anhima Documentation

Release 0.11.1

Alistair Miles <alimanfoo@googlemail.com>

July 22, 2015

Contents

1 Locating samples and variants 3

2 Genotypes 11

3 Allele frequencies 25

4 Site frequencies 33

5 Doubleton sharing 37

6 Linkage disequilibrium 41

7 Genetic distance 47

8 Principal components analysis 49

9 Multidimensional scaling 53

10 Trees 55

11 Pedigrees (families and crosses) 59

12 Input/output utilities 63

13 HDF5 utilities 65

14 Miscellaneous utilities 69

15 Simulations 71

Python Module Index 73

This package is no longer under active development. Existing functionality is gradually being migrated into the 'scikit-allel'_ package, where development efforts are now focused.

- Documentation: http://anhima.readthedocs.org
- Examples: http://nbviewer.ipython.org/github/alimanfoo/anhima/tree/master/examples/
- Source: http://github.com/alimanfoo/anhima
- Mailing list: https://groups.google.com/forum/#!forum/anhima
- Release notes: https://github.com/alimanfoo/anhima/releases

Installation

Install latest stable release from PyPI:

```
pip install -U anhima
```

Install from GitHub:

```
git clone https://github.com/alimanfoo/anhima.git
cd anhima
python setup.py install
```

Contents

CHAPTER 1

Locating samples and variants

Utilities for locating samples and variants.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/loc.ipynb

anhima.loc.**view_sample**(*a*, *selection*, *all_samples=None*)

View a single column from the array *a* corresponding to a selected sample.

Parameters **a** : array_like

An array with 2 or more dimensions, where the second dimension corresponds to samples.

selection : int or object

A sample identifier or column index.

all_samples : sequence, optional

A sequence (e.g., list) of sample identifiers corresponding to the second dimension of *a*, used to map *selection* to a column index. If not given, assume *selection* is a column index.

Returns **b** : ndarray

An array obtained from *a* by taking the column corresponding to the selected sample.

anhima.loc.**take_samples**(*a*, *selection*, *all_samples=None*)

Extract columns from the array *a* corresponding to selected samples.

Parameters **a** : array_like

An array with 2 or more dimensions, where the second dimension corresponds to samples.

selection : sequence of ints or objects

A sequence of sample identifiers or column indices.

all_samples : sequence, optional

A sequence (e.g., list) of sample identifiers corresponding to the second dimension of *a*, used to map *selection* to column indices. If not given, assume *selection* is a sequence of column indices.

Returns **b** : ndarray

An array obtained from *a* by taking columns corresponding to the selected samples.

anhima.loc.**query_variants**(*expression*, *variants*)
Evaluate *expression* with respect to the given *variants*.

Parameters **expression** : string

The query expression to apply. The expression will be evaluated by `numexpr` against the provided *variants*.

variants : dict-like

The variables to include in scope for the expression evaluation.

Returns **result** : ndarray

The result of evaluating *expression* against *variants*.

anhima.loc.**compress_variants**(*a*, *condition*)
Extract rows from the array *a* corresponding to a boolean *condition*.

Parameters **a** : array_like

An array to extract rows from (e.g., genotypes).

condition : array_like, bool

A 1-D boolean array of the same length as the first dimension of *a*.

Returns **b** : ndarray

An array obtained from *a* by taking rows corresponding to the selected variants.

See also:

take_variants, `numpy.compress`

anhima.loc.**take_variants**(*a*, *indices*, *mode='raise'*)
Extract rows from the array *a* corresponding to *indices*.

Parameters **a** : array_like

An array to extract rows from (e.g., genotypes).

indices : sequence of integers

The variant indices to extract.

mode : {'raise', 'wrap', 'clip'}, optional

Specifies how out-of-bounds indices will behave.

Returns **b** : ndarray

An array obtained from *a* by taking rows corresponding to the selected variants.

See also:

compress_variants, `numpy.take`

anhima.loc.**locate_position**(*pos*, *p*)
Locate the index of coordinate *p* within sorted array of genomic positions *pos*.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig, with no duplicates.

p : int

The position to locate.

Returns **index** : int or None

The index of *p* in *pos* if present, else None.

See also:

locate_positions, *locate_interval*, *locate_intervals*

anhima.loc.**view_position**(*a*, *pos*, *p*)

View a slice along the first dimension of *a* corresponding to a genome position.

Parameters **a** : array_like

The array to extract from.

pos : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig, with no duplicates.

p : int

The position to locate.

Returns **b** : ndarray

A view of *a* obtained by slicing along the first dimension.

See also:

locate_position

anhima.loc.**locate_interval**(*pos*, *start_position=0*, *stop_position=None*)

Locate the start and stop indices within the *pos* array that include all positions within the *start_position* and *stop_position* range.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

start_position : int

Start position of interval.

stop_position : int

Stop position of interval

Returns **loc** : slice

A slice object with the start and stop indices that include all positions within the interval.

See also:

locate_position, *locate_positions*, *locate_intervals*

anhima.loc.**view_interval**(*a*, *pos*, *start_position*, *stop_position*)

View a contiguous slice along the first dimension of *a* corresponding to a genome interval defined by *start_position* and *stop_position*.

Parameters **a** : array_like

The array to extract from.

pos : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

start_position : int

Start position of interval.

stop_position : int

Stop position of interval

Returns **b** : ndarray

A view of *a* obtained by slicing along the first dimension.

See also:

locate_interval

anhima.loc.**locate_positions**(*pos1*, *pos2*)

Find the intersection of two sets of positions.

Parameters **pos1, pos2** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig, with no duplicates.

Returns **cond1** : ndarray, bool

An array of the same length as *pos1* where an element is True if the corresponding item in *pos1* is also found in *pos2*.

cond2 : ndarray, bool

An array of the same length as *pos2* where an element is True if the corresponding item in *pos2* is also found in *pos1*.

See also:

locate_position, *locate_interval*, *locate_intervals*

anhima.loc.**locate_intervals**(*pos*, *start_positions*, *stop_positions*)

Locate items within the *pos* array that fall within any of the intervals given by *start_positions* and *stop_positions*.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

start_positions : array_like, int

Start positions of intervals.

stop_positions : array_like, int

Stop positions of intervals

Returns **cond1** : ndarray, bool

An array of the same length as *pos* where an element is True if the corresponding item in *pos* is also found in any of the intervals.

cond2 : ndarray, bool

An array of the same length as the number of intervals, where an element is True if the corresponding interval contains one or more positions in *pos*.

See also:

locate_position, *locate_positions*, *locate_interval*

anhima.loc.**plot_variant_locator**(*pos*, *step=1*, *ax=None*, *start_position=None*, *stop_position=None*, *flip=False*, *line_args=None*)

Plot lines indicating the physical genome location of variants. By default the top x axis is in variant index space, and the bottom x axis is in genome position space.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

step : int, optional

Plot a line for every *step* variants.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

flip : bool, optional

Flip the plot upside down.

line_args : dict-like

Additional keyword arguments passed through to *plt.Line2D*.

Returns **ax** : axes

The axes on which the plot was drawn

`anhima.loc.`**`windowed_variant_counts`**(*pos*, *window_size*, *start_position=None*, *stop_position=None*)

Count variants in non-overlapping windows over the genome.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

Returns **counts** : ndarray, int

The number of variants in each window.

bin_edges : ndarray, int

The edge positions of each window. Note that this has length `len(counts)+1`. To determine bin centers use `(bin_edges[:-1] + bin_edges[1:]) / 2`. To determine bin widths use `np.diff(bin_edges)`.

See also:

`windowed_variant_counts_plot`, *windowed_variant_density*

`anhima.loc.`**`plot_windowed_variant_counts`**(*pos*, *window_size*, *start_position=None*, *stop_position=None*, *ax=None*, *plot_kwargs=None*)

Plot windowed variant counts.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

plot_kwargs : dict-like

Additional keyword arguments passed through to *plt.plot*.

Returns **ax** : axes

The axes on which the plot was drawn.

See also:

windowed_variant_counts, windowed_variant_density_plot

anhima.loc.**windowed_variant_density**(*pos*, *window_size*, *start_position=None*, *stop_position=None*)

Calculate per-base-pair density of variants in non-overlapping windows over the genome.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

Returns **density** : ndarray, int

The density of variants in each window.

bin_edges : ndarray, int

The edge positions of each window. Note that this has length `len(density)+1`. To determine bin centers use `(bin_edges[:-1] + bin_edges[1:]) / 2`. To determine bin widths use `np.diff(bin_edges)`.

See also:

windowed_variant_density_plot, *windowed_variant_counts*

anhima.loc.**plot_windowed_variant_density**(*pos*, *window_size*, *start_position=None*, *stop_position=None*, *ax=None*, *plot_kwargs=None*)

Plot windowed variant density.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

plot_kwargs : dict-like

Additional keyword arguments passed through to *plt.plot*.

Returns **ax** : axes

The axes on which the plot was drawn.

See also:

windowed_variant_density, windowed_variant_counts_plot

anhima.loc.**windowed_statistic**(*pos*, *values*, *window_size*, *start_position=None*, *stop_position=None*, *statistic='mean'*)

Calculate a statistic for *values* binned in non-overlapping windows over the genome.

Parameters **pos** : array_like

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

values : array_like

A 1-D array of the same length as *pos*.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

statistic : string or function

The function to apply to values in each bin.

Returns **stats** : ndarray

The values of the statistic within each bin.

bin_edges : ndarray

The edge positions of each window. Note that this has length `len(stats)+1`. To determine bin centers use `(bin_edges[:-1] + bin_edges[1:]) / 2`. To determine bin widths use `np.diff(bin_edges)`.

anhima.loc.**evenly_downsample_variants**(*a*, *k*)

Evenly downsample an array along the first dimension to length *k* (or as near as possible), assuming the first dimension corresponds to variants.

Parameters **a** : array_like

The array to downsample.

k : int

The target number of variants.

Returns **b** : array_like

A downsampled view of *a*.

anhima.loc.**randomly_downsample_variants**(*a*, *k*)

Evenly downsample an array along the first dimension to length *k*, assuming the first dimension corresponds to variants.

Parameters **a** : array_like

The array to downsample.

k : int

The k number of variants.

Returns **b** : array_like

A downsampled copy of *a*.

CHAPTER 2

Genotypes

Utility functions for working with genotype data.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/gt.ipynb

anhima.gt.**is_called**(*genotypes*)

Find non-missing genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_called** : ndarray, bool

An array where elements are True if the genotype call is non-missing.

See also:

is_missing, *is_hom_ref*, *is_het*, *is_hom_alt*

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.gt.**is_missing**(*genotypes*)

Find missing genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns is_missing: ndarray, bool

An array where elements are True if the genotype call is missing.

See also:

is_called, *is_hom_ref*, *is_het*, *is_hom_alt*

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.gt.**is_hom**(*genotypes*)
Find homozygous genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_hom** : ndarray, bool

An array where elements are True if the genotype call is homozygous.

See also:

is_called, *is_missing*, *is_hom_ref*, *is_hom_alt*

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.gt.**is_het**(*genotypes*)
Find heterozygous genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_het** : ndarray, bool

An array where elements are True if the genotype call is heterozygous.

See also:

is_called, *is_missing*, *is_hom_ref*, *is_hom_alt*

Notes

Applicable to polyploid genotype calls, although note that all types of heterozygous genotype (i.e., anything not completely homozygous) will give an element value of True.

Applicable to multiallelic variants, although note that the element value will be True in any case where the two alleles in a genotype are different, e.g., (0, 1), (0, 2), (1, 2), etc.

anhima.gt.**is_hom_ref**(*genotypes*)
Find homozygous reference genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_hom_ref** : ndarray, bool

An array where elements are True if the genotype call is homozygous reference.

See also:

is_called, *is_missing*, *is_het*, *is_hom_alt*

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.gt.**is_hom_alt**(*genotypes*)

Find homozygous non-reference genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_hom_alt** : ndarray, bool

An array where elements are True if the genotype call is homozygous non-reference.

See also:

is_called, *is_missing*, *is_hom_ref*, *is_het*

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.gt.**count_called**(*genotypes*, *axis=None*)

Count non-missing genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the number of called (i.e., non-missing) genotypes. If *axis* is specified, returns the sum along the given *axis*.

See also:

is_called

anhima.gt.**count_missing**(*genotypes*, *axis=None*)

Count non-missing genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the number of missing genotypes. If *axis* is specified, returns the sum along the given *axis*.

See also:

is_missing

anhima.gt.**count_hom**(*genotypes*, *axis=None*)

Count homozygous genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the number of homozygous genotypes. If *axis* is specified, returns the sum along the given *axis*.

See also:

is_hom

anhima.gt.**count_het**(*genotypes*, *axis=None*)

Count heterozygous genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the number of heterozygous genotypes. If *axis* is specified, returns the sum along the given *axis*.

See also:

is_het

anhima.gt.**count_hom_ref**(*genotypes*, *axis=None*)
Count homozygous reference genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the number of homozygous reference genotypes. If *axis* is specified, returns the sum along the given *axis*.

See also:

is_hom_ref

anhima.gt.**count_hom_alt**(*genotypes*, *axis=None*)
Count homozygous non-reference genotype calls.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the number of homozygous non-reference genotypes. If *axis* is specified, returns the sum along the given *axis*.

See also:

is_hom_alt

anhima.gt.**max_allele**(*genotypes*, *axis=None*)
Return the highest allele index.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to determine the maximum (0 = variants, 1 = samples). If not given, return the highest overall.

Returns **n** : int

The value of the highest allele index present in the genotypes array.

anhima.gt.**as_haplotypes**(*genotypes*)

Reshape an array of genotypes to view it as haplotypes by dropping the ploidy dimension.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **haplotypes** : ndarray

An array of shape (n_variants, n_samples * ploidy).

Notes

Note that if genotype calls are unphased, the haplotypes returned by this function will bear no resemblance to the true haplotypes.

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.gt.**as_n_alt**(*genotypes*)

Transform genotypes as the number of non-reference alleles.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **gn** : ndarray, uint8

An array where each genotype is coded as a single integer counting the number of alternate alleles.

See also:

as_012

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants, although this function simply counts the number of non-reference alleles, it makes no distinction between different non-reference alleles.

Note that this function returns 0 for missing genotype calls **and** for homozygous reference genotype calls, because in both cases the number of non-reference alleles is zero.

anhima.gt.**as_012**(*genotypes*, *fill=-1*)

Transform genotypes recoding homozygous reference calls a 0, heterozygous calls as 1, homozygous non-reference calls as 2, and missing calls as -1.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

fill : int, optional

Default value for missing calls.

Returns **gn** : ndarray, int8

An array where each genotype is coded as a single integer as described above.

See also:

`as_nalt`

Notes

Applicable to polyploid genotype calls, although note that all types of heterozygous genotype (i.e., anything not completely homozygous) will be coded as 1.

Applicable to multiallelic variants, although note the following. All heterozygous genotypes, e.g., (0, 1), (0, 2), (1, 2), ..., will be coded as 1. All homozygous non-reference genotypes, e.g., (1, 1), (2, 2), ..., will be coded as 2.

anhima.gt.**as_allele_counts**(*genotypes*, *alleles=None*)

Transform genotypes into allele counts per call.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

alleles : sequence of ints, optional

The alleles to count. If not specified, all alleles will be counted.

Returns **gac** : ndarray, uint8

An array where the ploidy dimension has been replaced by counts of each allele.

anhima.gt.**pack_diploid**(*genotypes*)

Pack diploid genotypes into a single byte for each genotype, using the left-most 4 bits for the first allele and the right-most 4 bits for the second allele. Allows single byte encoding of diploid genotypes for variants with up to 15 alleles.

Parameters **genotypes** : array_like, int

An array of shape (n_variants, n_samples, ploidy) or (n_variants, ploidy) or (n_samples, ploidy), where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **packed** : ndarray, int8

An array of genotypes where the ploidy dimension has been collapsed by bit packing the two alleles for each genotype into a single byte.

See also:

`unpack_diploid_genotypes`

anhima.gt.**unpack_diploid**(*packed*)

Unpack an array of diploid genotypes that have been bit packed into single bytes.

Parameters **packed** : array_like

An array of genotypes where the ploidy dimension has been collapsed by bit packing the two alleles for each genotype into a single byte.

Returns **genotypes** : ndarray, int8

An array of genotypes where the ploidy dimension has been restored by unpacking the input array.

See also:

pack_diploid_genotypes

anhima.gt.**count_genotypes**(*gn*, *t*, *axis=None*)

Count genotypes of a given type.

Parameters **gn** : array_like, int

An array of shape (n_variants, n_samples) or (n_variants,) or (n_samples,) where each element is a genotype called coded as a single integer.

t : int

The genotype to count.

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the total number of matching genotypes. If *axis* is specified, returns the sum along the given *axis*.

anhima.gt.**windowed_genotype_counts**(*pos*, *gn*, *t*, *window_size*, *start_position=None*, *stop_position=None*)

Count genotype calls of a given type for a single sample in non-overlapping windows over the genome.

Parameters **pos** : array_like, int

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

gn : array_like, int

A 1-D array of genotypes for a single sample, where each genotype is coded as a single integer.

t : int

The genotype to count.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

Returns **counts** : ndarray, int

Genotype counts for each window.

bin_edges : ndarray, float

The edges of the windows.

See also:

as_diploid_012, *as_n_alt*, *windowed_genotype_density*, *windowed_genotype_rate*

anhima.gt.**windowed_genotype_density**(*pos*, *gn*, *t*, *window_size*, *start_position=None*, *stop_position=None*)

Compute per-base-pair density of genotype calls of a given type for a single sample in non-overlapping windows over the genome.

Parameters **pos** : array_like, int

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

gn : array_like, int

A 1-D array of genotypes for a single sample, where each genotype is coded as a single integer.

t : int

The genotype to count.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

Returns **density** : ndarray, float

Genotype density for each window.

bin_edges : ndarray, float

The edges of the windows.

See also:

as_diploid_012, *as_n_alt*, *windowed_genotype_counts*, *windowed_genotype_rate*

anhima.gt.**windowed_genotype_rate**(*pos*, *gn*, *t*, *window_size*, *start_position=None*, *stop_position=None*)

Compute per-variant rate of genotype calls of a given type for a single sample in non-overlapping windows over the genome.

Parameters **pos** : array_like, int

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

gn : array_like, int

A 1-D array of genotypes for a single sample, where each genotype is coded as a single integer.

t : int

The genotype to count.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

Returns **rate** : ndarray, float

Per-variant rate for each window.

bin_edges : ndarray, float

The edges of the windows.

See also:

`as_diploid_012`, *as_n_alt*, *windowed_genotype_counts*, *windowed_genotype_density*

anhima.gt.**plot_windowed_genotype_counts**(*pos*, *gn*, *t*, *window_size*, *start_position=None*, *stop_position=None*, *ax=None*, *plot_kwargs=None*)

Plots counts of genotype calls of a given type for a single sample in non-overlapping windows over the genome.

Parameters **pos** : array_like, int

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

gn : array_like, int

A 1-D array of genotypes for a single sample, where each genotype is coded as a single integer.

t : int

The genotype to count.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

plot_kwargs : dict-like

Additional keyword arguments passed through to *plt.plot*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_windowed_genotype_density**(*pos*, *gn*, *t*, *window_size*, *start_position=None*, *stop_position=None*, *ax=None*, *plot_kwargs=None*)

Plots per-base-pair density of genotype calls of a given type for a single sample in non-overlapping windows over the genome.

Parameters **pos** : array_like, int

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

gn : array_like, int

A 1-D array of genotypes for a single sample, where each genotype is coded as a single integer.

t : int

The genotype to count.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

plot_kwargs : dict-like

Additional keyword arguments passed through to *plt.plot*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_windowed_genotype_rate**(*pos*, *gn*, *t*, *window_size*, *start_position=None*, *stop_position=None*, *ax=None*, *plot_kwargs=None*)

Plots per-variant rate of genotype calls of a given type for a single sample in non-overlapping windows over the genome.

Parameters **pos** : array_like, int

A sorted 1-dimensional array of genomic positions from a single chromosome/contig.

gn : array_like, int

A 1-D array of genotypes for a single sample, where each genotype is coded as a single integer.

t : int

The genotype to count.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

plot_kwargs : dict-like

Additional keyword arguments passed through to *plt.plot*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_discrete_calldata**(*a*, *labels=None*, *colors='wbgrcmyk'*, *states=None*, *ax=None*, *pcolormesh_kwargs=None*)

Plot a color grid from discrete calldata (e.g., genotypes).

Parameters **a** : array_like, int, shape (n_variants, n_samples)

2-dimensional array of integers containing the call data to plot.

labels : sequence of strings, optional

Axis labels (e.g., sample IDs).

colors : sequence, optional

Colors to use for different values of the array.

states : sequence, optional

Manually specify discrete calldata states (if not given will be determined from the data).

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

pcolormesh_kwargs : dict-like, optional

Additional keyword arguments passed through to *plt.pcolormesh*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_continuous_calldata**(*a*, *labels=None*, *ax=None*, *pcolormesh_kwargs=None*)

Plot a color grid from continuous calldata (e.g., DP).

Parameters **a** : array_like, shape (n_variants, n_samples)

2-dimensional array of integers or floats containing the call data to plot.

labels : sequence of strings, optional

Axis labels (e.g., sample IDs).

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

pcolormesh_kwargs : dict-like, optional

Additional keyword arguments passed through to *plt.pcolormesh*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_diploid_genotypes**(*gn*, *labels=None*, *colors='wbgr'*, *states=(-1, 0, 1, 2)*, *ax=None*, *colormesh_kwargs=None*)

Plot diploid genotypes as a color grid.

Parameters **gn** : array_like, int, shape (n_variants, n_samples)

An array where each genotype is coded as a single integer as described above.

labels : sequence of strings, optional

Axis labels (e.g., sample IDs).

colors : sequence, optional

Colors to use for different values of the array.

states : sequence, optional

Manually specify discrete calldata states (if not given will be determined from the data).

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

colormesh_kwargs : dict-like

Additional keyword arguments passed through to *plt.pcolormesh*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_genotype_counts_by_sample**(*gn*, *states=(-1, 0, 1, 2)*, *colors='wbgr'*, *labels=None*, *ax=None*, *width=1*, *orientation='vertical'*, *bar_kwargs=None*)

Plot a bar graph of genotype counts by sample.

Parameters **gn** : array_like, int, shape (n_variants, n_samples)

An array where each genotype is coded as a single integer as described above.

states : sequence, optional

The genotype states to count.

colors : sequence, optional

Colors to use for corresponding states.

labels : sequence of strings, optional

Axis labels (e.g., sample IDs).

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

width : float, optional

Width of the bars (will be used as height if *orientation* == 'horizontal').

orientation : {'horizontal', 'vertical'}

Which type of bar to plot.

bar_kwargs : dict-like

Additional keyword arguments passed through to *plt.bar*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_genotype_counts_by_variant**(*gn*, *states=(-1, 0, 1, 2)*, *colors='wbgr'*, *ax=None*, *width=1*, *orientation='vertical'*, *bar_kwargs=None*)

Plot a bar graph of genotype counts by variant.

Parameters **gn** : array_like, int, shape (n_variants, n_samples)

An array where each genotype is coded as a single integer as described above.

states : sequence, optional

The genotype states to count.

colors : sequence, optional

Colors to use for corresponding states.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

width : float, optional

Width of the bars (will be used as height if *orientation* == 'horizontal').

orientation : {'horizontal', 'vertical'}

Which type of bar to plot.

bar_kwargs : dict-like

Additional keyword arguments passed through to *plt.bar*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.gt.**plot_continuous_calldata_by_sample**(*a*, *labels=None*, *ax=None*, *orientation='vertical'*, *boxplot_kwargs=None*)

Plot a boxplot of continuous call data (e.g., DP) by sample.

Parameters **a** : array_like, shape (n_variants, n_samples)

2-dimensional array of integers or floats containing the call data to plot.

labels : sequence of strings, optional

Axis labels (e.g., sample IDs).

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

orientation : {'horizontal', 'vertical'}

Which type of bar to plot.

boxplot_kwargs : dict-like

Additional keyword arguments passed through to *plt.boxplot*.

Returns **ax** : axes

The axes on which the plot was drawn.

CHAPTER 3

Allele frequencies

Allele frequency calculations.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/af.ipynb

anhima.af.**allelism**(*genotypes*)

Determine the number of distinct alleles found for each variant.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **n** : ndarray, int

An array of shape (n_variants,) where an element holds the allelism of the corresponding variant.

See also:

`max_allele`

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**allele_number**(*genotypes*)

Count the number of non-missing allele calls per variant.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **an** : ndarray, int

An array of shape (n_variants,) counting the total number of non-missing alleles observed.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**allele_count**(*genotypes*, *allele=1*)

Calculate number of observations of the given allele per variant.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

The allele to count.

Returns **ac** : ndarray, int

An array of shape (n_variants,) counting the number of times the given *allele* was observed.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants, but note that this function calculates the frequency of a specific *allele*.

anhima.af.**allele_frequency**(*genotypes*, *allele=1*)

Calculate frequency of the given allele per variant.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

The allele to calculate the frequency of.

Returns **an** : ndarray, int

An array of shape (n_variants,) counting the total number of non-missing alleles observed.

ac : ndarray, int

An array of shape (n_variants,) counting the number of times the given *allele* was observed.

af : ndarray, float

An array of shape (n_variants,) containing the allele frequency.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants, but note that this function calculates the frequency of a specific *allele*.

anhima.af.**allele_counts**(*genotypes*, *alleles=None*)
Calculate allele counts per variant.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

alleles : sequence of ints, optional

The alleles to count. If not specified, all alleles will be counted.

Returns **ac** : ndarray, int

An array of shape (n_variants, n_alleles) counting the number of times the given *alleles* were observed.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**allele_frequencies**(*genotypes*, *alleles=None*)
Calculate allele frequencies per variant.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

alleles : sequence of ints, optional

The alleles to calculate the frequency of. If not specified, all alleles will be counted.

Returns **an** : ndarray, int

An array of shape (n_variants,) counting the total number of non-missing alleles observed.

ac : ndarray, int

An array of shape (n_variants, n_alleles) counting the number of times the given *alleles* were observed.

af : ndarray, float

An array of shape (n_variants, n_alleles) containing the allele frequencies.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**is_variant**(*genotypes*)
Find variants with at least one non-reference allele observation.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_variant** : ndarray, bool

An array of shape (n_variants,) where an element is True if there are at least *min_ac* non-reference alleles found for the corresponding variant.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**is_non_variant**(*genotypes*)

Find variants with no non-reference alleles.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_non_variant** : ndarray, bool

An array of shape (n_variants,) where an element is True if there are no non-reference alleles found for the corresponding variant.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**is_segregating**(*genotypes*)

Find segregating variants (where more than one allele is observed).

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **is_segregating** : ndarray, bool

An array of shape (n_variants,) where an element is True if more than one allele is found for the given variant.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**is_non_segregating**(*genotypes*, *allele=None*)

Find non-segregating variants (fixed for a single allele).

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

If given, find variants fixed with respect to *allele*. Otherwise find variants fixed for any allele.

Returns **is_non_segregating** : ndarray, bool

An array of shape (n_variants,) where an element is True if all genotype calls for the corresponding variant are either missing or equal to the same allele.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**is_singleton**(*genotypes*, *allele=1*)

Find variants with only a single instance of *allele* observed.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

The allele to find singletons of.

Returns **is_singleton** : ndarray, bool

An array of shape (n_variants,) where an element is True if there is a single instance of *allele* observed.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants, but note this function checks for a specific *allele*.

anhima.af.**is_doubleton**(*genotypes*, *allele=1*)

Find variants with only two instances of *allele* observed.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

The allele to find doubletons of.

Returns **is_doubleton** : ndarray, bool

An array of shape (n_variants,) where an element is True if there are exactly two instances of *allele* observed.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants, but note this function checks for a specific *allele*.

anhima.af.**count_variant**(*genotypes*)
Count variants with at least one non-reference allele observed.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

min_ac : int, optional

The minimum number of non-reference alleles required to consider variant.

Returns **n** : int

The number of variants.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**count_non_variant**(*genotypes*)
Count variants with no non-reference alleles.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **n** : int

The number of variants.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**count_segregating**(*genotypes*)
Count segregating variants.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **n** : int

The number of variants.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**count_non_segregating**(*genotypes*, *allele=None*)
Count non-segregating variants.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

If given, find variants fixed with respect to *allele*.

Returns **n** : int

The number of variants.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants.

anhima.af.**count_singletons**(*genotypes*, *allele=1*)
Count variants with only a single instance of *allele* observed.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

The allele to find singletons of.

Returns **n** : int

The number of variants.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants, but note this function checks for a specific *allele*.

anhima.af.**count_doubletons**(*genotypes*, *allele=1*)
Count variants with only two instances of *allele* observed.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

allele : int, optional

The allele to find doubletons of.

Returns **n** : int

The number of variants.

Notes

Applicable to polyploid genotype calls.

Applicable to multiallelic variants, but note this function checks for a specific *allele*.

anhima.af.**maximum_likelihood_ancestry**(*genotypes*, *qa*, *qb*, *filter_size=0*)

Given alternate allele frequencies in two populations *qa* and *qb*, predict the ancestry for a set of *genotypes*.

Parameters **genotypes** : array_like

An array of diploid genotype calls of shape (n_variants, n_samples, 2) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

qa : array_like, float

A 1-dimensional array of shape (n_variants,) containing alternate allele frequencies for population A.

qb : array_like, float

A 1-dimensional array of shape (n_variants,) containing alternate allele frequencies for population B.

filter_size : int, optional

Sum likelihoods in a moving window of size *filter_size*.

Returns **ancestry** : ndarray, int, shape (n_variants, n_samples)

An array containing the ancestry predictions, where 0 = AA (both alleles derive from population A), 1 = AB (hybrid ancestry) and 2 = BB (both alleles derive from population B), and -1 = ambiguous (models are equally likely).

confidence : ndarray, float, shape (n_variants, n_samples)

The confidence in the ancestry prediction (natural logarithm of the likelihood ratio for the two most likely models).

Notes

Where allele frequencies are similar between populations A and B, ancestry predictions will have low confidence, because different ancestry models will have similar likelihoods. Greater confidence will be obtained by filtering variants to select those where the difference in allele frequencies is greater. E.g.:

```
>>> flt = np.abs(qa - qb) > .5
>>> genotypes_flt = genotypes[flt]
>>> qa_flt = qa[flt]
>>> qb_flt = qb[flt]
>>> ancestry, confidence = maximum_likelihood_ancestry(genotypes_flt, qa_flt, qb_flt)
```

CHAPTER 4

Site frequencies

Site frequency spectra.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/sf.ipynb

anhima.sf.**site_frequency_spectrum**(*derived_ac*)

Calculate the site frequency spectrum, given derived allele counts for a set of biallelic variant sites.

Parameters **derived_ac** : array_like, int

A 1-dimensional array of shape (n_variants,) where each array element holds the count of derived alleles found for a single variant across some set of samples.

Returns **sfs** : ndarray, int

An array of integers where the value of the kth element is the number of variant sites with k derived alleles.

See also:

site_frequency_spectrum_scaled, *site_frequency_spectrum_folded*, *site_frequency_spectrum_folded_scaled*, *plot_site_frequency_spectrum*

anhima.sf.**site_frequency_spectrum_scaled**(*derived_ac*)

Calculate the site frequency spectrum, scaled such that a constant value is expected across the spectrum for neutral variation and a population at constant size.

Parameters **derived_ac** : array_like, int

A 1-dimensional array of shape (n_variants,) where each array element holds the count of derived alleles found for a single variant across some set of samples.

Returns **sfs_scaled** : ndarray, int

An array of integers where the value of the kth element is the number of variant sites with k derived alleles, multiplied by k.

See also:

site_frequency_spectrum, *site_frequency_spectrum_folded*, *site_frequency_spectrum_folded_scaled*, *plot_site_frequency_spectrum*

Notes

Under neutrality and constant population size, site frequency is expected to be constant across the spectrum, and to approximate the value of the population-scaled mutation rate theta.

anhima.sf.**site_frequency_spectrum_folded**(*biallelic_ac*)

Calculate the folded site frequency spectrum, given reference and alternate allele counts for a set of biallelic variants.

Parameters **biallelic_ac** : array_like int

A 2-dimensional array of shape (n_variants, 2), where each row holds the reference and alternate allele counts for a single biallelic variant across some set of samples.

Returns **sfs_folded** : ndarray, int

An array of integers where the value of the kth element is the number of variant sites with k observations of the minor allele.

See also:

site_frequency_spectrum, *site_frequency_spectrum_scaled*, *site_frequency_spectrum_folded_scaled*, *plot_site_frequency_spectrum*

anhima.sf.**site_frequency_spectrum_folded_scaled**(*biallelic_ac*, *m=None*)

Calculate the folded site frequency spectrum, scaled such that a constant value is expected across the spectrum for neutral variation and a population at constant size.

Parameters **biallelic_ac** : array_like int

A 2-dimensional array of shape (n_variants, 2), where each row holds the reference and alternate allele counts for a single biallelic variant across some set of samples.

m : int, optional

The total number of alleles observed at each variant site. Equal to the number of samples multiplied by the ploidy. If not provided, will be inferred to be the maximum value of the sum of reference and alternate allele counts present in *biallelic_ac*.

Returns **sfs_folded_scaled** : ndarray, int

An array of integers where the value of the kth element is the number of variant sites with k observations of the minor allele, multiplied by the scaling factor (k * (m - k) / m).

See also:

site_frequency_spectrum, *site_frequency_spectrum_scaled*, *site_frequency_spectrum_folded*, *plot_site_frequency_spectrum*

Notes

Under neutrality and constant population size, site frequency is expected to be constant across the spectrum, and to approximate the value of the population-scaled mutation rate theta.

This function is useful where the ancestral and derived status of alleles is unknown.

anhima.sf.**plot_site_frequency_spectrum**(*sfs*, *bins=None*, *m=None*, *clip_endpoints=True*, *ax=None*, *label=None*, *plot_kwargs=None*)

Plot a site frequency spectrum.

Parameters **sfs** : array_like, int

Site frequency spectrum. Can be folded or unfolded, scaled or unscaled.

bins : int or sequence of ints, optional

Number of bins or bin edges to aggregate frequencies. If not given, no binning will be applied.

m : int, optional

The total number of alleles observed at each variant site. Equal to the number of samples multiplied by the ploidy. If given, will be used to scale the X axis as allele frequency instead of allele count. used to scale the X axis as allele frequency instead of allele count.

clip_endpoints : bool, optional

If True, remove the first and last values from the site frequency spectrum.

ax : axes, optional

The axes on which to plot. If not given, a new figure will be created.

label : string, optional

Label for this data series.

plot_kwargs : dict, optional

Passed through to ax.plot().

Returns **ax** : axes

The axes on which the plot was drawn.

See also:

site_frequency_spectrum, *site_frequency_spectrum_folded*, *site_frequency_spectrum_scaled*, *site_frequency_spectrum_folded_scaled*

CHAPTER 5

Doubleton sharing

Doubleton sharing, a.k.a., analysis of f2 variants.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/f2.ipynb

anhima.f2.**count_shared_doubletons**(*subpops_ac*)

Count subpopulation pairs sharing doubletons (where one allele is observed in each subpopulation).

Parameters **subpops_ac** : array_like, int

An array of shape (n_variants, n_subpops) holding alternate allele counts for each sub-population.

Returns **counts** : ndarray, int or float

A square matrix of shape (n_subpops, n_subpops) where the array element at index (i, j) holds the count of shared doubletons between the ith and jth subpopulations.

See also:

normalise_doubleton_counts, *plot_shared_doubletons*, *plot_total_doubletons*, *plot_f2_fig*

anhima.f2.**normalise_doubleton_counts**(*counts*, *n_samples*, *ploidy=2*)

Normalise doubleton counts by dividing by the number of distinct pairs of haplotypes in each population comparison.

Parameters **counts** : array_like, ints

A square matrix of shape (n_subpops, n_subpops) where the array element at index (i, j) holds the count of shared doubletons between the ith and jth subpopulations.

n_samples : int or sequence of ints

The number of samples in each sub-population.

ploidy : int, optional

The sample ploidy.

Returns **normed_counts** : ndarray, float

Normalised counts of shared doubletons.

See also:

count_shared_doubletons

Notes

This function corrects for the fact that there are fewer pairs of haplotypes when looking for doubletons within a single subpopulation of size n than there are when comparing two different subpopulations of size n.

This function may also help to correct for the case where the number of samples from each subpopulation is not equal. However, note that if this is the case then there may still also be some bias in how doubletons have been ascertained.

anhima.f2.**plot_shared_doubletons**(*counts*, *subpop_labels=None*, *subpop_colors='bgrcmyk'*, *axs=None*, *figsize_factor=1*, *ylim=None*, *relative=False*, *flip=False*)

Plot counts of doubleton sharing between subpopulations as a bar chart.

Parameters **counts** : array_like, ints

A square matrix of shape (n_subpops, n_subpops) where the array element at index (i, j) holds the count of shared doubletons between the ith and jth subpopulations.

subpop_labels : sequence of strings, optional

Labels for the subpopulations.

subpop_colors : sequence of colors, optional

Colors for the subpopulations.

axs : sequence of axes, optional

The axes to use. If not provided, a new figure will be created.

figsize_factor : float, optional

Figure size in inches per subpopulation. Only used if *axs* is None.

ylim : pair of ints or floats, optional

Limits for the Y axes of all subplots.

relative : bool, optional

If True, normalise counts by dividing by the sum along each row.

flip : bool, optional

If True, invert the Y axis.

Returns **axs** : sequence of axes

The axes on which the plot was drawn.

See also:

count_shared_doubletons, *plot_total_doubletons*, *plot_f2_fig*

anhima.f2.**plot_total_doubletons**(*counts*, *subpop_labels=None*, *width=0.8*, *orientation='vertical'*, *n_samples=None*, *ax=None*, *bar_kwargs=None*)

Plot total counts of doubletons per subpopulations as a bar chart.

Parameters **counts** : array_like, ints

A square matrix of shape (n_subpops, n_subpops) where the array element at index (i, j) holds the count of shared doubletons between the ith and jth subpopulations.

subpop_labels : sequence of strings, optional

Labels for the subpopulations.

width : float, optional

The relative width of each bar.

orientation : {'vertical', 'horizontal'}

The bar orientation.

n_samples : int or sequence of ints

The number of samples in each sub-population.

ax : axes, optional

The axes on which to plot. If not provided, a new figure will be created.

bar_kwargs : dict, optional

Keyword arguments passed through to ax.bar().

Returns **ax** : axes

The axes on which the plot was drawn.

See also:

count_shared_doubletons, *plot_shared_doubletons*, *plot_total_doubletons*, *plot_f2_fig*

anhima.f2.**plot_f2_fig**(*counts*, *subpop_labels=None*, *subpop_colors='bgrcmyk'*, *fig=None*, *figsize_factor=1*, *relative=False*, *normed=False*, *n_samples=None*, *ploidy=2*)

Plot a combined figure of shared doubleton counts and total counts per subpopulation.

Parameters **counts** : array_like, ints

A square matrix of shape (n_subpops, n_subpops) where the array element at index (i, j) holds the count of shared doubletons between the ith and jth subpopulations.

subpop_labels : sequence of strings, optional

Labels for the subpopulations.

subpop_colors : sequence of colors, optional

Colors for the subpopulations.

fig : figure, optional

The figure to use. If not provided, a new figure will be created.

figsize_factor : float, optional

Figure size in inches per subpopulation. Only used if *fig* is None.

relative : bool, optional

If True, plot counts relative to the sum along each row.

normed : bool, optional

If True, normalise counts by dividing by the number of possible pairs of haplotypes.

n_samples : int or sequence of ints

The number of samples in each sub-population.

ploidy : int, optional

The sample ploidy. (Only relevant if *normed* is True.)

Returns **fig** : figure

The figure on which the plot was drawn.

See also:

count_shared_doubletons, *plot_shared_doubletons*, *plot_total_doubletons*

CHAPTER 6

Linkage disequilibrium

Utilities for calculating and plotting linkage disequilbrium.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/ld.ipynb

anhima.ld.**pairwise_genotype_ld**(*gn*)

Given a set of genotypes at biallelic variants, calculate the square of the correlation coefficient between all distinct pairs of variants.

Parameters **gn** : array_like

A 2-dimensional array of shape (*n_variants*, *n_samples*) where each element is a genotype call coded as a single integer counting the number of non-reference alleles.

Returns **r_squared** : ndarray, float

A 2-dimensional array of squared correlation coefficients between each pair of variants.

anhima.ld.**plot_pairwise_ld**(*r_squared*, *cmap='Greys'*, *flip=True*, *ax=None*)

Make a classic triangular linkage disequilibrium plot, given an array of pairwise correlation coefficients between variants.

Parameters **r_squared** : array_like

A square 2-dimensional array of squared correlation coefficients between pairs of variants.

cmap : color map, optional

The color map to use when plotting. Defaults to 'Greys' (0=white, 1=black).

flip : bool, optional

If True, draw the triangle upside down.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

Returns **ax** : axes

The axes on which the plot was drawn

anhima.ld.**plot_windowed_ld**(*gn*, *pos*, *window_size*, *start_position=None*, *stop_position=None*, *percentiles=(5, 95)*, *ax=None*, *median_plot_kwargs=None*, *percentiles_plot_kwargs=None*)

Plot average LD within non-overlapping genome windows.

Parameters **gn** : array_like

A 2-dimensional array of shape (*n_variants*, *n_samples*) where each element is a genotype call coded as a single integer counting the number of non-reference alleles.

pos : array_like

A 1-dimensional array of genomic positions of variants.

window_size : int

The size in base-pairs of the windows.

start_position : int, optional

The start position for the region over which to work.

stop_position : int, optional

The stop position for the region over which to work.

percentiles : sequence of integers, optional

Percentiles to plot in addition to the median.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

median_plot_kwargs : dict, optional

Keyword arguments to pass through when plotting the median line.

percentiles_plot_kwargs : dict, optional

Keyword arguments to pass through when plotting the percentiles.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.ld.**pairwise_ld_decay**(*r_squared*, *pos*, *step=1*)

Compile data on linkage disequilibrium, separation (in number of variants), and physical distance between pairs of variants.

Parameters **r_squared** : array_like

A square 2-dimensional array of squared correlation coefficients between pairs of variants.

pos : array_like

A 1-dimensional array of genomic positions of variants.

step : int, optional

When compiling the data, advance *step* variants.

Returns **cor** : ndarray, float

Each element in the array is the squared genotype correlation coefficient between a distinct pair of variants.

sep : ndarray, int

Each element in the array is the separation (in number of variants) between a distinct pair of variants.

dist : ndarray, int

Each element in the array is the physical distance between a distinct pair of variants.

See also:

windowed_ld_decay

anhima.ld.**windowed_ld_decay**(*gn*, *pos*, *window_size*, *step=1*)

Compile data on linkage disequilibrium, separation (in number of variants), and physical distance between pairs of variants.

Parameters **gn** : array_like

A 2-dimensional array of shape (*n_variants*, *n_samples*) where each element is a genotype call coded as a single integer counting the number of non-reference alleles.

pos : array_like

A 1-dimensional array of genomic positions of variants.

window_size : int, optional

The number of variants to work with at a time.

step : int, optional

When compiling the data within each window, advance *step* variants.

Returns **cor** : ndarray, float

Each element in the array is the squared genotype correlation coefficient between a distinct pair of variants.

sep : ndarray, int

Each element in the array is the separation (in number of variants) between a distinct pair of variants.

dist : ndarray, int

Each element in the array is the physical distance between a distinct pair of variants.

See also:

pairwise_ld_decay

Notes

Similar to *pairwise_ld_decay()* except that not all pairs of variants are sampled to speed up computation and use less memory. Variants are divided into non-overlapping windows of size *window_size*. Genotype LD is calculated for all pairs within each window.

anhima.ld.**plot_ld_decay_by_separation**(*cor*, *sep*, *max_separation=100*, *percentiles=(5, 95)*, *ax=None*, *median_plot_kwargs=None*, *percentiles_plot_kwargs=None*)

Plot the decay of linkage disequilibrium with separation between variants.

Parameters **cor** : array_like

A 1-dimensional array of squared correlation coefficients between pairs of variants.

sep : array_like

A 1-dimensional array of separations (in number of variants) between pairs of variants.

max_separation : int, optional

Maximum separation to consider.

percentiles : sequence of integers, optional

Percentiles to plot in addition to the median.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

median_plot_kwargs : dict, optional

Keyword arguments to pass through when plotting the median line.

percentiles_plot_kwargs : dict, optional

Keyword arguments to pass through when plotting the percentiles.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.ld.**plot_ld_decay_by_distance**(*cor*, *dist*, *bins*, *percentiles=(5, 95)*, *ax=None*, *median_plot_kwargs=None*, *percentiles_plot_kwargs=None*)

Plot the decay of linkage disequilibrium with physical distance between variants.

Parameters **cor** : array_like

A 1-dimensional array of squared correlation coefficients between pairs of variants.

dist : array_like

A 1-dimensional array of physical distances between pairs of variants.

bins : int or sequence of ints

Number of bins or bin edges. Bins of distance to calculate LD within.

percentiles : sequence of integers, optional

Percentiles to plot in addition to the median.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

median_plot_kwargs : dict, optional

Keyword arguments to pass through when plotting the median line.

percentiles_plot_kwargs : dict, optional

Keyword arguments to pass through when plotting the percentiles.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.ld.**ld_prune_pairwise**(*gn*, *window_size=100*, *window_step=10*, *max_r_squared=0.2*)

Given a set of genotypes at biallelic variants, find a subset of the variants which are in approximate linkage equilibrium with each other.

Parameters **gn** : array_like

A 2-dimensional array of shape (*n_variants*, *n_samples*) where each element is a genotype call coded as a single integer counting the number of non-reference alleles.

window_size : int, optional

The number of variants to work with at a time.

window_step : int, optional

The number of variants to shift the window by.

max_r_squared : float, optional

The maximum value of the genotype correlation coefficient, above which variants will be excluded.

Returns **included** : ndarray, bool

A boolean array of the same length as the number of variants, where a True value indicates the variant at the corresponding index is included, and a False value indicates the corresponding variant is excluded.

Notes

The algorithm is as follows. A window of *window_size* variants is taken from the beginning of the genotypes array. The genotype correlation coefficient is calculated between each pair of variants in the window. The first variant in the window is considered, and any other variants in the window with linkage above *max_r_squared* with respect to the first variant is excluded. The next non-excluded variant in the window is then considered, and so on. The window then shifts along by *window_step* variants, and the process is repeated.

CHAPTER 7

Genetic distance

Genetic distance calculations.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/dist.ipynb

anhima.dist.**pairwise_distance**(*gn*, *metric='euclidean'*)

Compute pairwise distance between samples.

Parameters **gn** : array_like

A 2-dimensional array of shape (n_variants, n_samples) where each element is a genotype call coded as a single integer counting the number of non-reference alleles.

metric : string or function, optional

The distance metric to use. See documentation for the function `scipy.spatial.distance.pdist()` for a list of supported distance metrics.

Returns **dist** : ndarray, float

The distance matrix in compact form.

dist_square : ndarray, float

The distance matrix in square form.

anhima.dist.**plot_pairwise_distance**(*dist_square*, *labels=None*, *colorbar=True*, *ax=None*, *vmin=None*, *vmax=None*, *cmap='jet'*, *imshow_kwargs=None*)

Plot pairwise distances.

Parameters **dist_square** : array_like

The distance matrix in square form.

labels : sequence of strings, optional

Sample labels for the axes.

colorbar : bool, optional

If True, add a colorbar to the current figure.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

vmin : float, optional

The minimum distance value for normalisation.

vmax : float, optional

The maximum distance value for normalisation.

cmap : string, optional

The color map for the image.

imshow_kwargs : dict-like, optional

Additional keyword arguments passed through to *plt.imshow*.

Returns **ax** : axes

The axes on which the plot was drawn

CHAPTER 8

Principal components analysis

Utility functions for running principal components analysis and plotting the results.

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/pca.ipynb

anhima.pca.**pca**(*gn*, *n_components=10*, *whiten=False*)

Perform a principal components analysis of genotypes, treating each variant as a feature.

Parameters **gn** : array_like, shape (*n_variants*, *n_samples*)

A 2-dimensional array where each element is a genotype call coded as a single integer counting the number of non-reference alleles.

n_components : int, None or string

Number of components to keep. If *n_components* is None all components are kept: `n_components == min(n_samples, n_features)`. If *n_components* == 'mle', Minka's MLE is used to guess the dimension. If 0 < *n_components* < 1, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by *n_components*.

whiten : bool

When True (False by default) the components vectors are divided by n_samples times singular values to ensure uncorrelated outputs with unit component-wise variances.

Returns **model** : `sklearn.decomposition.PCA`

The fitted model.

coords : ndarray, shape (*n_samples*, *n_components*)

The result of fitting the model with *genotypes* and applying dimensionality reduction to *genotypes*.

See also:

`sklearn.decomposition.PCA`, *anhima.ld.ld_prune_pairwise*

Notes

The *anhima.ld.ld_prune_pairwise()* can be used to obtain a set of variants in approximate linkage equilibrium prior to running PCA.

anhima.pca.**plot_coords**(*model*, *coords*, *pcx=1*, *pcy=2*, *ax=None*, *colors='b'*, *sizes=20*, *labels=None*, *scatter_kwargs=None*, *annotate_kwargs=None*)

Scatter plot of transformed coordinates from principal components analysis.

Parameters **model** : `sklearn.decomposition.PCA`

The fitted model.

coords : ndarray, shape (*n_samples*, *n_components*)

The transformed coordinates.

pcx : int, optional

The principal component to plot on the X axis. N.B., this is one-based, so *1* is the first principal component, *2* is the second component, etc.

pcy : int, optional

The principal component to plot on the Y axis. N.B., this is one-based, so *1* is the first principal component, *2* is the second component, etc.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

colors : color or sequence of color, optional

Can be a single color format string, or a sequence of color specifications of length *n_samples*.

sizes : scalar or array_like, shape (*n_samples*), optional

Size in points^2.

labels : sequence of strings

If provided, will be used to label points in the plot.

scatter_kwargs : dict-like

Additional keyword arguments passed through to *plt.scatter*.

annotate_kwargs : dict-like

Additional keyword arguments passed through to *plt.annotate* when labelling points.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.pca.**plot_variance_explained**(*model*, *bar_kwargs=None*, *ax=None*)

Parameters **model** : `sklearn.decomposition.PCA`

The fitted model.

bar_kwargs : dict-like, optional

Additional keyword arguments passed through to `ax.bar()`.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.pca.**plot_loadings**(*model*, *pc=1*, *pos=None*, *plot_kwargs=None*, *ax=None*)

Plot loadings for the given principal component.

Parameters **model** : sklearn.decomposition.PCA

The fitted model.

pc : int, optional

The principal component to plot loadings for. N.B., this is one-based, so *1* is the first principal component, *2* is the second component, etc.

pos : array_like, int, optional

An array of variant positions to use for the X axis, If not given, variant index will be used for the X axis.

plot_kwargs : dict-like, optional

Additional keyword arguments passed through to ax.plot().

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

Returns **ax** : axes

The axes on which the plot was drawn.

CHAPTER 9

Multidimensional scaling

Utility functions for multidimensional scaling.

R must be installed, and the Python package `rpy2` must be installed, e.g.:

```
$ apt-get install r-base
$ pip install rpy2
```

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/mds.ipynb

`anhima.mds.`**`smacof`**(*dist_square*, ***kwargs*)

Multidimensional scaling using the SMACOF (Scaling by Majorizing a Complicated Function) algorithm.

Parameters **dist_square** : array_like, shape (n_samples, n_samples)

A distance matrix in square form.

kwargs : additional keyword arguments

Additional keyword arguments are passed through to `sklearn.manifold.MDS()`.

Returns **coords** : ndarray, shape (n_samples, n_components)

An array whose rows give the coordinates of the points chosen to represent the dissimilarities.

See also:

anhima.dist.pairwise_distance, *anhima.mds.classical*, `sklearn.manifold.MDS`, *anhima.pca.pca*

`anhima.mds.`**`classical`**(*dist_square*, *k=2*)

Classical multidimensional scaling via the R `cmdscale` function.

Parameters **dist_square** : array_like, shape (n_samples, n_samples)

A distance matrix in square form.

k : integer, optional

The maximum dimension of the space which the data are to be represented in; must be in {1, 2, ..., n-1}.

Returns **coords** : ndarray, shape (n_samples, k)

An array whose rows give the coordinates of the points chosen to represent the dissimilarities.

See also:

anhima.dist.pairwise_distance, *anhima.mds.smacof*, *anhima.pca.pca*

anhima.mds.**plot_coords**(*coords*, *dimx=1*, *dimy=2*, *ax=None*, *colors='b'*, *sizes=20*, *labels=None*, *scatter_kwargs=None*, *annotate_kwargs=None*)

Scatter plot of transformed coordinates from multidimensional scaling.

Parameters **coords** : ndarray, shape (*n_samples*, *n_components*)

The transformed coordinates.

dimx : int, optional

The dimension to plot on the X axis. N.B., this is one-based, so *1* is the first dimension, *2* is the second dimension, etc.

dimy : int, optional

The dimension to plot on the Y axis. N.B., this is one-based, so *1* is the first dimension, *2* is the second dimension, etc.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

colors : color or sequence of color, optional

Can be a single color format string, or a sequence of color specifications of length *n_samples*.

sizes : scalar or array_like, shape (*n_samples*), optional

Size in points^2.

labels : sequence of strings

If provided, will be used to label points in the plot.

scatter_kwargs : dict-like

Additional keyword arguments passed through to *plt.scatter*.

annotate_kwargs : dict-like

Additional keyword arguments passed through to *plt.annotate* when labelling points.

Returns **ax** : axes

The axes on which the plot was drawn.

See also:

anhima.mds.smacof, *anhima.mds.classical*

CHAPTER 10

Trees

This module provides some facilities for constructing and plotting trees. It is mostly a wrapper around a very limited subset of functions from the R *ape* package (Analyses of Phylogenetics and Evolution).

R must be installed, the *ape* R package must be installed, and the Python package `rpy2` must be installed, e.g.:

```
$ apt-get install r-base
$ pip install rpy2
$ R
> install.packages("ape")
```

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/tree.ipynb

anhima.tree.**nj**(*dist_square*, *labels=None*)

Wrapper for the *ape* `nj` function, which performs the neighbor-joining tree estimation of Saitou and Nei (1987).

Parameters **dist_square** : array_like, shape (*n_samples*, *n_samples*)

A pairwise distance matrix in square form.

labels : sequence of strings, optional

A sequence of strings to label the tips of the tree. Must be in the same order as rows of the distance matrix.

Returns An R object of class "phylo".

See also:

anhima.dist.pairwise_distance

anhima.tree.**bionj**(*dist_square*, *labels=None*)

Wrapper for the *ape* `bionj` function, which performs the BIONJ algorithm of Gascuel (1997).

Parameters **dist_square** : array_like, shape (*n_samples*, *n_samples*)

A pairwise distance matrix in square form.

labels : sequence of strings, optional

A sequence of strings to label the tips of the tree. Must be in the same order as rows of the distance matrix.

Returns An R object of class "phylo".

See also:

anhima.dist.pairwise_distance

anhima.tree.**read_tree**(*filename*, ***kwargs*)

Wrapper for the *ape* `read.tree` function, which reads a file which contains one or several trees in parenthetic format known as the Newick or New Hampshire format.

Parameters **filename** : string

Name of the file to read.

****kwargs** : keyword arguments

All further keyword arguments are passed through to `read.tree`.

Returns **tree** : R object of class "phylo"

If several trees are read in the file, the returned object is of class "multiPhylo", and is a list of objects of class "phylo". The name of each tree can be specified by tree.names, or can be read from the file (see details).

anhima.tree.**write_tree**(*tree*, *filename=None*, ***kwargs*)

Wrapper for the *ape* `write.tree` function, which writes in a file a tree in parenthetic format using the Newick (also known as New Hampshire) format.

Parameters **tree** : R object of class "phylo"

The tree to be written.

filename : string, optional

The name of the file to write to. If ommitted, write the file to a string and return it.

****kwargs** : keyword arguments

All further keyword arguments are passed through to `write.tree`.

Returns **result** : string

A string if *filename* is None, otherwise no return value.

anhima.tree.**plot_phylo**(*tree*, *plot_kwargs=None*, *add_scale_bar=None*, *filename=None*, *width=None*, *height=None*, *units=None*, *res=None*, *pointsize=None*, *bg=None*, *ax=None*, *imshow_kwargs=None*)

Wrapper for the *ape* `plot.phylo` function, which plots phylogenetic trees. Plotting will use the R *png* graphics device.

Parameters **tree** : R object of class "phylo"

The tree to plot.

plot_kwargs : dict-like, optional

A dictionary of keyword arguments that will be passed through to the *ape* function `plot.phylo()`. See the documentation for the *ape* package for a full list of supported arguments.

add_scale_bar : dict-like, optional

A dictionary of keyword arguments that will be passed through to the *ape* function `add.scale.bar()`. See the documentation for the *ape* package for a full list of supported arguments.

filename : string, optional

File path for the generated PNG image. If None, a temporary file will be used.

width : int or float, optional

Width of the plot in *units*.

height : int or float, optional

Height of the plot in *units*.

units : {'px', 'in', 'cm', 'mm'}, optional

The units in which 'height' and 'width' are given. Can be 'px' (pixels, the default), 'in' (inches), 'cm' or 'mm'.

res : int, optional

The nominal resolution in ppi which will be recorded in the bitmap file, if a positive integer. Also used for 'units' other than the default, and to convert points to pixels.

pointsize : float, optional

The default pointsize of plotted text, interpreted as big points (1/72 inch) at 'res' ppi.

bg : color, optional

The background color.

ax : axes, optional

The axes on which to draw. If not provided, a new figure will be created.

imshow_kwargs : dict-like

Additional keyword arguments passed through to *imshow()*.

Returns **ax** : axes

The axes on which the plot was drawn.

anhima.tree.**color_edges_by_group_majority**(*tree*, *labels*, *groups*, *colors*, *equality_color='gray'*)

Color the edges of a tree according to the majority group membership of the descendant tips.

Parameters **tree** : R object of class "phylo"

The tree containing the edges to be colored.

labels : sequence of strings

The tip labels.

groups : sequence of strings

A sequence of strings of the same length as *labels*, where each item is an identifier for the group to which the corresponding tip belongs.

colors : dict-like

A dictionary mapping groups to colors.

equality_color : string, optional

The color to use in the event of a tie.

Returns **edge_colors** : list of strings

A list of colors for the edges of the tree, to be passed into `plot_phylo()`.

CHAPTER 11

Pedigrees (families and crosses)

Utilities for working with related individuals (crosses, families, etc.).

See also the examples at:

- http://nbviewer.ipython.org/github/alimanfoo/anhima/blob/master/examples/ped.ipynb

anhima.ped.**diploid_inheritance**(*parent_diplotype*, *gamete_haplotypes*)

Determine the transmission of parental alleles to a set of gametes.

Parameters **parent_diplotype** : array_like, shape (n_variants, 2)

An array of phased genotypes for a single diploid individual, where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

gamete_haplotypes : array_like, shape (n_variants, n_gametes)

An array of haplotypes for a set of gametes derived from the given parent, where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **inheritance** : ndarray, uint8, shape (n_variants, n_gametes)

An array of integers coding the allelic inheritance, where 1 = inheritance from first parental haplotype, 2 = inheritance from second parental haplotype, 3 = inheritance of reference allele from parent that is homozygous for the reference allele, 4 = inheritance of alternate allele from parent that is homozygous for the alternate allele, 5 = non-parental allele, 6 = parental genotype is missing, 7 = gamete allele is missing.

anhima.ped.**diploid_mendelian_error**(*parental_genotypes*, *progeny_genotypes*)

Find impossible genotypes according to Mendelian inheritance laws.

Parameters **parental_genotypes** : array_like, int

An array of shape (n_variants, 2, 2) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

progeny_genotypes : array_like, int

An array of shape (n_variants, n_progeny, 2) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

Returns **errors** : ndarray, uint8

An array of shape (n_variants, n_progeny) where each element counts the number of non-Mendelian alleles in a progeny genotype call.

See also:

count_diploid_mendelian_error

Notes

Not applicable to polyploid genotype calls.

Applicable to multiallelic variants.

Assumes that genotypes are unphased.

anhima.ped.**count_diploid_mendelian_error**(*parental_genotypes*, *progeny_genotypes*, *axis=None*)

Count impossible genotypes according to Mendelian inheritance laws, summed over all progeny genotypes, or summed along variants or samples.

Parameters **parental_genotypes** : array_like, int

An array of shape (n_variants, 2, 2) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

progeny_genotypes : array_like, int

An array of shape (n_variants, n_progeny, 2) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

axis : int, optional

The axis along which to count (0 = variants, 1 = samples).

Returns **n** : int or array

If *axis* is None, returns the total number of Mendelian errors. If *axis* is specified, returns the sum along the given *axis*.

See also:

diploid_mendelian_error

anhima.ped.**impute_inheritance_nearest**(*inheritance*, *pos*, *pos_impute*)

Impute inheritance at unknown positions, by copying from nearest neighbouring position where inheritance is known.

Parameters **inheritance** : array_like, int, shape (n_variants, n_gametes)

An array of integers coding the allelic inheritance state at the known positions.

pos : array_like, int, shape (n_variants,)

Array of genomic positions at which *inheritance* was determined.

pos_impute : array_like, int

Array of positions at which to impute inheritance.

Returns **imputed_inheritance** : ndarray, int

An array of integers coding the imputed allelic inheritance.

anhima.ped.**tabulate_inheritance_switches**(*inheritance*, *pos*, *gametes=None*)
Tabulate switches in inheritance.

Parameters **inheritance** : array_like, int, shape (n_variants, n_gametes)

An array of integers coding the allelic inheritance state at the known positions.

pos : array_like, int, shape (n_variants,)

Array of genomic positions at which *inheritance* was determined.

gametes : sequence, length (n_gametes), optional

Returns **df** : DataFrame

A table of all inheritance switches observed in the gametes, where each row corresponds to a switch from one parental allele to another. The table has a hierarchical index, where the first level corresponds to the gamete.

See also:

tabulate_inheritance_blocks

anhima.ped.**tabulate_inheritance_blocks**(*inheritance*, *pos*, *gametes=None*)
Tabulate inheritance blocks.

Parameters **inheritance** : array_like, int, shape (n_variants, n_gametes)

An array of integers coding the allelic inheritance state at the known positions.

pos : array_like, int, shape (n_variants,)

Array of genomic positions at which *inheritance* was determined.

gametes : sequence, length (n_gametes), optional

Returns **df** : DataFrame

A table of all inheritance blocks observed in the gametes, where each row corresponds to a block of inheritance from a single parent. The table has a hierarchical index, where the first level corresponds to the gamete.

See also:

tabulate_inheritance_switches

anhima.ped.**inheritance_block_masks**(*inheritance*, *pos*)
Construct arrays suitable for masking the original inheritance array using the attributes of the inheritance blocks.

Parameters **inheritance** : array_like, int, shape (n_variants, n_gametes)

An array of integers coding the allelic inheritance state at the known positions.

pos : array_like, int, shape (n_variants,)

Array of genomic positions at which *inheritance* was determined.

Returns **block_support** : ndarray, int, shape (n_variants, n_gametes)

An array where each item has the number of variants supporting the containing inheritance block.

block_length_min : ndarray, int, shape (n_variants, n_gametes)

An array where each item has the minimal length of the containing inheritance block.

See also:

tabulate_inheritance_blocks

CHAPTER 12

Input/output utilities

Input/output utilities.

anhima.io.**save_tped**(*path*, *genotypes*, *ref*, *alt*, *pos*, *chromosome='0'*, *identifier=None*, *genetic_distance=None*)

Write biallelic diploid genotype data to a file using the Plink transposed format (TPED).

Parameters **path** : string or file-like

Path of file to write, or file-like object to write to.

genotypes : array_like, int

An array of shape (n_variants, n_samples, 2) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, etc.).

ref : array_like, string

A 1-dimensional array of single character strings encoding the reference nucleotide.

alt : array_like, string

A 1-dimensional array of single character strings encoding the alternate nucleotide.

pos : array_like, int

A 1-dimensional array of genomic positions.

chromosome : string or array_like, string, optional

Either a single string (if all variants are from the same chromosome/contig) or an array of strings with the chromosome of each variant.

identifier : array_like, string, optional

An array of SNP identifiers. If not provided, identifiers will be created based on the variant position, e.g., 'snp100042', 'snp100081', etc.

genetic_distance : array_like, float

An array of genetic distances. If not provided, a zero value ('0') will be written for all variants.

CHAPTER 13

HDF5 utilities

Utility functions for working with data stored in HDF5 files.

HDF5 file convention for variant call sets

Note that this module assumes that data for a variant call set has been organised in an HDF5 file following a particular convention. Briefly, the convention is as follows.

An HDF5 file may contain one or more call sets. Each call set is stored within a separate group. A call set may be stored within the root group.

Within each call set, data are grouped by chromosome.

Within each chromosome group, there are two subgroups, named *variants* and *calldata*.

The *variants* group contains one or more datasets holding data on the variants in the call set. The first dimension of all *variants* datasets must have the same length, being the number of variants on the chromosome.

The *variants* group must contain a *POS* dataset holding the genome positions of the variants.

The *calldata* group contains one or more datasets holding data relating to genotype calls. The first dimension of all *calldata* datasets must have the same length, being the number of variants on the chromosome. The second dimension of all *calldata* datasets must have the same length, being the number of samples in the cohort.

In addition to the above, a *samples* dataset may be stored within the callset, providing a list of labels or identifiers for the samples in the cohort. This *samples* dataset may be stored as a child of the callset group and/or as a child of the chromosome groups.

So, for example, an HDF5 file containing a SNP call set for a cohort of *Anopheles gambiae* samples with chromosomes (2R, 2L, 3R, 3L, X) might be organised as follows:

```
/ [callset group]
/samples [dataset, shape (n_samples,), dtype string]
/2L [chromosome group]
/2L/variants [variants group]
/2L/variants/POS [dataset, shape (n_variants,), dtype int32]
/2L/variants/REF [dataset, shape (n_variants,), dtype S1]
/2L/variants/ALT [dataset, shape (n_variants, 3), dtype S1]
/2L/variants/MQ [dataset, shape (n_variants,), dtype f4]
/2L/variants/...
/2L/calldata [calldata group]
/2L/calldata/genotype [dataset, shape (n_variants, n_samples, ploidy), dtype int8]
/2L/calldata/DP [dataset, shape (n_variants, n_samples), dtype=int32]
/2L/calldata/...
/3L/variants/...
/3L/calldata/...
/...
```

anhima.h5.**load_region**(*callset*, *chrom*, *start_position=0*, *stop_position=None*, *variants_fields=None*, *calldata_fields=None*, *variants_query=None*, *samples=None*)

Load data into memory from *callset* for the given region.

Parameters **callset** : HDF5 file or group

A file or group containing a variant call set.

chrom : string

The chromosome to extract data for.

start_position : int, optional

The start position for the region to extract data for.

stop_position : int, optional

The stop position for the region to extract data for.

variants_fields : sequence of strings, optional

Names of the variants datasets to extract.

calldata_fields : sequence of strings, optional

Names of the calldata datasets to extract.

variants_query : string, optional

A query to filter variants. Note that this query is applied after data for the region has been loaded, so any fields referenced in this query need to be included in *variants_fields*.

samples : sequence of strings, optional

Selected samples to extract.

Returns **variants** : dict

A dictionary mapping dataset identifiers to ndarrays.

calldata : dict

A dictionary mapping dataset identifiers to ndarrays.

anhima.h5.**take2d_pointsel**(*dataset*, *row_indices=None*, *col_indices=None*, *block_size=1000*)

Load selected rows and optionally columns from an HDF5 dataset with 2 or more dimensions, using HDF5 point selections.

Parameters **dataset** : HDF5 dataset

The dataset to load data from.

row_indices : sequence of ints, optional

The indices of the selected rows. If not provided, all rows will be returned.

col_indices : sequence of ints, optional

The indices of the selected columns. If not provided, all columns will be returned.

block_size : int, optional

The size (in number of points) of the block of data to load and process at a time.

Returns **out** : ndarray

An array containing the selected rows and columns.

See also:

`anhima.util.take2d`

Notes

This function is similar to `anhima.util.take2d()` but uses an HDF5 point selection under the hood. Performance characteristics will be different and may be much better or much worse, depending on the size, shape and configuration of the dataset, and depending on the number of points to be selected.

`anhima.h5.`**`save_tped`**(*path*, *callset*, *chrom*, *start_position=0*, *stop_position=None*, *samples=None*)

Save genotype data from an HDF5 callset to a Plink transposed format file (TPED).

Parameters **path** : string or file-like

Path of file to write, or file-like object to write to.

callset : HDF5 file or group

A file or group containing a variant call set.

chrom : string

The chromosome to extract data for.

start_position : int, optional

The start position for the region to extract data for.

stop_position : int, optional

The stop position for the region to extract data for.

samples : sequence of strings, optional

Selection of samples to extract genotypes for, defaults to all samples.

Notes

Note that the current implementation loads all data from the requested region into memory before writing out to TPED, so may not be applicable to very large datasets.

CHAPTER 14

Miscellaneous utilities

Miscellaneous utilities.

anhima.util.**block_apply**(*f*, *dataset*, *block_size=None*, *out=None*)

Apply function *f* to *dataset* split along the first axis into contiguous slices of *block_size*. The result should be equivalent to calling `f(dataset)` directly, however may require less total memory, especially if *dataset* is an HDF5 dataset.

Parameters **f** : function

The function to apply.

dataset : array_like or HDF5 dataset

The input dataset.

block_size : int, optional

The size (in number of items along *axis*) of the blocks passed to *f*.

out : array_like or HDF5 dataset, optional

If given, used to store the output.

Returns **out** : ndarray

The result of applying *f* to *dataset* blockwise.

anhima.util.**block_take2d**(*dataset*, *row_indices*, *col_indices=None*, *block_size=None*)

Select rows and optionally columns from a Numpy array or HDF5 dataset with 2 or more dimensions.

Parameters **dataset** : array_like or HDF5 dataset

The input dataset.

row_indices : sequence of ints

The indices of the selected rows. N.B., will be sorted in ascending order.

col_indices : sequence of ints, optional

The indices of the selected columns. If not provided, all columns will be returned.

block_size : int, optional

The size (in number of rows) of the block of data to process at a time.

Returns **out** : ndarray

An array containing the selected rows and columns.

See also:

anhima.util.block_compress2d, *anhima.h5.take2d_pointsel*

Notes

This function is mainly a work-around for the fact that fancy indexing via h5py is currently slow, and fancy indexing along more than one axis is not supported. The function works by reading the entire dataset in blocks of *block_size* rows, and processing each block in memory using numpy.

anhima.util.**block_compress2d**(*dataset*, *row_condition*, *col_condition=None*, *block_size=None*)

Select rows and optionally columns from a Numpy array or HDF5 dataset with 2 or more dimensions.

Parameters **dataset** : array_like or HDF5 dataset

The input dataset.

row_condition : array_like, bool

A boolean array indicating the selected rows.

col_indices : array_like, bool, optonal

A boolean array indicated the selected columns. If not provided, all columns will be returned.

block_size : int, optional

The size (in number of rows) of the block of data to process at a time.

Returns **out** : ndarray

An array containing the selected rows and columns.

See also:

anhima.util.block_take2d, *anhima.h5.take2d_pointsel*

Notes

This function is mainly a work-around for the fact that fancy indexing via h5py is currently slow, and fancy indexing along more than one axis is not supported. The function works by reading the entire dataset in blocks of *block_size* rows, and processing each block in memory using numpy.

CHAPTER 15

Simulations

Extremely naive simulation functions to generate genotype data for illustration of other features in the `anhima` package.

`anhima.sim.`**`simulate_biallelic_genotypes`**(*n_variants*, *n_samples*, *af_dist*, *p_missing=0.1*, *ploidy=2*)

Simulate genotypes at biallelic variants for a population in Hardy-Weinberg equilibrium

Parameters **n_variants** : int

The number of variants.

n_samples : int

The number of samples.

af_dist : frozen continuous random variable

The distribution of allele frequencies.

p_missing : float, optional

The fraction of missing genotype calls.

ploidy : int, optional

The sample ploidy.

Returns **genotypes** : ndarray, int8

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = alternate allele).

`anhima.sim.`**`simulate_genotypes_with_ld`**(*n_variants*, *n_samples*, *correlation=0.2*)

A very simple function to simulate a set of genotypes, where variants are in some degree of linkage disequilibrium with their neighbours.

Parameters **n_variants** : int

The number of variants to simulate data for.

n_samples : int

The number of individuals to simulate data for.

correlation : float, optional

The fraction of samples to copy genotypes between neighbouring variants.

Returns **gn** : ndarray, int8

A 2-dimensional array of shape (n_variants, n_samples) where each element is a genotype call coded as a single integer counting the number of non-reference alleles.

anhima.sim.**simulate_relatedness**(*genotypes*, *relatedness=0.5*, *n_iter=1000*, *copy=True*)

Simulate relatedness by randomly copying genotypes between individuals.

Parameters **genotypes** : array_like

An array of shape (n_variants, n_samples, ploidy) where each element of the array is an integer corresponding to an allele index (-1 = missing, 0 = reference allele, 1 = first alternate allele, 2 = second alternate allele, etc.).

relatedness : float, optional

Fraction of variants to copy genotypes for.

n_iter : int, optional

Number of times to randomly copy genotypes between individuals.

copy : bool, optional

If False, modify *genotypes* in place.

Returns **genotypes** : ndarray, shape (n_variants, n_samples, ploidy)

The input genotype array but with relatedness simulated.

Indices and tables

- genindex
- modindex
- search

a

anhima, 1
anhima.af, 25
anhima.dist, 47
anhima.f2, 37
anhima.gt, 11
anhima.h5, 65
anhima.io, 63
anhima.ld, 41
anhima.loc, 3
anhima.mds, 53
anhima.pca, 49
anhima.ped, 59
anhima.sf, 33
anhima.sim, 71
anhima.tree, 55
anhima.util, 69

A

allele_count() (in module anhima.af), 26
allele_counts() (in module anhima.af), 26
allele_frequencies() (in module anhima.af), 27
allele_frequency() (in module anhima.af), 26
allele_number() (in module anhima.af), 25
allelism() (in module anhima.af), 25
anhima (module), 1
anhima.af (module), 25
anhima.dist (module), 47
anhima.f2 (module), 37
anhima.gt (module), 11
anhima.h5 (module), 65
anhima.io (module), 63
anhima.ld (module), 41
anhima.loc (module), 3
anhima.mds (module), 53
anhima.pca (module), 49
anhima.ped (module), 59
anhima.sf (module), 33
anhima.sim (module), 71
anhima.tree (module), 55
anhima.util (module), 69
as_012() (in module anhima.gt), 16
as_allele_counts() (in module anhima.gt), 17
as_haplotypes() (in module anhima.gt), 16
as_n_alt() (in module anhima.gt), 16

B

bionj() (in module anhima.tree), 55
block_apply() (in module anhima.util), 69
block_compress2d() (in module anhima.util), 70
block_take2d() (in module anhima.util), 69

C

classical() (in module anhima.mds), 53
color_edges_by_group_majority() (in module anhima.tree), 57
compress_variants() (in module anhima.loc), 4
count_called() (in module anhima.gt), 13
count_diploid_mendelian_error() (in module anhima.ped), 60
count_doubletons() (in module anhima.af), 31
count_genotypes() (in module anhima.gt), 18
count_het() (in module anhima.gt), 14
count_hom() (in module anhima.gt), 14
count_hom_alt() (in module anhima.gt), 15
count_hom_ref() (in module anhima.gt), 15
count_missing() (in module anhima.gt), 14
count_non_segregating() (in module anhima.af), 31
count_non_variant() (in module anhima.af), 30
count_segregating() (in module anhima.af), 30
count_shared_doubletons() (in module anhima.f2), 37
count_singletons() (in module anhima.af), 31
count_variant() (in module anhima.af), 30

D

diploid_inheritance() (in module anhima.ped), 59
diploid_mendelian_error() (in module anhima.ped), 59

E

evenly_downsample_variants() (in module anhima.loc), 9

I

impute_inheritance_nearest() (in module anhima.ped), 60
inheritance_block_masks() (in module anhima.ped), 61
is_called() (in module anhima.gt), 11
is_doubleton() (in module anhima.af), 29
is_het() (in module anhima.gt), 12
is_hom() (in module anhima.gt), 12
is_hom_alt() (in module anhima.gt), 13
is_hom_ref() (in module anhima.gt), 12
is_missing() (in module anhima.gt), 11
is_non_segregating() (in module anhima.af), 28
is_non_variant() (in module anhima.af), 28
is_segregating() (in module anhima.af), 28
is_singleton() (in module anhima.af), 29
is_variant() (in module anhima.af), 27

L

ld_prune_pairwise() (in module anhima.ld), 44

load_region() (in module anhima.h5), 65
locate_interval() (in module anhima.loc), 5
locate_intervals() (in module anhima.loc), 6
locate_position() (in module anhima.loc), 4
locate_positions() (in module anhima.loc), 6

M

max_allele() (in module anhima.gt), 15
maximum_likelihood_ancestry() (in module anhima.af), 32

N

nj() (in module anhima.tree), 55
normalise_doubleton_counts() (in module anhima.f2), 37

P

pack_diploid() (in module anhima.gt), 17
pairwise_distance() (in module anhima.dist), 47
pairwise_genotype_ld() (in module anhima.ld), 41
pairwise_ld_decay() (in module anhima.ld), 42
pca() (in module anhima.pca), 49
plot_continuous_calldata() (in module anhima.gt), 22
plot_continuous_calldata_by_sample() (in module anhima.gt), 24
plot_coords() (in module anhima.mds), 54
plot_coords() (in module anhima.pca), 49
plot_diploid_genotypes() (in module anhima.gt), 22
plot_discrete_calldata() (in module anhima.gt), 22
plot_f2_fig() (in module anhima.f2), 39
plot_genotype_counts_by_sample() (in module anhima.gt), 23
plot_genotype_counts_by_variant() (in module anhima.gt), 24
plot_ld_decay_by_distance() (in module anhima.ld), 44
plot_ld_decay_by_separation() (in module anhima.ld), 43
plot_loadings() (in module anhima.pca), 50
plot_pairwise_distance() (in module anhima.dist), 47
plot_pairwise_ld() (in module anhima.ld), 41
plot_phylo() (in module anhima.tree), 56
plot_shared_doubletons() (in module anhima.f2), 38
plot_site_frequency_spectrum() (in module anhima.sf), 34
plot_total_doubletons() (in module anhima.f2), 38
plot_variance_explained() (in module anhima.pca), 50
plot_variant_locator() (in module anhima.loc), 6
plot_windowed_genotype_counts() (in module anhima.gt), 20
plot_windowed_genotype_density() (in module anhima.gt), 20
plot_windowed_genotype_rate() (in module anhima.gt), 21
plot_windowed_ld() (in module anhima.ld), 41
plot_windowed_variant_counts() (in module anhima.loc), 7
plot_windowed_variant_density() (in module anhima.loc), 8

Q

query_variants() (in module anhima.loc), 3

R

randomly_downsample_variants() (in module anhima.loc), 10
read_tree() (in module anhima.tree), 55

S

save_tped() (in module anhima.h5), 67
save_tped() (in module anhima.io), 63
simulate_biallelic_genotypes() (in module anhima.sim), 71
simulate_genotypes_with_ld() (in module anhima.sim), 71
simulate_relatedness() (in module anhima.sim), 72
site_frequency_spectrum() (in module anhima.sf), 33
site_frequency_spectrum_folded() (in module anhima.sf), 34
site_frequency_spectrum_folded_scaled() (in module anhima.sf), 34
site_frequency_spectrum_scaled() (in module anhima.sf), 33
smacof() (in module anhima.mds), 53

T

tabulate_inheritance_blocks() (in module anhima.ped), 61
tabulate_inheritance_switches() (in module anhima.ped), 60
take2d_pointsel() (in module anhima.h5), 66
take_samples() (in module anhima.loc), 3
take_variants() (in module anhima.loc), 4

U

unpack_diploid() (in module anhima.gt), 18

V

view_interval() (in module anhima.loc), 5
view_position() (in module anhima.loc), 5
view_sample() (in module anhima.loc), 3

W

windowed_genotype_counts() (in module anhima.gt), 18
windowed_genotype_density() (in module anhima.gt), 19
windowed_genotype_rate() (in module anhima.gt), 19
windowed_ld_decay() (in module anhima.ld), 43
windowed_statistic() (in module anhima.loc), 9
windowed_variant_counts() (in module anhima.loc), 7
windowed_variant_density() (in module anhima.loc), 8
write_tree() (in module anhima.tree), 56