments

--ped-12-prefix <*input_prefix*> Argument used to define the filename prefix shared by the 12-formatted ped file (.ped) and the map file (.map). Should not be used alongside the specific file arguments (e.g. --ped).

--ped-12 <*ped_filename*> Argument used to define the filename of the plink 12-formatted ped file (.ped). Must be called alongside --map. Cannot be called alongside --ped-prefix.

--map <*map_filename*> Argument used to define the filename of the plink map file (.map). Must be called alongside --ped. Cannot be called alongside --ped-prefix.

--binary-ped-prefix <*input_prefix*> Argument used to define the filename prefix shared by the binary ped file (.bed), the fam file (.fam), and the bim file (.bim). Should not be used alongside the specific file arguments (e.g. --binary-ped).

--binary-ped <*binary_ped_filename*> Argument used to define the filename of the plink binary ped file (.bed). Must be called alongside --fam and --bim. Cannot be called alongside --binary-ped-prefix.

--fam <*fam_filename*> Argument used to define the filename of the plink fam file (.fam). Must be called alongside --binary-ped and --bim. Cannot be called alongside --binary-ped-prefix.

--bim <*bim_filename*> Argument used to define the filename of the plink bim file (.bim). Must be called alongside --binary-ped and --fam. Cannot be called alongside --binary-ped-prefix.

6.2.4 Output Command-line Arguments

--overwrite Argument used to define if previous output should be overwritten.

6.2.5 Required Command-line Arguments

--pop <*K_int*> Argument used to defines the number of ancestral populations.

--admix-method <*em, block*> Argument used to define the algorithm to use. Two algorithm are supported: Block relaxation algorithm (block) or EM algorithm (em). By default, the Block relaxation algorithm is used.

6.2.6 Optional Command-line Arguments

--acceleration <*acceleration_int*> Argument used to defines the value of quasi-Newton acceleration method.

--major-converge-likelihood <*likelihood_float*> Argument used to define the major terminaton criterion. Halt when the log-likelihood increases by less than the specified value between iterations.

--major-converge-iter <*iter_int*> Argument used to define the major terminaton criterion. Defines the maximum number of iterations.

--minor-converge-likelihood <*likelihood_float*> Argument used to define the minor terminaton criterion. Halt when the log-likelihood increases by less than the specified value between iterations.

--minor-converge-iter <*iter_int*> Argument used to define the minor terminaton criterion. Defines the maximum number of iterations.

--bootstrap <*bootstrap_int*> Argument used to define the number of bootstrap replicates.

--random-seed <*seed_int*> Argument used to define the seed value for the random number generator.

--threads <*thread_int*> Argument used to define the number of threads to be used for computation.

6.3 ima3_wrapper.py: IMa3 Analysis

Wrapper for IMa3 executable. This wrapper will handle locating the proper version of IMa3 for the desired run, as well as providing multi-threading support if the system has the mpirun utility and the IMa executable was compiled with it. Most options for IMa3 are described in the IMa3 manual, and it is HIGHLY recommended that users overview this document before calling IMa3. Note that wrapper functions (--threads,--ima-path) require two dashes, while IMa arguments use only one.

6.3.1 Input Arguments

-i <*input_file*> Name of IMa3 input file generated by PPP

-o <*output_file*> Name of IMa3 output file. Additional files will use this as prefix.

6.3.2 Parameter Arguments

-q <*max_pop_size*> Sets maximum population size parameter for all populations.

-m <*migration_rate*> Sets migration rate prior.

-t <*max_split_time*> Sets maximum splitting time parameter.

6.3.3 Wrapper Arguments

--threads <*thread_count*> Set number of threads to use. This will check that the proper version of IMa3 has been compiled and the system has mpirun installed.

--ima-path <*path_to_ima*> Path to IMa executable to use if not on system path. This should include the name of the executable, not just the path to it.

6.4 plink_linkage_disequilibrium.py: Linkage Disequilibrium Analysis

Automates the calculation of multiple LD statistics using Plink.

6.4.1 Command-line Usage

The LD statistics automater may be called using the following command:

```
plink_ld.py
```

Example usage

Command-line to calculate Lewontin's D-prime statistic

```
plink_ld.py --ped-prefix hapmap1 --ld-format table --ld-statistic r2 --
↪table-d-statistic dprime
```

6.4.2 Dependencies

- plink 1.9
- *plink 2.0 <https://www.cog-genomics.org/plink/2.0/>*

6.4.3 Input Command-line Arguments

--ped-prefix <*input_prefix*> Argument used to define the filename prefix shared by the ped file (.ped) and the map file (.map). Should not be used alongside the specific file arguments (e.g. --ped).

--ped <*ped_filename*> Argument used to define the filename of the plink ped file (.ped). Must be called alongside --map. Cannot be called alongside --ped-prefix.

--map <*map_filename*> Argument used to define the filename of the plink map file (.map). Must be called alongside --ped. Cannot be called alongside --ped-prefix.

--binary-ped-prefix <*input_prefix*> Argument used to define the filename prefix shared by the binary ped file (.bed), the fam file (.fam), and the bim file (.bim). Should not be used alongside the specific file arguments (e.g. --binary-ped).

--binary-ped <*binary_ped_filename*> Argument used to define the filename of the plink binary ped file (.bed). Must be called alongside --fam and --bim. Cannot be called alongside --binary-ped-prefix.

--fam <*fam_filename*> Argument used to define the filename of the plink fam file (.fam). Must be called alongside --binary-ped and --bim. Cannot be called alongside --binary-ped-prefix.

--bim <*bim_filename*> Argument used to define the filename of the plink bim file (.bim). Must be called alongside --binary-ped and --fam. Cannot be called alongside --binary-ped-prefix.

--allow-extra-chr Argument used to force invalid chromosome names to be accepted.

6.4.4 Output Command-line Arguments

--out-format <*output_format*> Argument used to define the output format. Supported formats include: gzip compressed (gzipped); standard uncompresed (standard); single-precision binary (bin32); and double-precision binary (bin64). Please note that both binary formats are only supported when called with the square **--lf-format**. By default gzip compressed files are produced.

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**. Cannot be used if multiple output files are created.

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--overwrite Argument used to define if previous output should be overwritten.

6.4.5 Basic LD Command-line Arguments

--ld-statistic <*r, r2*> Argument used to define the correlation statistic to report. Two options are supported: the raw inter-variant allele count correlations (r) and squared correlations (r2).

--ld-format <*table, square, square-zero, triangle, inter-chr*> Argument used to define the matrix result format. Five formats are supported: The matrix as a limited window in table format (table); A symmetric matrix (square); a square matrix in which the cells of the upper right triangle are zeroed out (square-zero); only the lower-triangular of the matrix (triangle); the matrix with all pairs in a table (inter-chr). **--ld-window-snps** <*snp_int*>

Argument used to define the maximum number of SNPs between LD comparisons.

--ld-window-kb <*snp_int*> Argument used to define the maximum distance in bp between LD comparisons.

--ld-window-cm <*snp_int*> Argument used to define the maximum distance in cM between LD comparisons.

Table Command-line Arguments

Please note that the following arguments may only be used with **--ld-format** table.

--table-d-statistic <*dprime, dprime-signed, d*> Argument used to add the specified D statistic to table-formatted results. Three options are supported: the absolute value of Lewontin's D-prime statistic (dprime); Lewontin's D-prime statistic (dprime-signed); and the value of D prior to division by D_{max} (d). **--table-in-phase**

Argument used to add in-phase allele pairs to table-formatted results.

--table-maf Argument used to add MAF values to table-formatted results.

--table-r2-threshold <*r2_float*> Argument used to define the threshold for filtering pairs of r2 values.

--table-snp <*snp_str*> <*snp1_str, snp2_str, etc.*> Argument used to define one or more SNP(s) for LD analysis. This argument may be used

multiple times if desired.

--table-snps <*snp_filename*> Argument used to define a file with one or more SNP(s) for LD analysis.

6.5 vcf_to_sfs.py: Site Frequency Spectrum generator

For generating the site frequency spectrum (sfs)for a population model from a vcf file.

The sfs is an array with as many dimensions as populations in the model. For example, if population samples are in order A,B, C then position (i,j,k) of the array refers to the count of SNPs with derived alleles that were observed to have a count of i in A, j in B, and k in C

If the sfs is folded then the count in a cell of the sfs is the number of SNPs with that combination of minor allele counts.

This script can be run in stand-alone mode, or for more flexibility it can be imported to give access to more general functions for building and manipulating the sfs.

All vcf handling assumes that all individuals are diploid at all SNPs.

6.5.1 Required Arguments

--vcf <*input_vcf_filename*> The name of the vcf file. This can be a bgzipped vcf file. .

--model-file <*model_file_name*> The name of a PPP model file.

--modelname <*model_name*> The name of a model in the model file. The treemix file to be generated will contain the allele counts for each SNP in each of the populations. The treemix run will estimate the phylogeny for the populations in the model.

--out <*out file name*> The name of an output file. If --out is omitted the default is ppp_sfs.out in the same folder as the vcffile This will be a tab-delimited file If the number of dimensions is 2, the sfs is contained in the rows and columns, otherwise the values are given on the first line of the file

6.5.2 Optional Aguments

--bed-file <*BED_file_name*> The BED file is a sorted UCSC-style bedfile containing chromosome locations of the SNPs to be included in the output files. The BED file has no header. The

first column is the chromosome name (this must match the chromosome name in the vcf file). The second column is start position (0-based, open interval) The third column is end position (closed interval). Any other columns are ignored.

--outgroup_fasta <*name of alternative reference sequence*> This option is used to specify the name of a fasta file to use as an alternative reference to that originally used for the vcf file.

This fasta file must have been properly aligned to the reference used in the vcf file.

This option can be useful, for example, if an ancestral or outgroup reference is available that more accurately identifies the ancestral (and thus derived) allele at each SNP than does the reference used to make the vcf file.

--downsamplesizes <*down sample sizes*> A sequence of integers, one for each of the populations in the model in the same order as populations listed in the model. The values specify the down sampling to be used for each respective population. For a population with k>=1 diploid individuals (2k>=2 genomes) in the model, the downsample count d must be 2<=d<=2k.

--folded <*True/False*> The folded option indicates that the folded sfs should be returned. If folded is False (default) the sfs reports the count of the derived allele. If True, the sfs reports of the count of the minor (less frequent) allele.

--randomsnpprop <*floating point value between 0 and 1*> This option can be used to randomly sample a subset of SNPs. The default is to sample all biallelic SNPs.

--seed <*integer*> This is used with --randomsnpprop as the seed for the random number generator.

--makeint <*True/False*> If True, round the counts in the sfs to the nearest integer (default False)

6.5.3 Example usage

Example command-lines:

```
vcf_to_sfs.py -h
```

```
vcf_to_sfs.py --vcf pan_example2.vcf.gz --model-file panmodels.model --
↪modelname 4Pop --downsamplesizes 3 3 3 4 --folded --outgroup-fasta ␣
↪chr22_pan_example2_ref.fa --out vcf_to_sfs_test1.txt
```

6.5.4 Importing Functions

This file has two functions that can be useful for working with site frequency spectra

build_sfs()

The default script that runs when this file is run. This function can also be accessed directly by importing this file.

- generates an sfs from a vcf file
- can handle an arbitrary number of dimensions (populations) and sample sizes, so long as each population has at least a sample size of two genomes (i.e. one diploid individual)
- handles downsampling, and reduction of dimensions
- handles unfolded and folded sfs's
- can take a BED file name to sample portions of a vcf file
- can handle an alternative reference genome for rooting, rather than that used in the vcf file
- the arguments closely resemble those used when the function is called by running this file

Required Arguments

The first three arguments are, in order, the vcf filename, the model file name, and the name of the model. These are each strings.

Optional arguments

Each optional argument requires the use of the argument name.

BEDfilename The name of a ucsc-style bedfile with intervals to include

altreference The name of a fasta sequence file that contains the reference genome

folded True/False. To indicate that the folded sfs should be returned. (False is default)

downsamplesizes A list of sample sizes to be used if they are less than given in the model 2 <= downsamplesizes[i] <= samplesizes[i]

randomsnpprop The proportion of snps to include using random sampling

seed A random number seed that can be used with randomsnpprop.

makeint Causes the array to be rounded to the nearest integer (dtype remains float)

out The name of a file to contain the sfs if out is not None, this will write a tab-delimited file of the array

Example usage

Example python code:

```
import vcf_to_sfs as vs
mysfs = vs.build_sfs(pan_example2.vcf.gz,panmodels.model,'4Pop',
        folded=True,downsamplesizes=[3,3,3,4],
        altreference='chr22_pan_example2_ref.fa',
        out = 'mysfsfile.txt')
```

reduce_sfs_dims()

This function is for reducing the dimensionality of an sfs by summing across axes. It is accessed by importing this file.

There are three required arguments in order:

- the sfs (i.e. a numpy array with as many dimensions as populations)
- an instance of Class model that specifies the populations and samples to which the sfs corresponds.
- a list of names of the populations to keep in the reduced sfs

There is one optional argument, 'out', if the reduced sfs is to be written to a file (e.g. out=mysfs.txt)

Example usage

Example python code:

```
import vcf_to_sfs as vs
myreducedsfs = vs.reduce_sfs_dim(mysfs,mypopmodel,
        ['A','B','C'],out="myreducedsfs_file.txt")
```

CHAPTER 7

PPP Utilities

The utility functions were developed to perform various tasks often needed when preparing files for population genetic analyses.

7.1 vcf_utilities.py: VCF Utilities

Automates various utilites for VCF-formatted files. This currently includes: obtain a list of the chromosomes within a VCF-based file, obtain a list of the samples within a VCF-based file, concatenate multiple VCF-based files, merge multiple VCF-based files, and sort a VCF-based file.

7.1.1 Command-line Usage

The VCF utilites function may be called using the following command:

```
python vcf_utilites.py
```

Example usage

Concatenate multiple VCF files:

```
python vcf_utilites.py --vcfs chr21.vcf.gz chr22.vcf.gz --utility␣
↪concatenate
```

Merge multiple VCF files:

```
python vcf_utilites.py --vcfs chr22.ceu.vcf.gz chr22.yri.vcf.gz --
↪utility merge
```

7.1.2 Dependencies

- BCFtools

7.1.3 Input Command-line Arguments

--vcf <*input_filename*> Argument used to define the filename of the VCF file.

--vcfs <*input_filename*> <*input1_filename, input2_filename, etc.*> Argument used to define the filename of the VCF file(s). May be used multiple times.

7.1.4 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**.

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--overwrite Argument used to define if previous output should be overwritten.

7.1.5 Utility Command-line Specification

--utility <*sample-list, chr-list, concatenate, merge, sort*> Argument used to define the desired utility. Current utilities include: creation of a file of the samples within the VCF (sample-list); creation of a file of the chromosomes within the VCF (chr-list); combine multiple VCF files with different variants but the same samples (concatenate); combine multiple VCF files with different samples but the same variants (merge); or sort a single VCF file (sort).

Additional Utility Command-line Arguments

--record-merge-mode <*none, snps, indels, both, all, id*> Argument used to define the type of multiallelic records to create. Only usable with the merge utility.

--record-missing-as-ref Argument used to define that missing records should be converted to the reference allele. Only usable with the merge and concatenate utilites.

--out-format <*vcf, vcf.gz, bcf*> Argument used to define the desired output format. Formats include: uncompressed VCF (vcf); compressed VCF (vcf.gz) [default]; and BCF (bcf). Only usable with the merge and concatenate utilites.

7.2 bed_utilities.py: BED Utilites

Automates various utilites for BED-formatted files. This currently includes: i) sample a BED file; ii) subtract from a BED that overlap with a second BED file; iii) extend a BED upstream, downstream, or both upstream and downstream; iv) sort a single BED; v) merge features within one or more BED files; vi) create a BED of complementary features.

7.2.1 Command-line Usage

The BED utilites function may be called using the following command:

```
bed_utilities.py
```

7.2.2 Utilites

Windows Utility

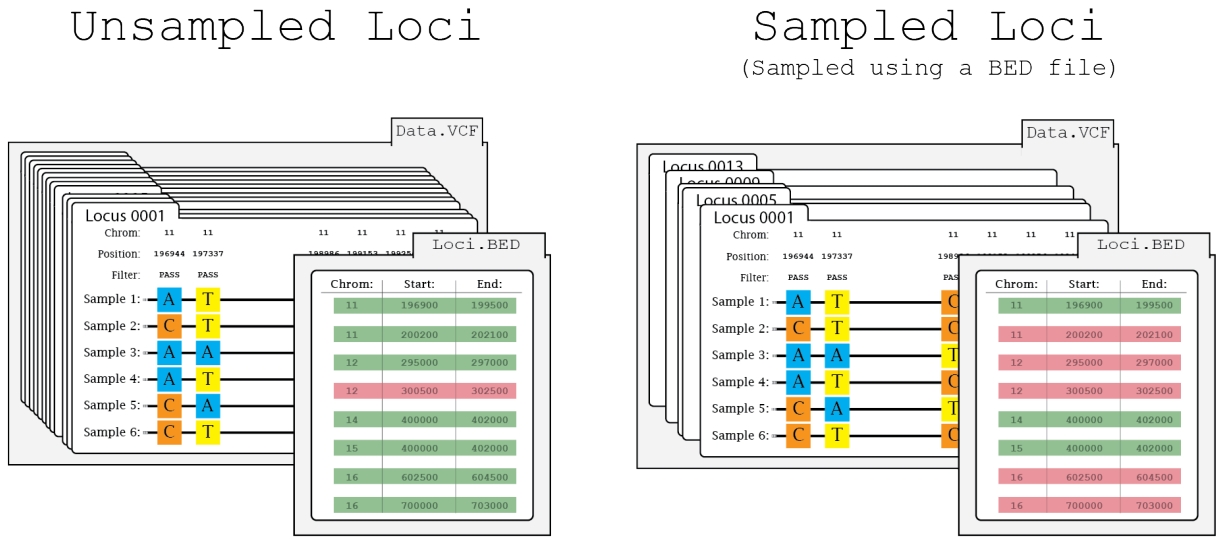
Given a chromosome size file and a window size, the windows utility will generate a BED file of interval features.

Example usage

Return a BED with interval features that do not extend outside the chromosomes:

```
bed_utilities.py --utility windows --chrom-file hg18.chrom.sizes --
↪window-size 1000 --out hg18_windows.bed
```

Sample Utility



Given a BED file and a sample size, the sample utility will generate a pseudorandomly sampled BED. Please note that the random seed may be used to reproduced the sample.

Example usage

Sample 20 features from a BED file:

```
bed_utilities.py --utility sample --bed examples/files/chr1_sites.bed --
↪sample-size 20
```

Sort Utility

Given an unsorted BED file, the sort utility will generate a sorted BED file.

Example usage

Sort an unsorted BED file:

```
bed_utilities.py --utility sort --bed examples/files/chr1_sites.
↪unsorted.bed
```

Extend Utility

Given a BED file and an extend length, the extend utility will increase the length of each feature upstream, downstream, or both upstream and downstream.

Example usage

Extend upstream by 1kb:

```
bed_utilities.py --utility extend --bed examples/files/chr1_sites.bed --
↪chrom-file examples/files/chr_sizes.txt --extend-upstream 1000
```

Extend downstream by 1kb:

```
bed_utilities.py --utility extend --bed examples/files/chr1_sites.bed --
↪chrom-file examples/files/chr_sizes.txt --extend-downstream 1000
```

Extend flanks (i.e. both upstream and downstream) by 1kb:

```
bed_utilities.py --utility extend --bed examples/files/chr1_sites.bed --
↪chrom-file examples/files/chr_sizes.txt--extend-flanks 1000
```

Subtract Utility

Given two BED files, the subtract utility will remove BED features from a BED file if they overlap with the features from a second BED file.

Example usage

Remove BED features if they overlap features within the subtract BED file:

```
bed_utilities.py --utility subtract --bed examples/files/chr1_sites.
↪bed --subtract-bed examples/files/chr1_sites.1.bed --subtract-entire-
↪feature
```

Complement Utility

Given a BED file, the complementary utility will generate a BED file of complementary features.

Example usage

Return a BED with features that do not overlap within the given file:

```
bed_utilities.py --utility complement --bed examples/files/chr1_sites.
↪bed --chrom-file examples/files/chr_sizes.txt
```

Intersect Utility

Given a BED file and an intersect file, return only the interval features within the BED file that overlap with the intersect file.

Example usage

Return a BED with only intersecting interval features:

```
bed_utilities.py --utility intersect --bed hg18_windows.bed --intersect-
↪file Intersect.vcf.gz --out hg18_intersects.bed
```

Merge Utility

Given one or more BED files, the merge utility will generate a single sorted BED file of merged BED features.

Example usage

Merge BED features from a single BED file:

```
bed_utilities.py --utility merge --bed examples/files/chr1_sites.bed
```

Merge BED features from multiple BED files:

```
bed_utilities.py --utility merge --beds examples/files/chr1_sites.1.
↪bed examples/files/chr1_sites.2.bed examples/files/chr1_sites.3.bed␣
↪examples/files/chr1_sites.4.bed
```

7.2.3 Dependencies

- BEDtools

7.2.4 Input Command-line Arguments

--bed <*input_filename*> Argument used to define the filename of the BED file.

--beds <*input_filename*> <*input1_filename, input2_filename, etc.*> Argument used to define the filename of the BED file(s). May be used multiple times.

--chrom-file <*chrom_filename*> Argument used to define the filename of a file with the sizes of each chromosome. Chromosome size files must be tab-delimited as follows:

```
chr1        247249719
chr2        242951149
...
chrX        154913754
chrY        57772954
```

Appropriate files may be downloaded from the UCSC Genome Browser. The supported **ASSEMBLY.chrom.sizes** file for each assembly may be found by clicking **Genome sequence files and select annotations** (followed by **Standard genome sequence files and select annotations** on select assemblies).

7.2.5 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename.

--overwrite Argument used to define if previous output should be overwritten.

7.2.6 Utility Command-line Specification

--utility <*sample, subtract, extend, sort, merge, complement*> Argument used to define the desired utility. Current utilities include: sample features from a BED file (sample); subtract features from a BED file that overlap with features within a second BED file (subtract); extend the flanks of features upstream, downstream, or both within a single BED file (extend); sort the features within a single BED file (sort); merge features within one or more BED files (merge); create a BED file of complementary features - i.e. features that do not overlap - from a BED file (complement).

Window Utility Command-line Arguments

--window-size <*window_size_int*> Argument used to define the window/interval size to return.

Sample Utility Command-line Arguments

--sample-size <*sample_size_int*> Argument used to define the total sample size.

--random-seed <*seed_int*> Argument used to define the seed value for the random number generator.

Subtract Utility Command-line Arguments

--subtract-bed <*subtract_file_filename*> Argument used to define the BED file used for removing features/positions.

--subtract-entire-feature Argument used to define if entire features within the input BED should be removed if they overlap with features in subtract-bed.

--min-reciprocal-overlap <*overlap_float*> Argument used to define the minimum reciprocal overlap of features required for removal (e.g. 0.1 indicates 10% overlap).

--min-input-overlap <*overlap_float*> Argument used to define the minimum overlap of input features required for removal.

--min-subtract-overlap <*overlap_float*> Argument used to define the minimum overlap of subtract-bed features required for removal.

--subtract-entire-feature Argument used to define that features should be removed from the input BED if the minimum overlap of **--min-input-overlap** or **--min-subtract-overlap** is reached.

Extend Utility Command-line Arguments

--extend-flanks <*bp_int*> Argument used to define the length of base pairs (bp) to extend both upstream and downstream of features.

--extend-upstream <*bp_int*> Argument used to define the length of base pairs (bp) to extend upstream of features.

--extend-downstream <*bp_int*> Argument used to define the length of base pairs (bp) to extend downstream of features.

Intersect Utility Command-line Arguments

--intersect-file <*intersect_file_filename*> Argument used to define the BED/VCF/VCF.gz file used to remove features that do not intersect with the given file's features/variants. removing features/positions.

Merge Utility Command-line Arguments

--max-merge-distance <*bp_int*> Argument used to define the maximum distance allowed between features to be merged.

7.3 vcf_bed_to_seq.py: Generate sequences from VCF/BED Files

For generating a file with reconstituted sequences for regions of a vcf file.

Given a vcf file, a corresponding fasta reference file, a population model, and a specific interval, this script will return a file containing reconstituted sequences - two for each diploid individual.

This script can be run in stand-alone mode, or for more flexibility it can be imported to give access to more general functions for generating reconstituted sequences.

7.3.1 Required Arguments

--vcf <*input_vcf_filename*> The name of the vcf file. This can be a bgzipped vcf file. .

--model-file <*model_file_name*> The name of a PPP model file.

--modelname <*model_name*> The name of a model in the model file.

--fasta-reference <*reference fasta file*> The reference genome fasta file is required in order to generate full sequences from the SNP data in the vcf file.

--region <*region string*> The region of the reference to return sequences from. Format: chromosome name,colon (:),first base number,dash(-),last base number. e.g. chr1:100392-101391

--out <*out file name*>

- the name of an output file. If --out is omitted the default is ppp_vcf_to_sequences.out

7.3.2 Optional Aguments

--return-single <*True/False*> If true, only a single sequence is returned for each individual in the model. The sequence for a given individual uses the first allele given for that individual for each SNP in the vcf file.

7.3.3 Example usage

Example command-lines:

```
vcf_bed_to_seq.py -h
```

```
vcf_bed_to_seq.py --vcf pan_example.vcf.gz --fasta-reference pan_
↪example_ref.fa --model-file panmodels.model --modelname 4Pop --
↪region 21:4431001-4499000  --out vcf_bed_to_seq_test.out
```

7.3.4 Importing Functions

This file has two functions that can be useful for getting lists of reconstituted sequences from a vcf file and a fasta reference file.

get_model_sequences_from_region()

Returns a list of sequences for a region in a vcf file and a samtools/pysam style region string.

Required Arguments

Each argument requires the use of the argument name.

vcf Either a vcf_reader (i.e. see vf.VcfReader()) or the name of a vcf file (can be bgzipped)

popmodel An instance of Class model. The individuals in the model must also be in the vcf file

seq_reference A string that either contains the DNA sequence for the region or is a string containing the name of a fasta file. If a fasta file name, the chromosome name(s) in the fasta file must match those in the vcf file

region Either a samtools/pysam style region string ("chromosome name:start-end") where the chromosome name matches that used in the vcf file where start and end are the 1-based endpoints (closed interval) Or an instance of class Region (Regions uses 0-based open interval on the left)

return_single If True, return only the first sequence for an individual else, return two sequences (default False)

Optional arguments

Each optional argument requires the use of the argument name.

return_single If True, return only the first sequence for an individual else, return two sequences (default False)

out The name of a file to which the sequences are to be written (or appended if the file exists).

Example usage

Example python code

```
import vcf_bed_to_seq as vbs
myregion,sequences = vbs.get_model_sequences_from_region(vcf="pan_
↪example.vcf.gz",
          popmodel=mymodel,seq_reference="pan_example_ref.fa",
          region="21:4431001-4499000",return_single=False,
          out = "chr21regionsequences.out")
```

get_model_sequences()

Returns a generator for getting sets of sequences from regions given in a BED file for individuals in a model. Each call to next() returns a list of sequences for the next region in the BED file

Arguments

Each argument requires the use of the argument name. All arguments are required with the exception taht either BED_filename or region_string (but not both) must be used.

vcf the name of the vcf file (can be bgzipped)

model_file name of a model file

modelname The name of a model in model_file Either both model_file and modelname must be used, or popmodel must be used

popmodel An instance of class model. The individuals in the model must also be in the vcf file Either popmmodel or both model_file and modelname must be used

fasta_reference A fasta file with one more sequences corresponding to the vcf file typically these are reference chromosome sequences the chromosome name(s) in the fasta file must match those in the vcf file

BED_filename A sorted BED file giving regions from which to pull sequences the first column with the chromosome name must match a name in the fasta_reference file The chromosome names in the BED file must match those in the fasta file and the vcf file

region_string A pysam style region string, if only one region is to be returned

Example usage

Example python code:

```
a = get_model_sequences(vcf="pan_example.vcf.gz",
            popmodel=mymodel,fasta_reference="pan_example_ref.fa",
```

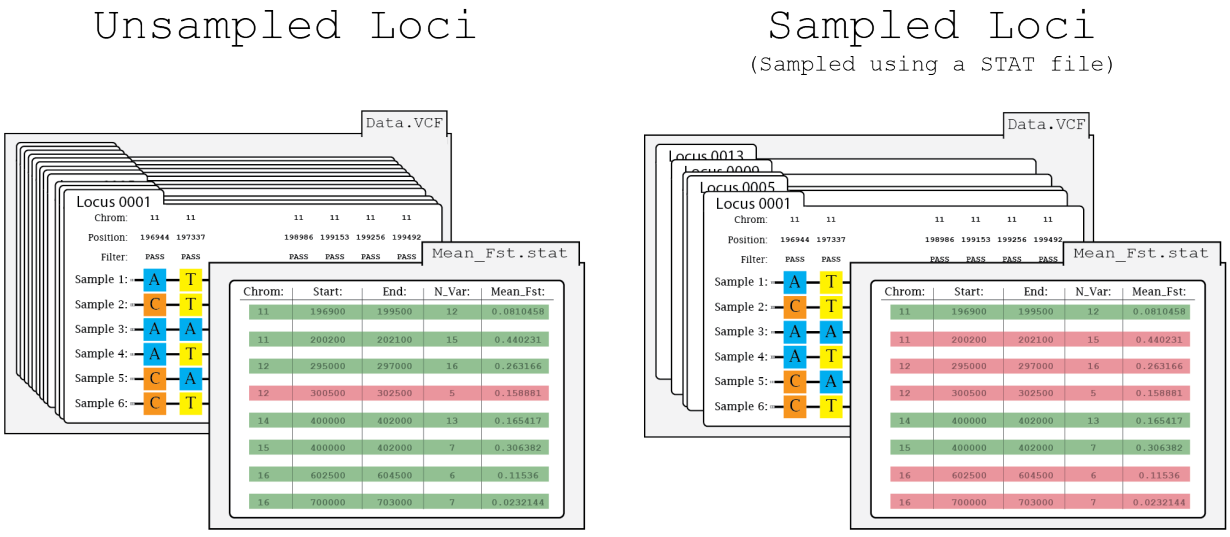
(continues on next page)

(continued from previous page)

```
            BED_filename = "pan_example_regions.bed")
while True:
    try:
        s = next(a)# Will raise StopIteration if lines is exhausted
        print(len(s),len(s[0]))
    except StopIteration:
        break    # end loop
```

7.4 stat_sampler.py: STAT File Sampler

As a single statistic file may include far more loci/windows than a technique is capable of analyzing, it is often necessary to sample the loci/windows from the file. Given a statistic file and a sampling scheme, stat_sampler will generate a pseudorandomly sampled file.



In this illustration of the sampling process, the loci found within Data.VCF are pseudorandomly sampled using the corrdinates found within the given statistic file.

Two pseudorandomly sampling schemes are provided: i) a random sampler that will randomly select loci/windows and ii) a uniform sampler that will evenly sample across equal-sized bins of the given statistic. Please note that all sampling is done without replacement.

For BED-based sampling, please see ../Utilities/bed_utilities.rst.

7.4.1 Command-line Usage

The statistic sampler may be called using the following command:

```
stat_sampler.py
```

Example usage

Randomly sampling 20 windows from a windowed Fst statistic file **merged_chr1_10000.windowed.weir.fst**.

```
stat_sampler.py --statistic-file examples/files/merged_chr1_10000.
↪windowed.weir.fst --calc-statistic windowed-weir-fst --sampling-
↪scheme random --sample-size 20
```

Uniform sampling 20 windows from four bins from a windowed pi statistic file **merged_chr1_10000.windowed.pi**.

```
stat_sampler.py --statistic-file examples/files/merged_chr1_10000.
↪windowed.pi --calc-statistic window-pi --sampling-scheme uniform --
↪uniform-bins 4 --sample-size 20
```

7.4.2 Input Command-line Arguments

--statistic-file <*statistic_filename*> Argument used to define the filename of the statistic file for sampling.

7.4.3 Output Command-line Arguments

--out <*output_filename*> Argument used to define the complete output filename, overrides **--out-prefix**. Cannot be used if multiple output files are created.

--out-prefix <*output_prefix*> Argument used to define the output prefix (i.e. filename without file extension)

--overwrite Argument used to define if previous output should be overwritten.

7.4.4 Sampling Command-line Arguments

--calc-statistic <*windowed-weir-fst, TajimaD, window-pi*> Argument used to define the statistic to be sampled. Windowed Fst (windowed-weir-fst), Tajima's D (TajimaD), and windowed nucleotide diversity (window-pi).

--sampling-scheme <*random, uniform*> Argument used to define the sampling scheme. Random [Default] sampling or uniform sampling across of number of equal-sized bins.

--uniform-bins <*bin_int*> Argument used to define the number of bins in uniform sampling.

--sample-size <*sample_size_int*> Argument used to define the total sample size. If using the uniform sampling scheme, this number must be divisible by the number of bins.

--random-seed <*seed_int*> Argument used to define the seed value for the random number generator.

CHAPTER 8

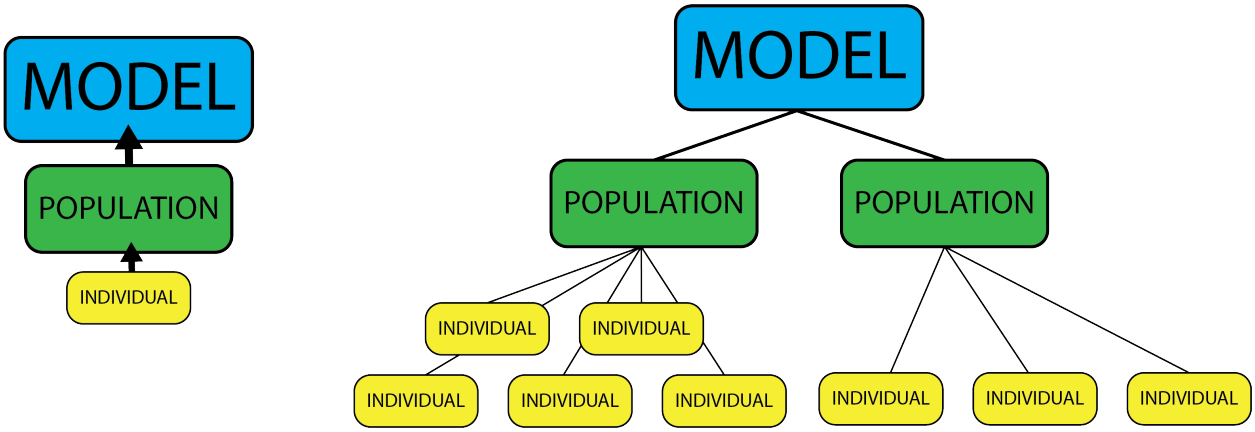
Model File and Creation

A core aspect of the PPP is the use of Model files, JSON-based files used to assign and store **population models**. A population model primarily consists of: the populations within the model; the individuals in each population; and a population tree. Model files offer various benefits within the PPP: i) automatic assignment of relevant populations, individuals, or other potential meta-data; ii) simplified process to examine multiple models; and iii) a single repository of all relevant meta-data.

Model files may be created and edited using our model creator.

8.1 model_creator.py: Model File Creator

Many PPP functions were designed to automatically assign relevant populations and/or individuals using a *Model* file. To enable this functionality, the model_creator.py function may be used to produce *Model* files by either: i) manually entering the necessary information or ii) by using files with the relevant information. It is also all possible to create multiple models simultaneously and assign populations to more than a single model.



A simple way to visualize models are as a hierarchy. Each *Model* may contain one or more *Populations* and each *Population* may contain one or more *Individuals*.

8.1.1 Command-line Usage

The model creator may be called using the following command:

```
model_creator.py
```

Example 1: Simple Model

A basic model only require a single population (pop) with a single in individual (ind). Only three commands are required:

1. Create and name a model: *--model 1Pop*
2. Assign a pop to a model: *--model-pop 1Pop Paniscus*
3. Assign an ind to a pop: *--pop-ind Paniscus Pan_paniscus-9731_LB502*

```
model_creator.py --model 1Pop --model-pop 1Pop Paniscus --pop-ind␣
↪Paniscus Pan_paniscus-9731_LB502
```

Example 2: Model Using Files

A model may also be created using two file options:

1. Assign multiple pops to model: *--model-pop-file 2Pop 2Pops.txt*
2. Assign multiple inds to pop: *--pop-ind-file Paniscus Paniscus.txt*

```
model_creator.py --model 2Pop --model-pop-file 2Pop examples/files/
↪2Pops.txt --pop-ind-file Paniscus examples/files/Paniscus.txt --pop-
↪ind-file Troglodytes examples/files/Troglodytes.txt
```

(continues on next page)

(continued from previous page)

Listing 1: examples/files/2Pops.txt

```
Paniscus
Troglodytes
```

Listing 2: examples/files/Paniscus.txt

```
Pan_paniscus-9731_LB502
Pan_paniscus-A915_Kosana
```

Listing 3: examples/files/Troglodytes.txt

```
Pan_troglodytes_troglodytes-A957_Vaillant
Pan_troglodytes_troglodytes-A958_Doris
```

Example 3: Update Model in Model File

A model may be updated if desired using the following options:

1. Assign the model file: *--model-file input.model*
2. Assign the model to update: *--update-model 2Pop*
3. Assign a pop to remove from a model: *--model-rm-pop 2Pop Troglodytes*
4. Assign a new or previously created pop to a model: *--model-pop 2Pop Schweinfurthii*
5. Assign multiple inds to pop (if new): *--pop-ind-file Schweinfurthii Schweinfurthii.txt*

```
model_creator.py --model-file examples/files/input.model --update-
↪model 2Pop --model-pop 2Pop Schweinfurthii --model-rm-pop 2Pop␣
↪Troglodytes
```

Example 4: Update Population in Model File

A population may be updated if desired using the following options:

1. Assign the model file: *--model-file input.model*
2. Assign the pop to update: *--update-pop Paniscus*
3. Assign a ind to add to a pop: *--pop-ind Paniscus Pan_paniscus-Unknown*
4. Assign a ind to remove from a pop: *--pop-rm-ind Paniscus Pan_paniscus-9731_LB502*

```
model_creator.py --model-file examples/files/input.model --update-pop␣
↪Paniscus --pop-ind Paniscus Pan_paniscus-Unknown --pop-rm-ind␣
↪Paniscus Pan_paniscus-9731_LB502
```

8.1.2 Standard Model Command-line Arguments

Except for **--model-file** all other model-based arguments may be used multiple times.

--model-file <*str*> Argument used to define the name of a model file.

--model <*model_str*> Argument used to define the name of a model to create.

--update-model <*str*> Argument used to define the name of a model to update. Allows for: i) tree update and ii) populations to be added.

--update-pop <*str*> Argument used to define the name of a population to update. Allows for individuals to be added.

--model-tree <*model_str*> <*newick_str*> Argument used to assign a population tree to a model, in Newick format.

--model-tree-file <*model_str*> <*newick_file*> Argument used to assign a population tree file to a model, in Newick format.

--model-pop <*model_str*> <*pop_str*> Argument used to assign a population to a model.

--model-pops <*model_str*> <*pop1_str*> <*pop2_str*> .. Argument used to assign a multiple populations to a model.

--model-pop-file <*model_str*> <*pop_file*> Argument used to assign a multiple populations to a model using a file.

--pop-ind <*pop_str*> <*ind_str*> Argument used to assign a individual to a population.

--pop-inds <*pop_str*> <*ind1_str*> <*ind2_str*> .. Argument used to assign a multiple individuals to a population.

--pop-ind-file <*pop_str*> <*ind_file*> Argument used to assign a multiple individuals to a population using a file.

8.1.3 Model Update: Compatible Command-line Arguments

Please note: **--update-model** is required to update a model.

--update-model <*str*> Argument used to define the name of a model to update. Allows for: i) tree update and ii) populations to be added and/or removed.

--model-file <*str*> Argument used to define the name of a model file.

--model-tree <*model_str*> <*newick_str*> Argument may be used to replace a population tree, in Newick format.

--model-tree-file <*model_str*> <*newick_file*> Argument may be used to replace a population usng a tree file, in Newick format.

--model-pop <*model_str*> <*pop_str*> Argument used to assign a population to the model begin updated.

--model-pops <*model_str*> <*pop1_str*> <*pop2_str*> .. Argument used to assign multiple populations to the model begin updated.

--model-pop-file <*model_str*> <*pop_file*> Argument used to assign multiple populations to the model begin updated using a file.

--model-rm-pop <*model_str*> <*pop_str*> Argument used to remove a population to the model begin updated.

--model-rm-pops <*model_str*> <*pop1_str*> <*pop2_str*> .. Argument used to remove multiple populations to the model begin updated.

--model-rm-pop-file <*model_str*> <*pop_file*> Argument used to remove multiple populations to the model begin updated using a file.

8.1.4 Population Update: Compatible Command-line Arguments

Please note: **--update-pop** is required to update a population.

--update-pop <*str*> Argument used to define the name of a population to update. Allows for individuals to be added.

--model-file <*str*> Argument used to define the name of a model file.

--pop-ind <*pop_str*> <*ind_str*> Argument used to assign a individual to the population begin updated.

--pop-inds <*pop_str*> <*ind1_str*> <*ind2_str*> .. Argument used to assign multiple individuals to the population begin updated.

--pop-ind-file <*pop_str*> <*ind_file*> Argument used to assign multiple individuals to thr population begin updated using a file.

--pop-rm-ind <*pop_str*> <*ind_str*> Argument used to remove a individual to the population begin updated.

--pop-rm-inds <*pop_str*> <*ind1_str*> <*ind2_str*> .. Argument used to remove multiple individuals to the population begin updated.

--pop-rm-ind-file <*pop_str*> <*ind_file*> Argument used to remove multiple individuals to thr population begin updated using a file.

8.1.5 Output Command-line Arguments

--out *<output_filename>* Argument used to define the complete output filename.

--overwrite Argument used to define if previous output should be overwritten.

An example Model file may be seen below:

```
[
{
    "name": "2Pop",
    "tree": "(Troglodytes,Verus);",
    "pops": {
        "Verus": {
            "inds": [
                "Pan_troglodytes_verus-9668_Bosco",
                "Pan_troglodytes_verus-9730_Donald",
                "Pan_troglodytes_verus-A956_Jimmie",
                "Pan_troglodytes_verus-Clint",
                "Pan_troglodytes_verus-X00100_Koby"
            ]
        },
        "Troglodytes": {
            "inds": [
                "Pan_troglodytes_troglodytes-A957_Vaillant",
                "Pan_troglodytes_troglodytes-A958_Doris",
                "Pan_troglodytes_troglodytes-A959_Julie",
                "Pan_troglodytes_troglodytes-A960_Clara"
            ]
        }
    }
}
]
```

CHAPTER 9

Development

As it would be unlikely that the PPP would encompass all desired procedures/methods, we has included this section to aid in the development of additional functionality for the PPP.

9.1 Development Guidelines

In general:

- Functions developed for the PPP should be modular – i.e. function independently – primarily to maintain a flexible platform, this is especially important when considering Galaxy integration.
- Functions should be able to be imported from **pgpipe**, so that multiple functions may be used within a single python script or Jupyter notebook.
- Functions should support the use of Model files for automatic assignment of relevant metadata. *Note: details on the model file class may be found below*
- Functions that use third-party software should include the relevant reference(s) within log files.

9.2 Using PPP Classes

9.2.1 VCF Class

italic **BOLD** outside link: Google. internal link: examples

```
my_code.py
```

The **VcfReader()** class is responsible for integrating VCF files into the PPP. It can parse unzipped and bgzipped VCFs (as well as BCFs), allowing for the SNP records of those files to be accessed in multiple ways, such as sequentially or from a particular region (as implemented in the **GeneRegion** class). It can also subsample requested individuals from a VCF using the **ModelFile** functionality or by providing a list of individuals to sample. The class can be initialized as follows:

There are two primary operating modes for this class: operating on bgzipped or unzipped data. For unzipped data, there is a requirement that records are fetched from the file sequentially, so selecting records using gene regions requires those regions be sorted (as they are by default). For a bgzipped file, a tabix index must be created in order to access any records.

To perform an action on different regions specified in a BED file:

To subsample the VCF for given individuals:

9.2.2 ModelFile Class

Model files may be read using the function **read_model_file()** found within the model.py module. The reader accepts the filename (as a string) of the model file and returns a **ModelFile** class object as shown below:

```
# Read in the model file
model_file = pgpipe.model.read_model_file("path/to/my_model_file.
↪model")
```

A **ModelFile** class object primarily behaves as a dictionary of **Model** class objects; the keys of the dictionary are the model names (as strings) of the **Model** class objects whereas the values are the objects themselves. Therefore, a **Model** named *2Pop* may assigned as shown below:

```
# Assign the 2Pop model, from model file
model_2pop =  model_file['2Pop']
```

A **ModelFile** class object also has two additional attributes:

- ind_file: If created, the filename of a file containing all the unique individuals found within all models stored within the **ModelFile**.

- exclude_file: If created, the filename of a file containing all the unique individuals found within all models stored within the **ModelFile** that do not match a given list of individuals.

These attributes may be populated and their files created using the following functions: **create_ind_file()** and **create_exclude_ind_file()**. The inverse operating can also be completed using **delete_ind_file()** and **delete_exclude_ind_file()**.

9.2.3 Model Class

The **Model** class is used to store model meta-data. The primary attributes of the class are:

- name: The name (as a string) of the model
- tree: The newick tree of the model
- pop_list: A list of the population names within the model
- ind_dict: A dictionary used to store the individuals within each population; the keys of the dictionary are the population names whereas the values are lists of individuals.
- nind: A dictionary used to store the number of individuals within each population; the keys of the dictionary are the population names whereas the values are individual counts.
- npop: The number of populations within the model
- inds: A list of all individuals in the model

A Model may be created with all primary attributes populated as shown below:

```
# Create the model
model = pgpipe.model.Model("2Pop")

# Assign the model tree
model.assign_tree("(A,B);")

# Assign the populations and their individuals
model.assign_pop("A", ["Ind1", "Ind2", "Ind3"])
model.assign_pop("B", ["Ind4", "Ind5", "Ind6"])
```

A **Model** class object also has two additional attributes:

- pop_files: If created, a list of population filenames. Each population file consist of the individuals found within the population.
- ind_file: If created, the filename of a file containing all the unique individuals found within the model.

These attributes may be populated and their files created using the following functions: **create_ind_file()** and **create_pop_files()**. The inverse operating can also be completed using **delete_ind_file()** and **delete_pop_files()**.

Lastly, a **Model** class object masy be assigned to a **ModelFile** class object as shown below:

```
# Create ModelFile object
models = pgpipe.model.ModelFile()

# Save the model
models[str(model.name)] = model
```

CHAPTER 10

Contact Us

To report bugs or request help, please visit the PPP Github.

CHAPTER 11

Citations

The PPP combines a number of software packages and analyses. Therefore, any packages and/or analyses used in a PPP analysis should be cited alongside the PPP.

To aid in this endeavour, the log file of each operation will include relevant citations.

Python Module Index

Index