

PADZE

Pythonic Allelic Diversity Analyzer

User Manual • Version 0.1.0

Andres del Castillo and Yitzchak Shmalo

July 5, 2026

Distributed under the MIT license

PADZE (Pythonic Allelic Diversity Analyzer) measures how many distinct alleles each of several populations carries, and corrects the counts by rarefaction so that populations sampled at different sizes can be compared fairly. For each locus and standardized sample depth it reports allelic richness, private allelic richness, and richness private to a named combination of populations, and then summarizes each statistic across loci. PADZE reads VCF or STRUCTURE genotypes, writes tidy CSV or ready-to-use summary files, and runs from either a **padze** command line or an importable Python API.

PADZE implements the rarefaction estimators of Szpiech, Jakobsson & Rosenberg (2008); cite that paper when you report results. The full citation is in the References.

Contents

1	Introduction	1
2	Core Concepts	1
2.1	Rarefaction and Gene Copies	1
2.2	The Three Richness Statistics	2
2.3	Summaries Across Loci	2
2.4	Rarefaction Depth	2
2.5	Precise Definitions	3
3	Installation	3
4	Quick Start	4
4.1	Inspect the Data with <code>info</code>	4
4.2	Compute the Feature Table	4
4.3	Compute the Classical Summary	5
4.4	Python API	5
5	Input Data	6
5.1	VCF and Population Map	6
5.2	STRUCTURE Input	6
6	Running PADZE	7
6.1	Subcommands	7
6.2	Flag Reference	7
6.3	Performance	9
7	Output Files	10
7.1	Feature CSV (default)	10
7.2	Classical CSV (<code>--classical</code>)	10
7.3	R / P / C Files (<code>--adze-prefix</code>)	10
7.4	FULL Per-Locus Files (<code>--adze-full</code>)	11
7.5	Deleted-Loci Report	11
8	Worked Example	11
9	Validation and Correctness	12
10	References	12

1 Introduction

The number of distinct alleles a population carries—its *allelic diversity*—is a basic measure of genetic variation. It is also treacherous to compare across populations, because a larger sample turns up more alleles simply by examining more gene copies. PADZE removes that sampling bias by *rarefaction*: for every population it estimates how many alleles you would expect to see if you had drawn the same standardized number of gene copies from each. The corrected counts are directly comparable even when the raw sample sizes are not.

For each locus and each standardized sample depth g , PADZE reports three quantities:

- **Allelic richness** (`alpha`)—expected distinct alleles in a population.
- **Private allelic richness** (`pi`)—expected alleles unique to one population.
- **Combination-private richness** (`pihat`)—expected alleles unique to a named *group* of populations.

It computes these at every locus and then summarizes each statistic *across loci*, reporting the mean, variance, and standard error together with the skewness and excess kurtosis of the across-loci distribution. The result is a compact numerical profile of how diversity is distributed across the genome, in a CSV you can analyze directly or feed to downstream models.

PADZE is packaged as the Python project `padze`, with console command `padze` and module entry point `python -m padze`. It is distributed under the MIT license.

Coming from ADZE 1.0?

PADZE runs your existing analyses and adds new capability:

- **Input.** Point PADZE at a VCF plus a one-line-per-sample population map; no STRUCTURE conversion. Your existing STRUCTURE files still work.
- **Install and run.** Install with `pip` on any platform, then use the `padze` CLI or the Python API, in place of per-platform binaries.
- **Reproduce R/P/C.** Run your data with `--adze-prefix` to regenerate the R, P, and C files (and the FULL per-locus files) directly.
- **New statistics.** The same mean, variance, and standard error, plus across-loci skewness and excess kurtosis, and classical statistics over sliding genomic windows.

PADZE matches ADZE 1.0 wherever they overlap. Full mapping and parity details are in `docs/adze-1.0-reference.md`.

2 Core Concepts

This section explains everything needed to read PADZE's output. If you already work with rarefied allelic diversity, skim to Section 2.4 for the depth rule and Section 2.5 for the formulas.

2.1 Rarefaction and Gene Copies

Diversity statistics are counted in *gene copies*, not individuals. A diploid individual contributes two gene copies per locus, a haploid one, and so on. Two diploid samples therefore provide up to four gene copies at a locus.

Because a bigger sample finds more alleles just by looking harder, raw counts are not comparable across populations of different size. *Rarefaction* fixes this by trimming every population to a common standardized depth g (a number of gene copies) and asking: *if you had drawn only g gene copies from this population, how many distinct alleles would you expect to see?* Answering that question for a range of depths $g = 2, 3, \dots$ is the core of what PADZE does. Combinatorial formulas give the exact expectation, so no resampling is needed.

2.2 The Three Richness Statistics

At a single locus and depth g , PADZE evaluates:

- **alpha_j—allelic richness.** Expected number of distinct alleles observed when g gene copies are drawn from population j .
- **pi_j—private allelic richness.** Expected number of alleles *private* to population j —present in j and absent from every other population—at depth g .
- **pihat—combination-private richness.** Expected number of alleles private to a *combination* of populations: present in every member of the named group and absent everywhere outside it. **pihat** generalizes **pi** (a single-population combination is exactly **pi**), which is why it carries the “hat.” For example **pihat_12** counts alleles found in populations 1 and 2 but in no other.

Private always means present here, absent everywhere else. The numeric suffix indexes populations in the population order (by default, the order in which they first appear), so with P populations in total **alpha_1** is the first population, **alpha_2** the second, and **pihat_13** is the combination of the first and third. In the shipped example (populations A, B, C) that makes **alpha_1** population A and **pihat_13** the A – C combination.

2.3 Summaries Across Loci

Each statistic has one value *per locus*. Across the loci in a dataset those values form a distribution, and PADZE summarizes its shape at each depth with five moments:

Moment	Meaning
mean	Average across loci—the headline diversity estimate.
variance	Sample variance across loci (locus-to-locus spread).
se	Standard error of the mean, $\sqrt{\text{variance}/n_{\text{loci}}}$.
skewness	Asymmetry of the across-loci distribution.
kurtosis	Excess kurtosis—tailedness relative to a normal distribution.

The **mean**, **variance**, and **se** are the classical summaries. The **skewness** and **kurtosis** add a description of the distribution’s asymmetry and tails—useful when diversity is concentrated in a few loci rather than spread evenly.

2.4 Rarefaction Depth

You can only rarefy down to a depth the data actually support. The maximum usable depth is set by the **smallest number of non-missing gene copies** available for a population at any single

locus. Two diploid samples give four gene copies per population, but a single missing genotype leaves three copies at that locus—so the depth for that population is capped at three.

Choosing `--max-depth` is a trade-off: a larger g sits closer to the raw allele count but is supported by fewer loci and populations, while a smaller g is more comparable across small samples. PADZE reports the maximum depth your data support in the `info` summary (Section 4.1).

2.5 Precise Definitions

Consider a locus with I distinct alleles. Let N_{ij} be the number of copies of allele i in population j , and

$$N_j = \sum_{i=1}^I N_{ij}$$

the total number of gene copies in j . The probability that a draw of g gene copies from j (without replacement) misses allele i entirely is

$$Q_{ijg} = \frac{\binom{N_j - N_{ij}}{g}}{\binom{N_j}{g}},$$

and $P_{ijg} = 1 - Q_{ijg}$ is the probability the draw contains allele i . Summing over the alleles at the locus,

$$\hat{\alpha}_g^{(j)} = \sum_{i=1}^I P_{ijg}, \quad \hat{\pi}_g^{(j)} = \sum_{i=1}^I P_{ijg} \prod_{\substack{j'=1 \\ j' \neq j}}^P Q_{ij'g},$$

$$\hat{\pi}_g^{(S)} = \sum_{i=1}^I \left(\prod_{j \in S} P_{ijg} \right) \left(\prod_{j \notin S} Q_{ijg} \right).$$

Here $\hat{\alpha}_g^{(j)}$ counts alleles the draw is expected to contain; $\hat{\pi}_g^{(j)}$ counts alleles present in j but absent from every other population; and $\hat{\pi}_g^{(S)}$ counts alleles present in every population of the set S and absent from all populations outside it. Across L loci, the summaries of a per-locus value x are

$$\bar{x} = \frac{1}{L} \sum_{\ell=1}^L x_{\ell}, \quad s^2 = \frac{1}{L-1} \sum_{\ell=1}^L (x_{\ell} - \bar{x})^2, \quad \text{SE} = \sqrt{\frac{s^2}{L}}.$$

3 Installation

PADZE requires Python 3.10 or newer and NumPy. The optional `test` extra adds `pytest` and `SciPy` for the validation suite.

Install from a checkout of the repository:

```
pip install .
```

Install with the test/validation dependencies:

```
pip install -e "[test]"
```

Installing creates the **padze** console command; you can also invoke the package as a module:

```
padze --help
python -m padze --help
```

To run *without installing*, from the `GitHub/` directory of the checkout put the source on the path and call the module:

```
PYTHONPATH=src python -m padze --help
```

Every command in this manual uses the no-install form `PYTHONPATH=src python -m padze ...` so it runs directly from a fresh checkout. After `pip install .` you can drop the `PYTHONPATH=src python -m` prefix and just type **padze**.

4 Quick Start

PADZE ships a small example in `examples/`: `trio.vcf` and `trio.popmap` describe three populations (A, B, C), two diploid samples each, across seven loci.

4.1 Inspect the Data with info

Start with **info** to confirm PADZE reads your data as you expect:

```
PYTHONPATH=src python -m padze info \
  --vcf examples/trio.vcf \
  --popmap examples/trio.popmap
```

Output:

```
source: VCF:examples/trio.vcf
populations (3): A, B, C
samples/pop: A=2, B=2, C=2
loci: kept 7 / read 7
filters: drop all-missing loci
missing call fraction (kept): 0.0119
ADZE feature shape per locus: alpha_j, pi_j (j=1..P) and pihat over population
combinations; summarized across loci as (mean, variance, se, skewness,
kurtosis) per rarefaction depth.
max usable rarefaction depth: 3
```

The depth caps at 3 (not 4) because a missing genotype at `chr1:600` leaves population *A* with three gene copies at that locus—see Section 2.4.

4.2 Compute the Feature Table

The default **features** output is a wide CSV: one row per rarefaction depth g , with the five moment columns for every statistic. Use `--out -` to write to standard output.

```
PYTHONPATH=src python -m padze features \
  --vcf examples/trio.vcf \
  --popmap examples/trio.popmap \
  --max-depth 3 \
  --out -
```

Output (columns and rows truncated; 46 columns total, full schema in Section 7):

```
g,alpha_1_mean,alpha_1_variance,alpha_1_se,alpha_1_skewness,alpha_1_kurtosis, ... ,
  pihat_23_kurtosis
2,1.380952380952381,0.07142857142857144,0.10101525445522108,-0.9839173128183324, ...
3,1.5714285714285714,0.16071428571428573,0.15152288168283162,-0.983917312818333, ...
```

Read the first row as: population *A* (`alpha_1`), rarefied to $g = 2$ gene copies and averaged over 7 loci, is expected to carry **1.38 distinct alleles** per locus, with variance 0.071 and standard error 0.101 across those loci.

4.3 Compute the Classical Summary

Pass `--classical` for a compact long-format table of just the three classical moments—mean, variance, and standard error—one row per statistic and depth:

```
PYTHONPATH=src python -m padze features \
  --vcf examples/trio.vcf \
  --popmap examples/trio.popmap \
  --max-depth 3 \
  --classical \
  --out -
```

Output:

```
statistic,g,n_loci,mean,variance,se
alpha_1,2,7,1.380952380952381,0.07142857142857144,0.10101525445522108
alpha_1,3,7,1.5714285714285714,0.16071428571428573,0.15152288168283162
alpha_2,2,7,1.3095238095238095,0.08730158730158728,0.11167656571008164
alpha_2,3,7,1.4642857142857144,0.19642857142857148,0.1675148485651225
alpha_3,2,7,1.476190476190476,0.05026455026455028,0.0847387162859628
alpha_3,3,7,1.7142857142857142,0.11309523809523812,0.1271080744289442
pi_1,2,7,0.047619047619047616,0.004298941798941799,0.02478173808888253
pi_1,3,7,0,0,0
pi_2,2,7,0.1547619047619048,0.031084656084656086,0.06663831596724867
pi_2,3,7,0.044642857142857144,0.008742559523809524,0.035340303830469995
pi_3,2,7,0.16666666666666669,0.13888888888888892,0.14085904245475278
pi_3,3,7,0.14285714285714285,0.14285714285714285,0.14285714285714285
pihat_12,2,7,0.32142857142857145,0.026455026455026454,0.061475926130276456
pihat_12,3,7,0.24107142857142855,0.06305803571428571,0.0949120161851308
pihat_13,2,7,0.4761904761904762,0.13128306878306875,0.13694788830743965
pihat_13,3,7,0.39285714285714285,0.18452380952380953,0.16235930591649828
pihat_23,2,7,0.2976190476190476,0.0853174603174603,0.11040022018447265
pihat_23,3,7,0.24107142857142858,0.12555803571428573,0.13392857142857142
```

4.4 Python API

The same pipeline is available as an importable library:

```
import padze

loci = padze.read_vcf("examples/trio.vcf", "examples/trio.popmap")
```

```
table = padze.compute_features(loci)
matrix, columns = table.to_frame()
print(matrix.shape)          # (2, 46)
print(columns[:4])           # ['g', 'alpha_1_mean', 'alpha_1_variance', 'alpha_1_se']
```

5 Input Data

PADZE accepts two input formats: **VCF plus a population map** (recommended for new work) and **STRUCTURE** (for parity with, and migration from, existing pipelines).

5.1 VCF and Population Map

VCF is the primary input. PADZE reads genotypes from the GT FORMAT subfield:

- Genotypes are taken from the GT subfield (the first FORMAT field, as usual).
- Ploidy is detected per genotype from the GT separators; samples are diploid by default, and a . allele is treated as a missing gene copy.
- Multiallelic sites are supported: alleles are aligned by VCF allele index across populations, so no biallelic decomposition is required.
- Plain .vcf and gzip-compressed .vcf.gz files are both accepted.

The **population map** (popmap) assigns each VCF sample to a population. It is a plain-text file with one **sample**<TAB>**population** pair per line (any whitespace is accepted; a tab is recommended). Lines beginning with # are comments. The shipped `examples/trio.popmap`:

```
# sample population
sA1      A
sA2      A
sB1      B
sB2      B
sC1      C
sC2      C
```

Population labels are ordered as first encountered. Use `--population-order` when the numbered outputs (`alpha_1`, `pihat_13`, ...) must follow a specific order, for example `--population-order A B C`.

5.2 STRUCTURE Input

STRUCTURE-format input lets you run the exact genotype files used by earlier pipelines through PADZE. A STRUCTURE file has an optional header row of locus names, one or more leading metadata columns (individual label, population, ...), and then the allele tokens. PADZE reads both the two-rows-per-individual layout (one row per gene copy) and the packed one-row-per-individual layout. The relevant flags are documented in Section 6.2; a typical diploid, packed file with one header row and two metadata columns (label, population) is read with:

```
PYTHONPATH=src python -m padze features \
  --structure mydata.stru \
  --structure-header-rows 1 \
```

```
--structure-meta-columns 2 \
--structure-pop-column 2 \
--ploidy 2 \
--one-row-per-individual \
--population-order A B C \
--max-depth 3 \
--classical --out -
```

A STRUCTURE encoding of the trio example produces byte-for-byte the same allelic-richness, private-richness, and `pihat` values as the matched VCF (Section 9). For the crosswalk between these flags and the original ADZE parameter-file keys, see `docs/adze-1.0-reference.md`.

6 Running PADZE

6.1 Subcommands

PADZE has two subcommands, both taking the same input and filtering flags:

- **info** loads the input, applies any filters, and prints a transparent summary—source, population labels, samples per population, loci read vs. kept, filters applied, missing-call fraction, and the maximum usable rarefaction depth.
- **features** runs the full rarefaction pipeline and writes the feature table (default), or the classical, ADZE-compatible, or rolling-window outputs on request.

```
PYTHONPATH=src python -m padze info      --vcf examples/trio.vcf --popmap examples/trio.
popmap
PYTHONPATH=src python -m padze features --vcf examples/trio.vcf --popmap examples/trio.
popmap
```

6.2 Flag Reference

Input and formats

Flag	Meaning
<code>--vcf VCF</code>	Input <code>.vcf</code> / <code>.vcf.gz</code> file.
<code>--popmap POPMAP</code>	Population map: one sample population pair per line; <code>#</code> comments allowed.
<code>--population-order P1 P2 ...</code>	Population order for the numbered outputs (<code>alpha_1</code> , <code>pihat_12</code> , ...). Default: order of first appearance.
<code>--structure STRUCTURE</code>	Input STRUCTURE-format file.
<code>--structure-header-rows N</code>	Leading non-data rows to skip (default 0).
<code>--structure-meta-columns N</code>	Leading metadata columns before the genotypes (default: enough to include the label and population columns).
<code>--structure-pop-column N</code>	1-based column that assigns each individual to a population (default 2).
<code>--structure-label-column N</code>	1-based individual-label column (default 1).

Flag	Meaning
<code>--structure-missing TOKEN</code>	STRUCTURE missing-data token (default -9).
<code>--ploidy P</code>	STRUCTURE ploidy (default 2).
<code>--one-row-per-individual</code>	STRUCTURE rows pack ploidy alleles per locus (one row per individual).
<code>--default-ploidy P</code>	VCF fallback ploidy for a bare missing GT . before a sample's ploidy is observed (default 2).

Filtering and missingness

Flag	Meaning
<code>--require-pass</code>	VCF: keep only records whose FILTER is PASS or ..
<code>--max-missing-fraction F</code>	VCF: drop sites whose overall missing-call fraction exceeds F.
<code>--max-missing-per-population F</code>	Drop a locus if <i>any</i> population exceeds missing fraction F.
<code>--min-populations-genotyped N</code>	Drop sites genotyped in fewer than N populations (default: all populations for the feature table, 1 for classical / ADZE-compatible modes).
<code>--biallelic-only</code>	VCF: drop multiallelic sites.

Statistics and depth

Flag	Meaning
<code>--max-depth G</code>	Largest rarefaction depth g (default: the maximum the data support).
<code>--pihat-sizes K [K ...]</code>	Combination sizes used for <code>pihat</code> (default: 2 when $P \geq 2$, 1 when $P = 1$).
<code>--moments M ...</code>	Which across-loci moments to write in the feature table; each M is one of <code>mean</code> , <code>variance</code> , <code>se</code> , <code>skewness</code> , <code>kurtosis</code> (default: all five).
<code>--depth-policy {common,ragged}</code>	Feature-table depth handling: <code>common</code> requires every locus/population to support each depth; <code>ragged</code> summarizes whatever per-locus values are available up to the largest sample size.
<code>--population-moments</code>	Use biased population skew/kurtosis ($g1$, $g2$) instead of the sample estimators ($G1$, $G2$).

Output selection

Flag	Meaning
<code>--out PATH</code>	Output CSV path (- writes to standard output).

Flag	Meaning
<code>--classical</code>	Emit the classical (<code>mean</code> , <code>variance</code> , <code>se</code>) table over each statistic's full depth range.
<code>--adze-prefix PREFIX</code>	Write ADZE-compatible R/P/C files using this prefix (<code>PREFIX_r</code> , <code>PREFIX_p</code> , <code>PREFIX_c_K</code>).
<code>--adze-full</code>	With <code>--adze-prefix</code> , also write the FULL per-locus files.
<code>--per-locus-out PATH</code>	Also write per-locus values in long format.

Rolling windows

Rolling windows compute the classical statistics over sliding stretches of the genome, so you can trace how diversity changes along a chromosome.

Flag	Meaning
<code>--rolling-window W</code>	Compute the classical statistics over sliding windows of size <code>W</code> .
<code>--step S</code>	Rolling-window step (default: <code>W</code> , i.e. non-overlapping windows).
<code>--window-unit {loci,bp}</code>	Rolling-window unit: number of loci (default) or base pairs.

Example—non-overlapping-ish windows of four loci, stepping by three:

```
PYTHONPATH=src python -m padze features \
  --vcf examples/trio.vcf \
  --popmap examples/trio.popmap \
  --max-depth 2 \
  --classical \
  --rolling-window 4 --step 3 --window-unit loci \
  --out -
```

Output (first rows):

```
window,unit,start,end,n_loci,statistic,g,n_used,mean,variance,se
0,loci,0,4,4,alpha_1,2,4,1.5,0,0
0,loci,0,4,4,alpha_2,2,4,1.25,0.0833333333333333,0.14433756729740643
0,loci,0,4,4,alpha_3,2,4,1.4583333333333335,0.09953703703703705,0.15774745405000762
0,loci,0,4,4,pi_1,2,4,0.0625,0.006365740740740739,0.03989279615651408
0,loci,0,4,4,pi_2,2,4,0.22916666666666669,0.03877314814814815,0.098454492213596
```

6.3 Performance

The rarefaction core is NumPy-vectorized and processes hundreds of loci and thousands of individuals in seconds. The one cost to watch is combination-private richness: the number of population combinations of size k grows as $\binom{P}{k}$ in the number of populations P , so a large combination size over many populations (set with `--pihat-sizes`) can make the `pihat` calculation dominate the run.

7 Output Files

PADZE writes one of several formats depending on the flags you pass. Choose by goal:

Goal	Flags	Format
Analyze in Python / feed downstream models	<i>(default)</i>	Wide feature CSV (§7.1)
Standard summary numbers (mean, variance, SE)	<code>--classical</code>	Long CSV (§7.2)
Diversity along a chromosome	<code>--classical</code> <code>--rolling-window</code>	Long CSV, one block per window (§6.2)
Drop-in R / P / C files	<code>--adze-prefix</code>	R/P/C text files (§7.3)
Per-locus values	<code>--adze-full</code> or <code>--per-locus-out</code>	FULL / long files (§7.4)

7.1 Feature CSV (default)

A wide CSV with **one row per rarefaction depth g** . For every statistic there are **five moment columns**, named `<statistic>_<moment>`, where `<moment>` is one of `mean`, `variance`, `se`, `skewness`, `kurtosis` (excess kurtosis). The statistics are:

- `alpha_j`—allelic richness for population j .
- `pi_j`—private allelic richness for population j .
- `pihat_XY`—combination-private richness for the population combination X, Y (for example `pihat_12`, `pihat_13`, `pihat_23`); combination size is set by `--pihat-sizes`.

For the three-population example that is `alpha_1..alpha_3`, `pi_1..pi_3`, and `pihat_12`, `pihat_13`, `pihat_23`—nine statistics $\times$ five moments = 45 columns, plus the leading `g` column, giving 46. `--moments` restricts which of the five suffixes are written.

7.2 Classical CSV (`--classical`)

A long-format table of the three classical moments, one row per (`statistic`, `depth`):

<code>statistic,g,n_loci,mean,variance,se</code>
--

7.3 R / P / C Files (`--adze-prefix`)

`--adze-prefix PREFIX` writes three summary files, one grouping per block:

- `PREFIX_r`—allelic richness (R file).
- `PREFIX_p`—private allelic richness (P file).
- `PREFIX_c_K`—combination-private richness for combination size K (C file).

Each data line is `grouping ... g num_loci mean variance std_err`.

7.4 FULL Per-Locus Files (--adze-full)

Adding --adze-full to --adze-prefix also writes per-locus files:

- PREFIX_r_fulldata
- PREFIX_p_fulldata
- PREFIX_c_K_fulldata

Each row carries the grouping, depth g , locus count, **one column per locus** (the per-locus statistic value), then the AVG, VAR, and STD.ERR summary columns. --per-locus-out PATH writes the same per-locus values in a single long-format CSV.

7.5 Deleted-Loci Report

When --max-missing-per-population drops loci in R/P/C mode, PADZE writes a companion report PREFIX_p_deletedloci listing the loci excluded because at least one grouping exceeded the missing tolerance.

8 Worked Example

This end-to-end run turns the shipped VCF into a complete set of R/P/C output files. From the `GitHub/` directory:

```
PYTHONPATH=src python -m padze features \
  --vcf examples/trio.vcf \
  --popmap examples/trio.popmap \
  --population-order A B C \
  --adze-prefix out \
  --adze-full
```

PADZE reports the files it wrote:

```
wrote ADZE-compatible files: out_r, out_r_fulldata, out_p, out_p_fulldata, out_c_2,
  out_c_2_fulldata
```

- out_r, out_p, out_c_2—the R, P, and C summary files.
- out_r_fulldata, out_p_fulldata, out_c_2_fulldata—the FULL per-locus files.

The R summary file out_r (allelic richness per population and depth):

```
A 2 7 1.380952380952381 0.07142857142857144 0.10101525445522108
A 3 7 1.5714285714285714 0.1607142857142857 0.1515228816828316

B 2 7 1.3095238095238098 0.08730158730158731 0.11167656571008165
B 3 7 1.4642857142857142 0.19642857142857142 0.16751484856512247
B 4 7 1.5714285714285714 0.2857142857142857 0.20203050891044214

C 2 7 1.4761904761904763 0.050264550264550255 0.08473871628596277
C 3 7 1.7142857142857142 0.11309523809523814 0.1271080744289442
C 4 7 1.8571428571428572 0.1428571428571429 0.14285714285714288
```

Each block reports one population; columns are grouping, depth g , number of loci, mean, variance, and standard error. Each population is carried to the largest depth its own sample size supports— B and C keep all four gene copies at every locus and reach $g = 4$, while A drops to $g = 3$ because of the missing call at `chr1:600` (Section 2.4).

The FULL R file `out_r_fulldata` adds one column per locus. Its header and the population- A block:

```
POP_GROUPING G NUM_LOCI chr1:100 chr1:200 chr1:300 chr1:400 chr1:500 chr1:600 chr1:700 AVG VAR
STD_ERR
A 2 7 1.5 1.5 1.5 1.5 1 1.6666666666666667 1 1.380952380952381 0.07142857142857144
0.10101525445522108
A 3 7 1.75 1.75 1.75 1.75 1 2 1 1.5714285714285714 0.1607142857142857 0.1515228816828316
```

The C file `out_c_2` reports combination-private richness for each pair of populations:

```
A B 2 7 0.32142857142857145 0.026455026455026457 0.061475926130276456
A B 3 7 0.24107142857142858 0.0630580357142857 0.09491201618513079

A C 2 7 0.4761904761904762 0.13128306878306878 0.13694788830743965
A C 3 7 0.39285714285714285 0.18452380952380953 0.16235930591649828

B C 2 7 0.2976190476190476 0.08531746031746033 0.11040022018447268
B C 3 7 0.24107142857142858 0.1255580357142857 0.13392857142857142
```

9 Validation and Correctness

PADZE is a validated implementation of the rarefaction method.

- **Automated test suite.** 82 automated tests pass (with 11 environment-gated tests skipped when optional external artifacts are absent). Run them with `pytest` after `pip install -e ".[test]"`.
- **Numerical core validated against the original C++ ADZE.** The rarefaction core, the classical mean / variance / standard-error summaries, and the per-locus FULL outputs are checked against the original C++ ADZE and agree to print precision.
- **VCF path validated against STRUCTURE.** The VCF reader is validated against matched STRUCTURE encodings of the same genotypes—including allele-coding permutations, multiallelic sites, and missing-data edge cases—so the VCF input path yields exactly the STRUCTURE-derived counts and statistics. A STRUCTURE encoding of the trio example reproduces byte-for-byte the classical values shown in Section 4.3.

For the full verification procedure—commands, expected shapes, and the cross-tool comparison—see `docs/usage-and-verification.md`.

10 References

If you use PADZE’s allelic-rarefaction statistics, cite the paper that introduced the method:

Szpiech ZA, Jakobsson M, Rosenberg NA. 2008. ADZE: a rarefaction approach for counting alleles private to combinations of populations. *Bioinformatics* 24:2498–2504. doi:[10.1093/bioinformatics/btn478](https://doi.org/10.1093/bioinformatics/btn478).

Original ADZE 1.0 source, manual, and prebuilt binaries: <https://github.com/szpiech/ADZE>.

PADZE is an independent Python implementation, not endorsed by or affiliated with the original ADZE authors, and is distributed under the MIT license.