PyPop API Reference

Developer documentation

Release 1.3.1

Alexander K. Lancaster

Oct 22, 2025

Contents

1 Package introduction 1

2 Submodules 1
- PyPop.Arlequin 1
- PyPop.CommandLineInterface 3
- PyPop.DataTypes 4
- PyPop.Filter 6
- PyPop.GUIApp 8
- PyPop.Haplo 10
- PyPop.HardyWeinberg 12
- PyPop.Homozygosity 14
- PyPop.Main 15
- PyPop.Meta 16
- PyPop.ParseFile 17
- PyPop.RandomBinning 19
- PyPop.Utils 19
- PyPop.citation 23
- PyPop.popmeta 23
- PyPop.pypop 23
- PyPop.xslt 24

3 Attributes 24

4 Functions 24

5 Package Contents 24

6 GNU Free Documentation License 25

Python Module Index 28

Index 30

Documenting API for release* 1.3.1 *of PyPop.

Document revision: 1.3.1.post22+gbc51f10d7

This API reference guide for PyPop is automatically generated from the 1.3.1 source code via sphinx-autoapi[1].

References to the *User Guide* can be found in the *PyPop User Guide*: HTML[2]| PDF[3].

1 Package introduction

2 Submodules

PyPop.Arlequin

Module for exposing Arlequin functionality in Python.

Attributes

```
usage_message
```

Classes

ArlequinWrapper	New wrapper for Arlequin
ArlequinExactHWTest	Wraps the Arlequin Hardy-Weinberg exact functionality
ArlequinBatch	A Python `wrapper' class for Arlequin.

Module Contents

class ArlequinWrapper(*matrix=None*, *arlequinPrefix='arl_run'*, *arlequinExec='arlecore.exe'*, *untypedAllele='****'*, *arpFilename='output.arp'*, *arsFilename='arl_run.ars'*, *debug=None*)

New wrapper for Arlequin

outputArpFile(*group*)

outputArsFile(*arsFilename*, *arsContents*)

Outputs the run-time Arlequin program file.

outputRunFiles()

Generates the expected '.txt' set-up files for Arlequin.

runArlequin()

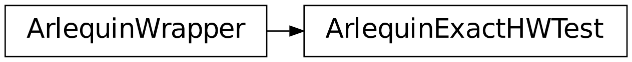
Run the Arlequin haplotyping program.

Forks a copy of 'arlecore.exe', which must be on 'PATH' to actually generate the desired statistics estimates from the generated '.arp' file.

cleanup()

class ArlequinExactHWTest(*matrix=None*, *lociList=None*, *markovChainStepsHW=100000*, *markovChainDememorisationStepsHW=1000*, ***kw*)

Bases: *ArlequinWrapper*



Wraps the Arlequin Hardy-Weinberg exact functionality

Setup run HW exact test.

Run Hardy-Weinberg exact test on list specified in 'lociList'.

- 'markovChainStepsHW': Number of steps to use in Markov chain (default: 100000).
- 'markovChainDememorisationStepsHW': "Burn-in" time for Markov chain (default: 1000).

getHWExactTest()

Returns a dictionary of loci.

Each dictionary element contains a tuple of the results from the Arlequin implementation of the Hardy-Weinberg exact test, namely:

- number of genotypes,
- observed heterozygosity,
- expected heterozygosity,
- the p-value,
- the standard deviation,
- number of steps,

If locus is monomorphic, the HW exact test can't be run, and the contents of the dictionary element simply contains the string 'monomorphic', rather than the tuple of values.

class ArlequinBatch(*arpFilename*, *arsFilename*, *idCol*, *prefixCols*, *suffixCols*, *windowSize*, *mapOrder=None*, *untypedAllele='0'*, *arlequinPrefix='arl_run'*, *debug=0*)

A Python `wrapper' class for Arlequin.

Given a delimited text file of multi-locus genotype data: provides methods to output Arlequin format data files and runtime info and execution of Arlequin itself.

Is used to provide a `batch' (command line) mode for generating appropriate Arlequin input files and for forking Arlequin itself.

Constructor for HaploArlequin object.

Expects:

- arpFilename: Arlequin filename (must have '.arp' file extension)
- arsFilename: Arlequin settings filename (must have '.ars' file extension)
- idCol: column in input file that contains the individual id.
- prefixCols: number of columns to ignore before allele data starts
- suffixCols: number of columns to ignore after allele data stops
- windowSize: size of sliding window
- mapOrder: list order of columns if different to column order in file (defaults to order in file)
- untypedAllele: (defaults to '0')
- arlequinPrefix: prefix for all Arlequin run-time files (defaults to 'arl_run').
- debug: (defaults to 0)

`outputArlequin`(*data*)

Outputs the specified .arp sample file.

`outputRunFiles`()

Generates the expected '.txt' set-up files for Arlequin.

`runArlequin`()

Run the Arlequin haplotyping program.

Forks a copy of 'arlecore.exe', which must be on 'PATH' to actually generate the desired statistics estimates from the generated '.arp' file.

`usage_message = Multiline-String`

```
"""Usage: Arlequin.py [OPTION] INPUTFILE ARPFILE ARSFILE
Process a tab-delimited INPUTFILE of alleles to produce an data files
(including ARPFILE), using parameters from ARSFILE for the Arlequin population
genetics program.

 -i, --idcol=NUM       column number of identifier (first column is zero)
 -l, --ignorelines=NUM number of header lines to ignore in in file
 -c, --cols=POS1,POS2  number of leading columns (POS1) before start and
                        number of trailing columns before the end (POS2) of
                        allele data (including IDCOL)
 -k, --sort=POS1,..    specify order of loci if different from column order
                        in file (must not repeat a locus)
 -w, --windowsize=NUM  number of loci involved in window size
                        (note that this is half the number of allele columns)
 -u, --untyped=STR     the string that represents `untyped' alleles
                        (defaults to '****')
 -x, --execute         execute the Arlequin program
 -h, --help            this message
 -d, --debug           switch on debugging

  INPUTFILE   input text file
  ARPFILE     output Arlequin '.arp' project file
  ARSFILE     input Arlequin '.ars' settings file"""
```

PyPop.CommandLineInterface

Classes

PyPopFormatter	Help message formatter which adds default values to argument help.
CitationAction	Information about how to convert command line strings to Python objects.

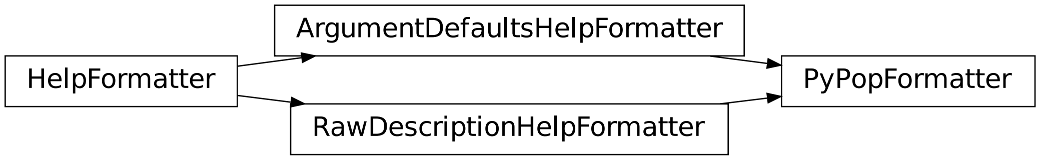
Functions

get_parent_cli([version, copyright_message])
get_pypop_cli([version, copyright_message])
get_popmeta_cli([version, copyright_message])

Module Contents

class PyPopFormatter(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Bases: `argparse.ArgumentDefaultsHelpFormatter`[4], `argparse.RawDescriptionHelpFormatter`[5]

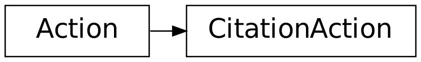


Help message formatter which adds default values to argument help.

Only the name of this class is considered a public API. All the methods provided by the class are considered an implementation detail.

class CitationAction(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `argparse.Action`[6]



Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments

- **which** (*- option_strings -- A list of command-line option strings*) – should be associated with this action.
- **object** (*- dest -- The name of the attribute to hold the created*)
- **be** (*- nargs -- The number of command-line arguments that should*) – consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- **the** (*- metavar -- The name to be used for the option's argument with*) – option uses an action that takes no values.
- **specified.** (*- default -- The value to be produced if the option is not*)
- **and** (*- type -- A callable that accepts a single string argument,*) – returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- **None,** (*- choices -- A container of values that should be allowed. If not*) – after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
- **the** – command line. This is only meaningful for optional command-line arguments.
- **argument.** (*- help -- The help string describing the*)
- **the** – help string. If None, the 'dest' value will be used as the name.

get_parent_cli(*version=''*, *copyright_message=''*)

get_pypop_cli(*version=''*, *copyright_message=''*)

get_popmeta_cli(*version=''*, *copyright_message=''*)

PyPop.DataTypes

Module for storing genotype and allele count data.

Classes

Genotypes	Base class that stores and caches basic genotype statistics.
AlleleCounts	WARNING: this class is now obsolete, the Genotypes class

Functions

checkIfSequenceData(matrix)	
getMetaLocus(locus, isSequenceData)	
getLocusPairs(matrix, sequenceData)	Returns a list of all pairs of loci from a given StringMatrix
getLumpedDataLevels(genotypeData, locus, lumpLevels)	Returns a dictionary of tuples with alleleCount and locusData

Module Contents

class **Genotypes**(*matrix=None*, *untypedAllele='****'*, *unsequencedSite=None*, *allowSemiTyped=0*, *debug=0*)

Base class that stores and caches basic genotype statistics.

getLocusList()

Returns the list of loci.

Note: this list has filtered out all loci that consist of individuals that are all untyped.

***Note 2: the order of this list is now fixed for the lifetime**
of the object.*

getAlleleCount()

Return allele count statistics for all loci.

Return a map of tuples where the key is the locus name. Each tuple is a triple, consisting of a map keyed by alleles containing counts, the total count at that locus and the number of untyped individuals.

getAlleleCountAt(*locus*, *lumpValue=0*)

Return allele count for given locus.

- 'lumpValue': the specified amount of lumping (Default: 0)

Given a locus name, return a tuple: consisting of a map keyed by alleles containing counts, the total count at that locus, and number of untyped individuals.

serializeSubclassMetadataTo(*stream*)

Serialize subclass-specific metadata.

Specifically, total number of individuals and loci and population name.

serializeAlleleCountDataAt(*stream*, *locus*)

serializeAlleleCountDataTo(*stream*)

getLocusDataAt(*locus*, *lumpValue=0*)

Returns the genotyped data for specified locus.

Given a 'locus', return a list genotypes consisting of 2-tuples which contain each of the alleles for that individual in the list.

- 'lumpValue': the specified amount of lumping (Default: 0)

Note: *this list has filtered out all individuals that are untyped at either chromosome.*

Note 2: data is sorted so that allele1 < allele2, alphabetically

getLocusData()

Returns the genotyped data for all loci.

Returns a dictionary keyed by locus name of lists of 2-tuples as defined by 'getLocusDataAt()'

getIndividualsData()

Returns the individual data.

Returns a 'StringMatrix'.

checkIfSequenceData(*matrix*)

getMetaLocus(*locus*, *isSequenceData*)

getLocusPairs(*matrix*, *sequenceData*)

Returns a list of all pairs of loci from a given StringMatrix

getLumpedDataLevels(*genotypeData*, *locus*, *lumpLevels*)

Returns a dictionary of tuples with alleleCount and locusData lumped by different levels specified as a list of integers.

class AlleleCounts(*alleleTable=None*, *locusName=None*, *debug=0*)

WARNING: this class is now obsolete, the Genotypes class now holds allele count data as pseudo-genotype matrix.

Class to store information in allele count form.

serializeSubclassMetadataTo(*stream*)

Serialize subclass-specific metadata.

Specifically, total number of alleles and loci.

serializeAlleleCountDataAt(*stream*, *locus*)

getAlleleCount()

getLocusName()

PyPop.Filter

Module for filtering data files.

Filters and cleans data before being accepted as input to PyPop analysis routines.

.

Exceptions

SubclassError	Common base class for all non-exit exceptions.

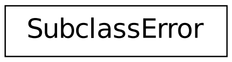
Classes

Filter	Abstract base class for Filters
PassThroughFilter	A filter that doesn't change input data.
AnthonyNolanFilter	Filters data via anthonynolan's allele call data.
BinningFilter	Filters data through rules defined in one file for each locus.
AlleleCountAnthonyNolanFilter	Filters data with an allelecount less than a threshold.

Module Contents

exception SubclassError

Bases: Exception[7]

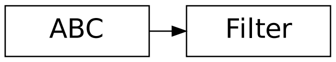


Common base class for all non-exit exceptions.

Initialize self. See help(type(self)) for accurate signature.

class Filter

Bases: abc.ABC[8]



Abstract base class for Filters

abstractmethod doFiltering(*matrix=None*)

abstractmethod startFirstPass(*locus*)

abstractmethod checkAlleleName(*alleleName*)

abstractmethod addAllele(*alleleName*)

abstractmethod endFirstPass()

abstractmethod startFiltering()

abstractmethod filterAllele(*alleleName*)

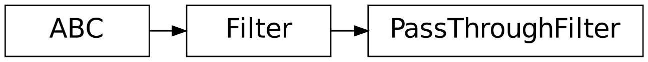
abstractmethod endFiltering()

abstractmethod writeToLog(*logstring=None*)

abstractmethod cleanup()

class PassThroughFilter

Bases: `Filter`



A filter that doesn't change input data.

doFiltering(*matrix=None*)

startFirstPass(*locus*)

checkAlleleName(*alleleName*)

addAllele(*alleleName*)

endFirstPass()

startFiltering()

filterAllele(*alleleName*)

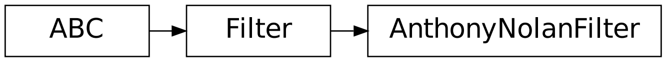
endFiltering()

writeToLog(*logstring=None*)

cleanup()

class AnthonyNolanFilter(*directoryName=None, remoteMSF=None, alleleFileFormat='msf', preserveAmbiguousFlag=0, preserveUnknownFlag=0, preserveLowresFlag=0, alleleDesignator='*', logFile=None, untypedAllele='****', unsequencedSite='#', sequenceFileSuffix='_prot', filename=None, numDigits=4, verboseFlag=1, debug=0, sequenceFilterMethod='strict'*)

Bases: `Filter`



Filters data via anthonynolan's allele call data.

Allele call data files can be of either txt or msf formats. txt files available at http://www.anthonynolan.com msf files available at ftp://ftp.ebi.ac.uk/pub/databases/imgt/mhc/hla/ Use of msf files is required in order to translate allele codes into polymorphic sequence data.

doFiltering(*matrix=None*)

Do filtering on StringMatrix

Given a StringMatrix, does the filtering on the matrix, and returns it for further downstream processing

startFirstPass(*locus*)

checkAlleleName(*alleleName*)

Checks allele name against the database.

Returns the allele truncated to appropriate number of digits, if it can't be found using any of the heuristics, return it as an untyped allele (normally four asterisks)

addAllele(*alleleName*)

endFirstPass()

startFiltering()

filterAllele(*alleleName*)

endFiltering()

writeToLog(*logstring='\n'*)

cleanup()

makeSeqDictionaries(*matrix=None*, *locus=None*)

translateMatrix(*matrix=None*)

class **BinningFilter**(*customBinningDict=None*, *logFile=None*, *untypedAllele='****'*, *filename=None*, *binningDigits=4*, *debug=0*)

Filters data through rules defined in one file for each locus.

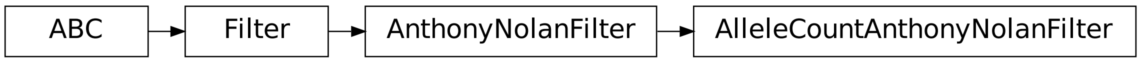
doDigitBinning(*matrix=None*)

doCustomBinning(*matrix=None*)

lookupCustomBinning(*testAllele*, *locus*)

class **AlleleCountAnthonyNolanFilter**(*lumpThreshold=None*, ***kw*)

Bases: *AnthonyNolanFilter*



Filters data with an allelecount less than a threshold.

endFirstPass()

Do regular AnthonyNolanFilter then translate alleles with count < lumpThreshold to 'lump'

PyPop.GUIApp

Attributes

ID_ABOUT
ID_OPEN_CONFIG
ID_OPEN_POP
ID_EXIT
EVT_RESULT_ID

Classes

ResultEvent	Simple event to carry arbitrary result data
WorkerThread	A class that represents a thread of control.
MainWindow	Creates the main application window for PyPop.

Functions

EVT_RESULT(win, func)

Module Contents

`ID_ABOUT = 101`

`ID_OPEN_CONFIG = 102`

`ID_OPEN_POP = 103`

`ID_EXIT = 110`

`EVT_RESULT_ID`

`EVT_RESULT`(*win*, *func*)

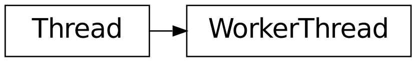
`class ResultEvent`(*data*)

Bases: `wxPyEvent`

Simple event to carry arbitrary result data

`class WorkerThread`(*notify_window*)

Bases: `threading.Thread`[9]



A class that represents a thread of control.

This class can be safely subclassed in a limited fashion. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the run() method in a subclass.

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is a list or tuple of arguments for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

`run`()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

`abort`()

`class MainWindow`(*parent*, *_id*, *title*, *datapath=None*, *altpath=None*, *debugFlag=0*)

Bases: `wxFrame`

Creates the main application window for PyPop.

`OnAbout`(*_event*)

`OnExit`(*_event*)

`OnConfig`(*event*)

Select config file

`OnPop`(*event*)

Select pop file

OnRun(*_event*)

OnStop(*_event*)

OnResult(*event*)

PyPop.Haplo

Module for estimating haplotypes.

Classes

Haplo	*Abstract* base class for haplotype parsing/output.
HaploArlequin	Haplotype estimation implemented via Arlequin
Emhaplofreq	Haplotype and LD estimation implemented via emhaplofreq.
Haplostats	Haplotype and LD estimation implemented via haplo-stats.

Module Contents

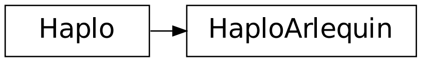
class Haplo

Abstract base class for haplotype parsing/output.

Currently a stub class (unimplemented).

class HaploArlequin(*arpFilename*, *idCol*, *prefixCols*, *suffixCols*, *windowSize*, *mapOrder=None*, *untypedAllele='0'*, *arlequinPrefix='arl_run'*, *debug=0*)

Bases: *Haplo*



Haplotype estimation implemented via Arlequin

Outputs Arlequin format data files and runtime info, also runs and parses the resulting Arlequin data so it can be made available programmatically to rest of Python framework.

Delegates all calls Arlequin to an internally instantiated ArlequinBatch Python object called 'batch'.

Constructor for HaploArlequin object.

Expects:

- arpFilename: Arlequin filename (must have '.arp' file extension)
- idCol: column in input file that contains the individual id.
- prefixCols: number of columns to ignore before allele data starts
- suffixCols: number of columns to ignore after allele data stops
- windowSize: size of sliding window
- mapOrder: list order of columns if different to column order in file (defaults to order in file)
- untypedAllele: (defaults to '0')
- arlequinPrefix: prefix for all Arlequin run-time files

(defaults to 'arl_run').

- debug: (defaults to 0)

outputArlequin(*data*)

Outputs the specified .arp sample file.

runArlequin()

Run the Arlequin haplotyping program.

Generates the expected '.txt' set-up files for Arlequin, then forks a copy of 'arlecore.exe', which must be on 'PATH' to actually generate the haplotype estimates from the generated '.arp' file.

`genHaplotypes()`

Gets the haplotype estimates back from Arlequin.

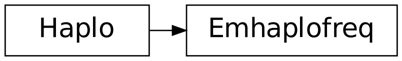
Parses the Arlequin output to retrieve the haplotype estimated data. Returns a list of the sliding `windows' which consists of tuples.

Each tuple consists of a:

- dictionary entry (the haplotype-frequency) key-value pairs.
- population name (original '.arp' file prefix)
- sample count (number of samples for that window)
- ordered list of loci considered

class Emhaplofreq(*locusData*, *debug=0*, *untypedAllele='****'*, *stream=None*, *testMode=False*)

Bases: *Haplo*



Haplotype and LD estimation implemented via emhaplofreq.

This is essentially a wrapper to a Python extension built on top of the 'emhaplofreq' command-line program.

Will refuse to estimate haplotypes longer than that defined by 'emhaplofreq'.

`serializeStart()`

Serialize start of XML output to XML stream

`serializeEnd()`

Serialize end of XML output to XML stream

`estHaplotypes`(*locusKeys=None*, *numInitCond=None*)

Estimate haplotypes for listed groups in 'locusKeys'.

Format of 'locusKeys' is a string consisting of:

- comma (',') separated haplotypes blocks for which to estimate haplotypes
- within each `block', each locus is separated by colons (':')

e.g. '*DQA1:*DPB1,*DRB1:*DQB1', means to est. haplotypes for
'DQA1' and 'DPB1' loci followed by est. of haplotypes for 'DRB1' and 'DQB1' loci.

`estLinkageDisequilibrium`(*locusKeys=None*, *permutationPrintFlag=0*, *numInitCond=None*, *numPermutations=None*, *numPermuInitCond=None*)

Estimate linkage disequilibrium (LD) for listed groups in 'locusKeys'.

Format of 'locusKeys' is a string consisting of:

- comma (',') separated haplotypes blocks for which to estimate haplotypes
- within each `block', each locus is separated by colons (':')

e.g. '*DQA1:*DPB1,*DRB1:*DQB1', means to est. LD for
'DQA1' and 'DPB1' loci followed by est. of LD for 'DRB1' and 'DQB1' loci.

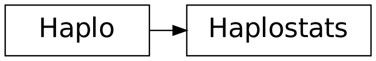
`allPairwise`(*permutationPrintFlag=0*, *numInitCond=None*, *numPermutations=None*, *numPermuInitCond=None*, *haploSuppressFlag=None*, *haplosToShow=None*, *mode=None*)

Run pairwise statistics.

Estimate pairwise statistics for a given set of loci. Depending on the flags passed, can be used to estimate both LD (linkage disequilibrium) and HF (haplotype frequencies), an optional permutation test on LD can be run

class Haplostats(*locusData*, *debug=0*, *untypedAllele='****'*, *stream=None*, *testMode=False*)

Bases: *Haplo*



Haplotype and LD estimation implemented via haplo-stats.

This is a wrapper to a portion of the 'haplo.stats' R package

serializeStart()

Serialize start of XML output to XML stream

serializeEnd()

Serialize end of XML output to XML stream

estHaplotypes(*locusKeys=None*, *weight=None*, *control=None*, *numInitCond=10*, *testMode=False*)

Estimate haplotypes for the submatrix given in locusKeys, if locusKeys is None, assume entire matrix

LD is estimated if there are locusKeys consists of only two loci

FIXME: this does *not* yet remove missing data before haplotype estimations

allPairwise(*weight=None*, *control=None*, *numInitCond=10*)

Estimate pairwise statistics for all pairs of loci.

PyPop.HardyWeinberg

Module for calculating Hardy-Weinberg statistics.

Attributes

use_scipy
chi2

Classes

HardyWeinberg	Calculate Hardy-Weinberg statistics.
HardyWeinbergGuoThompson	Wrapper class for 'gthwe'
HardyWeinbergEnumeration	Uses Hazael Maldonado Torres' exact enumeration test
HardyWeinbergGuoThompsonArlequin	Wrapper class for 'Arlequin'.

Functions

pval(chisq, dof)

Module Contents

use_scipy = False

chi2 = None

pval(*chisq*, *dof*)

class HardyWeinberg(*locusData=None*, *alleleCount=None*, *lumpBelow=5*, *flagChenTest=0*, *debug=0*)

Calculate Hardy-Weinberg statistics.

Given the observed genotypes for a locus, calculate the expected genotype counts based on Hardy Weinberg proportions for individual genotype values, and test for fit.

Constructor.

- locusData and alleleCount to be provided by driver script via a call to ParseFile.getLocusData(locus).
- lumpBelow: treat alleles with frequency less than this as if they were in same class (Default: 5)
- flagChenTest: if enabled do Chen's chi-square-based "corrected" p-value (Default: 0 [False])

serializeTo(*stream*, *allelelump=0*)

serializeXMLTableTo(*stream*)

class HardyWeinbergGuoThompson(*locusData=None*, *alleleCount=None*, *runMCMCTest=0*, *runPlainMCTest=0*, *dememorizationSteps=2000*, *samplingNum=1000*, *samplingSize=1000*, *maxMatrixSize=250*, *monteCarloSteps=1000000*, *testing=False*, ***kw*)

Bases: `HardyWeinberg`

HardyWeinberg
HardyWeinbergGuoThompson

Wrapper class for 'gthwe'

A wrapper for the Guo & Thompson program 'gthwe'.

- 'locusData', 'alleleCount': As per base class.

In addition to the arguments for the base class, this class accepts the following additional keywords:

- 'runMCMCTest': If enabled run the Monte Carlo-Markov chain (MCMC) version of the test (what is normally referred to as "Guo & Thompson")
- 'runPlainMCTest': If enabled run a plain Monte Carlo/randomization without the Markov-chain version of the test (this is also described in the original "Guo & Thompson" Biometrics paper, but was not in their original program)
- 'dememorizationSteps': number of `dememorization' initial steps for random number generator (default 2000).
- 'samplingNum': the number of chunks for random number generator (default 1000).
- 'samplingSize': size of each chunk (default 1000).
- **'maxMatrixSize': maximum size of `flattened' lower-triangular matrix of**
  observed alleles (default 250).
- **'monteCarloSteps': number of steps for the plain Monte Carlo**
  randomization test (without Markov-chain)

Constructor.

- locusData and alleleCount to be provided by driver script via a call to ParseFile.getLocusData(locus).
- lumpBelow: treat alleles with frequency less than this as if they were in same class (Default: 5)
- flagChenTest: if enabled do Chen's chi-square-based "corrected" p-value (Default: 0 [False])

generateFlattenedMatrix()

dumpTable(*locusName*, *stream*, *allelelump=0*)

class HardyWeinbergEnumeration(*locusData=None*, *alleleCount=None*, *doOverall=0*, ***kw*)

Bases: `HardyWeinbergGuoThompson`

HardyWeinberg
HardyWeinbergGuoThompson
HardyWeinbergEnumeration

Uses Hazael Maldonado Torres' exact enumeration test

- **'doOverall': if set to true ('1'), then do overall p-value test**
  default is false ('0')

Constructor.

- locusData and alleleCount to be provided by driver script via a call to ParseFile.getLocusData(locus).
- lumpBelow: treat alleles with frequency less than this as if they were in same class (Default: 5)
- flagChenTest: if enabled do Chen's chi-square-based "corrected" p-value (Default: 0 [False])

serializeTo(*stream*, *allelelump=0*)

class HardyWeinbergGuoThompsonArlequin(*matrix=None*, *locusName=None*, *arlequinExec='arlecore.exe'*, *markovChainStepsHW=100000*, *markovChainDememorisationStepsHW=1000*, *untypedAllele='****'*, *debug=None*)

Wrapper class for 'Arlequin'.

This class extracts the Hardy-Weinberg (HW) statistics using the Arlequin implementation of the HW exact test, by the following:

1. creates a subdirectory 'arlequinRuns' in which all the Arlequin specific files are generated;
2. then the specified arlequin executable is run, generating the Arlequin output HTML files (*.htm);
3. the Arlequin output is then parsed for the relevant statistics;
4. lastly, the 'arlequinRuns' directory is removed.

Since the directory name 'arlequinRuns' is currently hardcoded, this has the consequence that this class cannot be invoked concurrently.

Parameters:

- 'markovChainStepsHW': Number of steps to use in Markov chain

(default: 100000).

- 'markovChainDememorisationStepsHW': "Burn-in" time for Markov

chain (default: 1000).

serializeTo(*stream*)

PyPop.Homozygosity

Module for calculating homozygosity statistics.

Classes

Homozygosity	Calculate homozygosity statistics.
HomozygosityEWSlatkinExact	Calculate homozygosity statistics.
HomozygosityEWSlatkinExactPairwise	

Functions

getObservedHomozygosityFromAlleleData(alleleData)	

Module Contents

getObservedHomozygosityFromAlleleData(*alleleData*)

class **Homozygosity**(*alleleData*, *rootPath='.'*, *debug=0*)

Calculate homozygosity statistics.

Given allele count data for a given locus, calculates the observed homozygosity and returns the approximate expected homozygosity statistics taken from previous simulation runs.

Constructor for homozygosity statistics.

Given:

- 'alleleCountData': tuple consisting of a dictionary of alleles with their associated counts and the total number of alleles.
- 'rootPath': path to the root of the directory where the pre-calculated expected homozygosity statistics can be found.
- 'debug': flag to switch debugging on.

getObservedHomozygosity()

Calculate and return observed homozygosity.

Available even if expected stats cannot be calculated

canGenerateExpectedStats()

Can expected homozygosity stats be calculated?

Returns true if expected homozygosity statistics can be calculated. Should be called before attempting to get any expected homozygosity statistics.

getPValueRange()

Gets lower and upper bounds for p-value.

Returns a tuple of (lower, upper) bounds.

Only meaningful if 'canGenerateExpectedStats()' returns true.

getCount()

Number of runs used to calculate statistics.

Only meaningful if 'canGenerateExpectedStats()' returns true.

getExpectedHomozygosity()

Gets mean of expected homozygosity.

This is the estimate of the *expected* homozygosity.

Only meaningful if ‘canGenerateExpectedStats()’ returns true.

getVarExpectedHomozygosity()

Gets variance of expected homozygosity.

This is the estimate of the variance *expected* homozygosity.

Only meaningful if ‘canGenerateExpectedStats()’ returns true.

getNormDevHomozygosity()

Gets normalized deviate of homozygosity.

Only meaningful if ‘canGenerateExpectedStats()’ returns true.

serializeHomozygosityTo(*stream*)

class **HomozygosityEWSlatkinExact**(*alleleData=None*, *numReplicates=10000*, *debug=0*)

Bases: *Homozygosity*

Homozygosity
HomozygosityEWSlatkinExact

Calculate homozygosity statistics.

Given allele count data for a given locus, calculates the observed homozygosity and returns the approximate expected homozygosity statistics taken from previous simulation runs.

Constructor for homozygosity statistics.

Given:

- ‘alleleCountData’: tuple consisting of a dictionary of alleles with their associated counts and the total number of alleles.
- ‘rootPath’: path to the root of the directory where the pre-calculated expected homozygosity statistics can be found.
- ‘debug’: flag to switch debugging on.

doCalcs(*alleleData*)

getHomozygosity()

serializeHomozygosityTo(*stream*)

returnBulkHomozygosityStats(*alleleCountDict=None*, *binningMethod=None*)

class **HomozygosityEWSlatkinExactPairwise**(*matrix=None*, *numReplicates=10000*, *untypedAllele='****'*, *debug=0*)

serializeTo(*stream*)

PyPop.Main

Python population genetics statistics.

Classes

Main	Main interface to the PyPop modules.

Functions

getConfigInstance([configFilename, altpath])	Create and return ConfigParser instance.
get_sequence_directory(directory_str[, debug])	

Module Contents

getConfigInstance(*configFilename=None*, *altpath=None*)

Create and return ConfigParser instance.

Taken a specific .ini filename and an alternative path to search if no .ini filename is given.

get_sequence_directory(*directory_str*, *debug=False*)

class Main(*config=None*, *xslFilename=None*, *xslFilenameDefault=None*, *debugFlag=0*, *fileName=None*, *datapath=None*, *thread=None*, *outputDir=None*, *version=None*, *testMode=False*)

Main interface to the PyPop modules.

Given a config instance, which can be:

- created from a filename passed from command-line argument or;
- from values populated by the GUI (currently selected from an .ini file, but can ultimately be set directly from the GUI or values based from a form to a web server or the).

runs the specified modules.

getXmlOutPath()

getTxtOutPath()

PyPop.Meta

Module for collecting multiple population outputs.

Classes

Meta	Aggregates output from multiple population runs.

Functions

translate_string_to_stdout(xslFilename, inString[, ...])
translate_string_to_file(xslFilename, inString, outFile)
translate_file_to_stdout(xslFilename, inFile[, ...])
translate_file_to_file(xslFilename, inFile, outFile[, ...])

Module Contents

translate_string_to_stdout(*xslFilename*, *inString*, *outputDir=None*, *params=None*)

translate_string_to_file(*xslFilename*, *inString*, *outFile*, *outputDir=None*, *params=None*)

translate_file_to_stdout(*xslFilename*, *inFile*, *inputDir=None*, *params=None*)

translate_file_to_file(*xslFilename*, *inFile*, *outFile*, *inputDir=None*, *outputDir=None*, *params=None*)

class Meta(*popmetabinpath=None*, *datapath=None*, *metaXSLTDirectory=None*, *dump_meta=False*, *TSV_output=True*, *prefixTSV=None*, *PHYLIP_output=False*, *ihwg_output=False*, *batchsize=0*, *outputDir=None*, *xml_files=None*)

Aggregates output from multiple population runs.

Transform a specified list of XML output files to *.tsv tab-separated values (TSV) form.

Defaults: # output .tsv tables by default (can be used by R) TSV_output=True

# don't output PHYLIP by default PHYLIP_output=False

# by default, don't enable the 13th IHWG format headers ihwg_output = False

# by default process separately (batchsize=0) batchsize = 0

PyPop.ParseFile

Module for parsing data files.

Includes classes for parsing individuals genotyped at multiple loci and classes for parsing literature data which only includes allele counts.

Classes

`ParseFile`	*Abstract* class for parsing a datafile.
`ParseGenotypeFile`	Class to parse standard datafile in genotype form.
`ParseAlleleCountFile`	Class to parse datafile in allele count form.

Module Contents

class ParseFile(*filename*, *validPopFields=None*, *validSampleFields=None*, *separator='\t'*, *fieldPairDesignator='_1:_2'*, *alleleDesignator='*'*, *popNameDesignator='+'*, *debug=0*)

Abstract class for parsing a datafile.

Not to be instantiated.

Constructor for ParseFile object.

- 'filename': filename for the file to be parsed.
- **'validPopFields': a string consisting of valid headers (one**
  per line) for overall population data (no default)
- **'validSampleFields': a string consisting of valid headers**
  (one per line) for lines of sample data. (no default)
- **'separator': separator for adjacent fields (default: a tab**
  stop, 't').
- 'fieldPairDesignator': a string which consists of additions to the allele `stem' for fields grouped in pairs (allele fields) [e.g. for `HLA-A', and `HLA-A(2)', then we use ':(2)', for `DQA1_1' and `DQA1_2', then use use '_1:_2', the latter case distinguishes both fields from the stem] (default: ':(2)')
- 'alleleDesignator': The first character of the key which

determines whether this column contains allele data. Defaults to '*'

- 'popNameDesignator': The first character of the key which

determines whether this column contains the population name. Defaults to '+'

- 'debug': Switches debugging on if set to '1' (default: no debugging, '0')

getPopData()

Returns a dictionary of population data.

Dictionary is keyed by types specified in population metadata file

getSampleMap()

Returns dictionary of sample data.

Each dictionary position contains either a 2-tuple of column position or a single column position keyed by field originally specified in sample metadata file

getFileData()

Returns file data.

Returns a 2-tuple `wrapper':

- raw sample lines, *without* header metadata.
- the field separator.

genSampleOutput(*fieldList*)

Prints the data specified in ordered field list.

Use is currently deprecated.

serializeMetadataTo(*stream*)

class ParseGenotypeFile(*filename*, *untypedAllele='****'*, ***kw*)

Bases: *ParseFile*

ParseFile
ParseGenotypeFile

Class to parse standard datafile in genotype form.

Constructor for ParseGenotypeFile.

- 'filename': filename for the file to be parsed.

In addition to the arguments for the base class, this class accepts the following additional keywords:

- 'untypedAllele': The designator for an untyped locus. Defaults

to '****'.

genValidKey(*field*, *fieldList*)

Check and validate key.

- 'field': string with field name.
- 'fieldList': a dictionary of valid fields.

Check to see whether 'field' is a valid key, and generate the appropriate 'key'. Returns a 2-tuple consisting of 'isValidKey' boolean and the 'key'.

Note: this is explicitly done in the subclass of the abstract 'ParseFile' class (i.e. since this subclass should have `knowledge' about the nature of fields, but the abstract class should not have)

getMatrix()

Returns the genotype data.

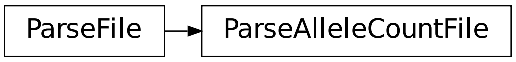
Returns the genotype data in a 'StringMatrix' NumPy array.

serializeSubclassMetadataTo(*stream*)

Serialize subclass-specific metadata.

class ParseAlleleCountFile(*filename*, ***kw*)

Bases: *ParseFile*



Class to parse datafile in allele count form.

Currently only handles one locus per population, in format:

<metadata-line1> <metadata-line2> DQA1 count 0102 20 0103 33 …

Currently a prototype implementation.

Constructor for ParseFile object.

- 'filename': filename for the file to be parsed.
- **'validPopFields': a string consisting of valid headers (one**
  per line) for overall population data (no default)
- **'validSampleFields': a string consisting of valid headers**
  (one per line) for lines of sample data. (no default)
- **'separator': separator for adjacent fields (default: a tab**
  stop, 't').
- 'fieldPairDesignator': a string which consists of additions to the allele `stem' for fields grouped in pairs (allele fields) [e.g. for `HLA-A', and `HLA-A(2)', then we use ':(2)', for `DQA1_1' and `DQA1_2', then use use '_1:_2', the latter case distinguishes both fields from the stem] (default: ':(2)')
- 'alleleDesignator': The first character of the key which

determines whether this column contains allele data. Defaults to '*'

- 'popNameDesignator': The first character of the key which

determines whether this column contains the population name. Defaults to '+'

- 'debug': Switches debugging on if set to '1' (default: no debugging, '0')

genValidKey(*field*, *fieldList*)

Checks to see validity of a field.

Given a 'field', this is checked against the 'fieldList' and a tuple of a boolean (key is valid) and a a key is returned.

The first element in the 'fieldList' which is a locus name, can match one of many loci (delimited by colons ':'). E.g. it may look like:

'DQA1:DRA:DQB1'

If the field in the input file match *any* of these keys, return the field and a valid match.

serializeSubclassMetadataTo(*stream*)

getAlleleTable()

getLocusName()

getMatrix()

Returns the genotype data.

Returns the genotype data in a 'StringMatrix' NumPy array.

PyPop.RandomBinning

Python population genetics statistics.

Classes

RandomBinsForHomozygosity	

Module Contents

class RandomBinsForHomozygosity(*logFile=None*, *untypedAllele='****'*, *filename=None*, *numReplicates=10000*, *binningReplicates=100*, *locus=None*, *xmlfile=None*, *debug=0*, *randomResultsFileName=None*)

randomMethod(*alleleCountsBefore=None*, *alleleCountsAfter=None*)

sequenceMethod(*alleleCountsBefore=None*, *alleleCountsAfter=None*, *polyseq=None*, *polyseqpos=None*)

PyPop.Utils

Module for common utility classes and functions.

Contains convenience classes for output of text and XML files.

Attributes

GENOTYPE_SEPARATOR	
GENOTYPE_TERMINATOR	

Classes

TextOutputStream	Output stream for writing text files.
XMLOutputStream	Output stream for writing XML files.
OrderedDict	Allows dict to have _ORDERED_ pairs
Index	Returns an Index object for OrderedDict
StringMatrix	StringMatrix is a subclass of NumPy (Numeric Python)
Group	

Functions

glob_with_pathlib(pattern)	
getStreamType(stream)	Return the type of stream.
natural_sort_key(s[, _nsre])	
unique_elements(li)	Gets the unique elements in a list
appendTo2dList(aList[, appendStr])	
convertLineEndings(file, mode)	
fixForPlatform(filename[, txt_ext])	
copyfileCustomPlatform(src, dest[, txt_ext])	
copyCustomPlatform(file, dist_dir[, txt_ext])	
checkXSLFile(xslFilename[, path, subdir, abort, ...])	
getUserFilenameInput(prompt, filename)	Read user input for a filename, check its existence, continue
splitIntoNGroups(alist[, n])	Divides a list up into n parcels (plus whatever is left over)

Module Contents

GENOTYPE_SEPARATOR = '~'

GENOTYPE_TERMINATOR = '~'

glob_with_pathlib(*pattern*)

class **TextOutputStream**(*file*)

Output stream for writing text files.

write(*str*)

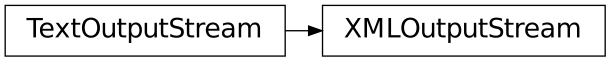
writeln(*str='\n'*)

close()

flush()

class **XMLOutputStream**(*file*)

Bases: *TextOutputStream*



Output stream for writing XML files.

opentag(*tagname*, ***kw*)

Generate an open XML tag.

Generate an open XML tag. Attributes are passed in the form of optional named arguments, e.g. opentag('tagname', role=something, id=else) will produce the result '<tagname role="something" id="else"> Note that the attribute and values are optional and if omitted produce '<tag-name>'.

emptytag(*tagname*, ***kw*)

Generate an empty XML tag.

As per 'opentag()' but without content, i.e.:

'<tagname attr="val"/>'.

closetag(*tagname*)

Generate a closing XML tag.

Generate a tag in the form: '</tagname>'.

`tagContents`(*tagname*, *content*, ***kw*)

Generate open and closing XML tags around contents.

Generates tags in the form: '<tagname>content</tagname>'. 'content' must be a string. Convert '&' and '<' and '>' into valid XML equivalents.

`getStreamType`(*stream*)

Return the type of stream.

Returns either 'xml' or 'text'.

class `OrderedDict`(*hash=None*)

Allows dict to have _ORDERED_ pairs

Creates an ordered dict

`index`(*key*)

Returns position of key in dict

`keys`()

Returns list of keys in dict

`values`()

Returns list of values in dict

`items`()

Returns list of tuples of keys and values

`insert`(*i*, *key*, *value*)

Inserts a key-value pair at a given index

`remove`(*i*)

Removes a key-value pair from the dict

`reverse`()

Reverses the order of the key-value pairs

`sort`(*cmp=0*)

Sorts the dict (allows for sort algorithm)

`clear`()

Clears all the entries in the dict

`copy`()

Makes copy of dict, also of OrderdDict class

`get`(*key*)

Returns the value of a key

`has_key`(*key*)

Looks for existence of key in dict

`update`(*dict*)

Updates entries in a dict based on another

`count`(*key*)

Finds occurrences of a key in a dict (0/1)

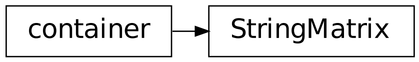
class `Index`(*i=0*)

Returns an Index object for OrderedDict

Creates an Index object for use with OrderedDict

class `StringMatrix`(*rowCount=None*, *colList=None*, *extraList=None*, *colSep='\t'*, *headerLines=None*)

Bases: `numpy.lib.user_array.container`[10]



StringMatrix is a subclass of NumPy (Numeric Python) UserArray class, and uses NumPy to store the data in an efficient array format, rather than internal Python lists.

Constructor for StringMatrix.

colList is a mutable type so we freeze the list of locus keys in the original order in file by making a *clone* of the list of keys.

the order of loci in the array will correspond to the original file order, and we don't want this tampered with by the `callee' function (i.e. effectively override the Python 'pass by reference' default and 'pass by value').

`dump`(*locus=None*, *stream=sys.stdout*)

`copy`()

Make a (deep) copy of the StringMatrix

Currently this goes via the constructor, not sure if there is a better way of doing this

`getNewStringMatrix`(*key*)

Create an entirely new StringMatrix using only the columns supplied in the keys.

The format of the keys is identical to __getitem__ except that it in this case returns a full StringMatrix instance which includes all metadata

`getUniqueAlleles`(*key*)

Return a list of unique integers for given key sorted by allele name using natural sort

`convertToInts`()

Convert matrix to integers: needed for haplo-stats Note that integers start at 1 for compatibility with haplo-stats module FIXME: check whether we need to release memory

`countPairs`()

Given a matrix of genotypes (pairs of columns for each locus), compute number of possible pairs of haplotypes for each subject (the rows of the geno matrix)

FIXME: this does *not* do any involved handling of missing data as per geno.count.pairs from haplo.stats

FIXME: should these methods eventually be moved to Genotype class?

`flattenCols`()

Flatten columns into a single list FIXME: assumes entries are integers

`filterOut`(*key*, *blankDesignator*)

Returns a filtered matrix.

When passed a designator, this method will return the rows of the matrix that *do not* contain that designator at any rows

`getSuperType`(*key*)

Returns a matrix grouped by columns.

e.g if matrix is [[A01, A02, B01, B02], [A11, A12, B11, B12]]

then getSuperType('A:B') will return the matrix with the column vector:

[[A01:B01, A02:B02], [A11:B11, A12:B12]]

`class Group`(*li*, *size*)

`natural_sort_key`(*s*, *_nsre=re.compile('([0-9]+)')*)

`unique_elements`(*li*)

Gets the unique elements in a list

`appendTo2dList`(*aList*, *appendStr=':'*)

`convertLineEndings`(*file*, *mode*)

`fixForPlatform`(*filename*, *txt_ext=0*)

`copyfileCustomPlatform`(*src*, *dest*, *txt_ext=0*)

`copyCustomPlatform`(*file*, *dist_dir*, *txt_ext=0*)

`checkXSLFile`(*xslFilename*, *path=''*, *subdir=''*, *abort=False*, *debug=None*, *msg=''*)

`getUserFilenameInput`(*prompt*, *filename*)

Read user input for a filename, check its existence, continue requesting input until a valid filename is entered.

`splitIntoNGroups`(*alist*, *n=1*)

Divides a list up into n parcels (plus whatever is left over)

This class currently works with Python 2.2, but will eventually use iterators, so ultimately will need least Python 2.3!

PyPop.citation

Attributes

citation_output_formats

Functions

convert_citation_formats(build_lib, citation_path)

Module Contents

citation_output_formats = ['apalike', 'bibtex', 'endnote', 'ris', 'codemeta', 'cff', 'schema.org', 'zenodo']

convert_citation_formats(*build_lib*, *citation_path*)

PyPop.popmeta

Attributes

DIR

Functions

main([argv])

Module Contents

main(*argv=sys.argv*)

DIR

PyPop.pypop

Python population genetics statistics.

Attributes

DIR

Functions

main([argv])
main_interactive([argv])

Module Contents

main(*argv=sys.argv*)

main_interactive(*argv=sys.argv*)

DIR

PyPop.xslt

Python XSLT extensions for handling things outside the scope of XSLT 1.0

Attributes

ns	
root	

Functions

num_zeros(decimal)	
exponent_len(num)	
format_number_fixed_width(_context, *args)	

Package Contents

ns

num_zeros(*decimal*)

exponent_len(*num*)

format_number_fixed_width(*_context*, **args*)

root

3 Attributes

copyright_message	
platform_info	

4 Functions

setup_logging([debug, filename])	Provide defaults for logging.

5 Package Contents

copyright_message = Multiline-String

```
"""Copyright (C) 2003-2006 Regents of the University of California.
Copyright (C) 2007-2025 PyPop team.
This is free software.  There is NO warranty; not even for
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE."""
```

platform_info = '[Python Uninferable | Uninferable | Uninferable]'

setup_logging(*debug=False*, *filename=None*)

Provide defaults for logging.

6 GNU Free Documentation License

"""

Python Module Index

p

PyPop, 1
PyPop.Arlequin, 1
PyPop.citation, 23
PyPop.CommandLineInterface, 3
PyPop.DataTypes, 4
PyPop.Filter, 6
PyPop.GUIApp, 8
PyPop.Haplo, 10
PyPop.HardyWeinberg, 12
PyPop.Homozygosity, 14
PyPop.Main, 15
PyPop.Meta, 16
PyPop.ParseFile, 17
PyPop.popmeta, 23
PyPop.pypop, 23
PyPop.RandomBinning, 19
PyPop.Utils, 19
PyPop.xslt, 24

Notes

1. https://github.com/readthedocs/sphinx-autoapi
2. http://pypop.org/docs
3. http://pypop.org/pypop-guide-1.3.1.pdf
4. https://docs.python.org/3/library/argparse.html#argparse.ArgumentDefaultsHelpFormatter
5. https://docs.python.org/3/library/argparse.html#argparse.RawDescriptionHelpFormatter
6. https://docs.python.org/3/library/argparse.html#argparse.Action
7. https://docs.python.org/3/library/exceptions.html#Exception
8. https://docs.python.org/3/library/abc.html#abc.ABC
9. https://docs.python.org/3/library/threading.html#threading.Thread
10. https://numpy.org/doc/stable/reference/generated/numpy.lib.user\protect_array.container.html#numpy.lib.user\protect_array.container

Index

A

abort() (*WorkerThread method*), 9
addAllele() (*AnthonyNolanFilter method*), 7
addAllele() (*Filter method*), 7
addAllele() (*PassThroughFilter method*), 7
AlleleCountAnthonyNolanFilter (*class in PyPop.Filter*), 8
AlleleCounts (*class in PyPop.DataTypes*), 6
allPairwise() (*Emhaplofreq method*), 11
allPairwise() (*Haplostats method*), 12
AnthonyNolanFilter (*class in PyPop.Filter*), 7
appendTo2dList() (*in module PyPop.Utils*), 22
ArlequinBatch (*class in PyPop.Arlequin*), 2
ArlequinExactHWTest (*class in PyPop.Arlequin*), 2
ArlequinWrapper (*class in PyPop.Arlequin*), 2

B

BinningFilter (*class in PyPop.Filter*), 8

C

canGenerateExpectedStats() (*Homozygosity method*), 14
checkAlleleName() (*AnthonyNolanFilter method*), 7
checkAlleleName() (*Filter method*), 7
checkAlleleName() (*PassThroughFilter method*), 7
checkIfSequenceData() (*in module PyPop.DataTypes*), 5
checkXSLFile() (*in module PyPop.Utils*), 22
chi2 (*in module PyPop.HardyWeinberg*), 12
citation_output_formats (*in module PyPop.citation*), 23
CitationAction (*class in PyPop.CommandLineInterface*), 4
cleanup() (*AnthonyNolanFilter method*), 8
cleanup() (*ArlequinWrapper method*), 2
cleanup() (*Filter method*), 7
cleanup() (*PassThroughFilter method*), 7
clear() (*OrderedDict method*), 21
close() (*TextOutputStream method*), 20
closetag() (*XMLOutputStream method*), 20
convert_citation_formats() (*in module PyPop.citation*), 23
convertLineEndings() (*in module PyPop.Utils*), 22
convertToInts() (*StringMatrix method*), 22
copy() (*OrderedDict method*), 21
copy() (*StringMatrix method*), 22
copyCustomPlatform() (*in module PyPop.Utils*), 22
copyfileCustomPlatform() (*in module PyPop.Utils*), 22
copyright_message (*in module PyPop*), 24
count() (*OrderedDict method*), 21
countPairs() (*StringMatrix method*), 22

D

DIR (*in module PyPop.popmeta*), 23
DIR (*in module PyPop.pypop*), 23
doCalcs() (*HomozygosityEWSlatkinExact method*), 15
doCustomBinning() (*BinningFilter method*), 8
doDigitBinning() (*BinningFilter method*), 8
doFiltering() (*AnthonyNolanFilter method*), 7
doFiltering() (*Filter method*), 6
doFiltering() (*PassThroughFilter method*), 7
dump() (*StringMatrix method*), 22
dumpTable() (*HardyWeinbergGuoThompson method*), 13

E

Emhaplofreq (*class in PyPop.Haplo*), 11
emptytag() (*XMLOutputStream method*), 20
endFiltering() (*AnthonyNolanFilter method*), 8
endFiltering() (*Filter method*), 7
endFiltering() (*PassThroughFilter method*), 7
endFirstPass() (*AlleleCountAnthonyNolanFilter method*), 8
endFirstPass() (*AnthonyNolanFilter method*), 8
endFirstPass() (*Filter method*), 7
endFirstPass() (*PassThroughFilter method*), 7
estHaplotypes() (*Emhaplofreq method*), 11
estHaplotypes() (*Haplostats method*), 12
estLinkageDisequilibrium() (*Emhaplofreq method*), 11
EVT_RESULT() (*in module PyPop.GUIApp*), 9
EVT_RESULT_ID (*in module PyPop.GUIApp*), 9
exponent_len() (*in module PyPop.xslt*), 24

F

Filter (*class in PyPop.Filter*), 6
filterAllele() (*AnthonyNolanFilter method*), 8
filterAllele() (*Filter method*), 7
filterAllele() (*PassThroughFilter method*), 7
filterOut() (*StringMatrix method*), 22
fixForPlatform() (*in module PyPop.Utils*), 22
flattenCols() (*StringMatrix method*), 22
flush() (*TextOutputStream method*), 20
format_number_fixed_width() (*in module PyPop.xslt*), 24

G

generateFlattenedMatrix() (*HardyWeinbergGuoThompson method*), 13
genHaplotypes() (*HaploArlequin method*), 10
GENOTYPE_SEPARATOR (*in module PyPop.Utils*), 20
GENOTYPE_TERMINATOR (*in module PyPop.Utils*), 20
Genotypes (*class in PyPop.DataTypes*), 5
genSampleOutput() (*ParseFile method*), 17
genValidKey() (*ParseAlleleCountFile method*), 18
genValidKey() (*ParseGenotypeFile method*), 18
get() (*OrderedDict method*), 21
get_parent_cli() (*in module PyPop.CommandLineInterface*), 4
get_popmeta_cli() (*in module PyPop.CommandLineInterface*), 4
get_pypop_cli() (*in module PyPop.CommandLineInterface*), 4
get_sequence_directory() (*in module PyPop.Main*), 16
getAlleleCount() (*AlleleCounts method*), 6
getAlleleCount() (*Genotypes method*), 5
getAlleleCountAt() (*Genotypes method*), 5
getAlleleTable() (*ParseAlleleCountFile method*), 19
getConfigInstance() (*in module PyPop.Main*), 16
getCount() (*Homozygosity method*), 14
getExpectedHomozygosity() (*Homozygosity method*), 14
getFileData() (*ParseFile method*), 17
getHomozygosity() (*HomozygosityEWSlatkinExact method*), 15
getHWExactTest() (*ArlequinExactHWTest method*), 2
getIndividualsData() (*Genotypes method*), 5
getLocusData() (*Genotypes method*), 5
getLocusDataAt() (*Genotypes method*), 5
getLocusList() (*Genotypes method*), 5
getLocusName() (*AlleleCounts method*), 6
getLocusName() (*ParseAlleleCountFile method*), 19
getLocusPairs() (*in module PyPop.DataTypes*), 6
getLumpedDataLevels() (*in module PyPop.DataTypes*), 6
getMatrix() (*ParseAlleleCountFile method*), 19
getMatrix() (*ParseGenotypeFile method*), 18
getMetaLocus() (*in module PyPop.DataTypes*), 5
getNewStringMatrix() (*StringMatrix method*), 22
getNormDevHomozygosity() (*Homozygosity method*), 15
getObservedHomozygosity() (*Homozygosity method*), 14
getObservedHomozygosityFromAlleleData() (*in module PyPop.Homozygosity*), 14
getPopData() (*ParseFile method*), 17
getPValueRange() (*Homozygosity method*), 14
getSampleMap() (*ParseFile method*), 17

getStreamType() (*in module PyPop.Utils*), 21
getSuperType() (*StringMatrix method*), 22
getTxtOutPath() (*Main method*), 16
getUniqueAlleles() (*StringMatrix method*), 22
getUserFilenameInput() (*in module PyPop.Utils*), 22
getVarExpectedHomozygosity() (*Homozygosity method*), 15
getXmlOutPath() (*Main method*), 16
glob_with_pathlib() (*in module PyPop.Utils*), 20
Group (*class in PyPop.Utils*), 22

H

Haplo (*class in PyPop.Haplo*), 10
HaploArlequin (*class in PyPop.Haplo*), 10
Haplostats (*class in PyPop.Haplo*), 11
HardyWeinberg (*class in PyPop.HardyWeinberg*), 12
HardyWeinbergEnumeration (*class in PyPop.HardyWeinberg*), 13
HardyWeinbergGuoThompson (*class in PyPop.HardyWeinberg*), 12
HardyWeinbergGuoThompsonArlequin (*class in PyPop.HardyWeinberg*), 13
has_key() (*OrderedDict method*), 21
Homozygosity (*class in PyPop.Homozygosity*), 14
HomozygosityEWSlatkinExact (*class in PyPop.Homozygosity*), 15
HomozygosityEWSlatkinExactPairwise (*class in PyPop.Homozygosity*), 15

I

ID_ABOUT (*in module PyPop.GUIApp*), 9
ID_EXIT (*in module PyPop.GUIApp*), 9
ID_OPEN_CONFIG (*in module PyPop.GUIApp*), 9
ID_OPEN_POP (*in module PyPop.GUIApp*), 9
Index (*class in PyPop.Utils*), 21
index() (*OrderedDict method*), 21
insert() (*OrderedDict method*), 21
items() (*OrderedDict method*), 21

K

keys() (*OrderedDict method*), 21

L

lookupCustomBinning() (*BinningFilter method*), 8

M

Main (*class in PyPop.Main*), 16
main() (*in module PyPop.popmeta*), 23
main() (*in module PyPop.pypop*), 23
main_interactive() (*in module PyPop.pypop*), 23
MainWindow (*class in PyPop.GUIApp*), 9
makeSeqDictionaries() (*AnthonyNolanFilter method*), 8
Meta (*class in PyPop.Meta*), 16
module
    PyPop, 1
    PyPop.Arlequin, 1
    PyPop.citation, 23
    PyPop.CommandLineInterface, 3
    PyPop.DataTypes, 4
    PyPop.Filter, 6
    PyPop.GUIApp, 8
    PyPop.Haplo, 10
    PyPop.HardyWeinberg, 12
    PyPop.Homozygosity, 14
    PyPop.Main, 15
    PyPop.Meta, 16
    PyPop.ParseFile, 17
    PyPop.popmeta, 23
    PyPop.pypop, 23
    PyPop.RandomBinning, 19
    PyPop.Utils, 19
    PyPop.xslt, 24

N

natural_sort_key() (*in module PyPop.Utils*), 22
ns (*in module PyPop.xslt*), 24
num_zeros() (*in module PyPop.xslt*), 24

O

OnAbout() (*MainWindow method*), 9
OnConfig() (*MainWindow method*), 9
OnExit() (*MainWindow method*), 9
OnPop() (*MainWindow method*), 9
OnResult() (*MainWindow method*), 10
OnRun() (*MainWindow method*), 9
OnStop() (*MainWindow method*), 10
opentag() (*XMLOutputStream method*), 20
OrderedDict (*class in PyPop.Utils*), 21
outputArlequin() (*ArlequinBatch method*), 3
outputArlequin() (*HaploArlequin method*), 10
outputArpFile() (*ArlequinWrapper method*), 2
outputArsFile() (*ArlequinWrapper method*), 2
outputRunFiles() (*ArlequinBatch method*), 3
outputRunFiles() (*ArlequinWrapper method*), 2

P

ParseAlleleCountFile (*class in PyPop.ParseFile*), 18
ParseFile (*class in PyPop.ParseFile*), 17
ParseGenotypeFile (*class in PyPop.ParseFile*), 17
PassThroughFilter (*class in PyPop.Filter*), 7
platform_info (*in module PyPop*), 24
pval() (*in module PyPop.HardyWeinberg*), 12
PyPop
    module, 1
PyPop.Arlequin
    module, 1
PyPop.citation
    module, 23
PyPop.CommandLineInterface
    module, 3
PyPop.DataTypes
    module, 4
PyPop.Filter
    module, 6
PyPop.GUIApp
    module, 8
PyPop.Haplo
    module, 10
PyPop.HardyWeinberg
    module, 12
PyPop.Homozygosity
    module, 14
PyPop.Main
    module, 15
PyPop.Meta
    module, 16
PyPop.ParseFile
    module, 17
PyPop.popmeta
    module, 23
PyPop.pypop
    module, 23
PyPop.RandomBinning
    module, 19
PyPop.Utils
    module, 19
PyPop.xslt
    module, 24
PyPopFormatter (*class in PyPop.CommandLineInterface*), 4

R

RandomBinsForHomozygosity (*class in PyPop.RandomBinning*), 19
randomMethod() (*RandomBinsForHomozygosity method*), 19

remove() (*OrderedDict method*), 21
ResultEvent (*class in PyPop.GUIApp*), 9
returnBulkHomozygosityStats() (*HomozygosityEWSlatkinExact method*), 15
reverse() (*OrderedDict method*), 21
root (*in module PyPop.xslt*), 24
run() (*WorkerThread method*), 9
runArlequin() (*ArlequinBatch method*), 3
runArlequin() (*ArlequinWrapper method*), 2
runArlequin() (*HaploArlequin method*), 10

S

sequenceMethod() (*RandomBinsForHomozygosity method*), 19
serializeAlleleCountDataAt() (*AlleleCounts method*), 6
serializeAlleleCountDataAt() (*Genotypes method*), 5
serializeAlleleCountDataTo() (*Genotypes method*), 5
serializeEnd() (*Emhaplofreq method*), 11
serializeEnd() (*Haplostats method*), 12
serializeHomozygosityTo() (*Homozygosity method*), 15
serializeHomozygosityTo() (*HomozygosityEWSlatkinExact method*), 15
serializeMetadataTo() (*ParseFile method*), 17
serializeStart() (*Emhaplofreq method*), 11
serializeStart() (*Haplostats method*), 11
serializeSubclassMetadataTo() (*AlleleCounts method*), 6
serializeSubclassMetadataTo() (*Genotypes method*), 5
serializeSubclassMetadataTo() (*ParseAlleleCountFile method*), 19
serializeSubclassMetadataTo() (*ParseGenotypeFile method*), 18
serializeTo() (*HardyWeinberg method*), 12
serializeTo() (*HardyWeinbergEnumeration method*), 13
serializeTo() (*HardyWeinbergGuoThompsonArlequin method*), 14
serializeTo() (*HomozygosityEWSlatkinExactPairwise method*), 15
serializeXMLTableTo() (*HardyWeinberg method*), 12
setup_logging() (*in module PyPop*), 24
sort() (*OrderedDict method*), 21
splitIntoNGroups() (*in module PyPop.Utils*), 22
startFiltering() (*AnthonyNolanFilter method*), 8
startFiltering() (*Filter method*), 7
startFiltering() (*PassThroughFilter method*), 7
startFirstPass() (*AnthonyNolanFilter method*), 7
startFirstPass() (*Filter method*), 6
startFirstPass() (*PassThroughFilter method*), 7
StringMatrix (*class in PyPop.Utils*), 21
SubclassError, 6

T

tagContents() (*XMLOutputStream method*), 20
TextOutputStream (*class in PyPop.Utils*), 20
translate_file_to_file() (*in module PyPop.Meta*), 16
translate_file_to_stdout() (*in module PyPop.Meta*), 16
translate_string_to_file() (*in module PyPop.Meta*), 16
translate_string_to_stdout() (*in module PyPop.Meta*), 16
translateMatrix() (*AnthonyNolanFilter method*), 8

U

unique_elements() (*in module PyPop.Utils*), 22
update() (*OrderedDict method*), 21
usage_message (*in module PyPop.Arlequin*), 3
use_scipy (*in module PyPop.HardyWeinberg*), 12

V

values() (*OrderedDict method*), 21

W

WorkerThread (*class in PyPop.GUIApp*), 9
write() (*TextOutputStream method*), 20
writeln() (*TextOutputStream method*), 20
writeToLog() (*AnthonyNolanFilter method*), 8
writeToLog() (*Filter method*), 7
writeToLog() (*PassThroughFilter method*), 7

X

XMLOutputStream (*class in PyPop.Utils*), 20