Welcome to the GenoPop Package Documentation

GenoPop is a R package designed to assist with population genomic analyses of data sets from non-model organisms or with low sequencing quality. It's created with the intention to simplify and streamline the analysis of large genomic datasets in VCF (Variant Call Format) files in a efficient manner, while handling problems of missing data.

The GenoPop can be divided into parts. One part contains different genotype imputation methods to deal with missing data in the vcf file, and the second part contains several function to calculate commonly used population genomics metrics, like Fst, and Dxy.

This document will give an overview about GenoPops functionalities and usability. Starting with an overview of its methods, a guide how to install and get started with the package, and lastly a detailed documentation about all the functions of the package.

Imputation Methods Overview

GenoPop includes a variety of imputation methods, which all work without a high quality reference panel to be applicable to data sets of non-model organisms. A key aspect of these methods is their approach to handling large genomic datasets. Recognizing that SNPs within a linkage block share the same evolutionary history, GenoPop employs the assumption that these SNPs exhibit more comparable patterns than those from different linkage blocks. This assumption justifies segmenting the dataset into smaller blocks for parallel processing. Essentially, GenoPop performs a batch-based imputation, where each batch contains SNPs likely to have similar characteristics due to their close proximity in the genome and therefore eventually shared linkage. This approach enhances the efficiency and accuracy of the imputation process, especially in large-scale genomic datasets.

1. Mean Imputation (`meanImputation`)

Mean Imputation is a straightforward approach where missing values are replaced with the mean (average) value of available data. This method can be applied in two ways:

- **By Variant:** The mean value for each variant (e.g., a specific SNP) is calculated using available data across all individuals. Missing values for that variant are then replaced with this mean.

- **By Individual:** The mean value for each individual is calculated using their available data across all variants. Missing values for that individual are replaced with their mean.

This method is fast and simple but assumes that the missing data are randomly distributed and that the mean is a reasonable estimate, which might not always be the case in genomic datasets.

2. k-Nearest Neighbors Imputation (`kNNImputation`)

k-Nearest Neighbors (kNN) is a more nuanced method that imputes missing data based on similarity to neighbors. In this context, 'neighbors' refer to other genomic data points that are most similar to the point with missing data. Key parameters include:

- **k (Number of Neighbors):** Determines how many neighboring data points are considered for imputing a missing value.
- **Max Iterations:** Controls how many times the algorithm iteratively refines its imputations.
- **Batch Size:** Impacts the number of variants processed at a time, affecting both accuracy and computational load.

kNN is more computationally intensive than Mean Imputation but often provides more accurate results, especially in datasets where patterns of similarity can be reliably identified.

3. Random Forest Imputation (`rfImputation`)

Random Forest Imputation uses an ensemble learning method, where multiple decision trees are used to predict missing values. Key aspects include:

- **Number of Trees (ntree):** The total count of decision trees used in the imputation process.
- **Max Iterations (maxiter):** Number of iterations for the algorithm to improve imputations.
- **Batch Size:** Similar to kNN, it affects the computational load and accuracy.

This method is especially powerful for complex datasets with intricate patterns, as it can capture non-linear relationships and interactions between variables. It's generally more computationally demanding than Mean Imputation but a little less then kNN.

Each of these methods has its strengths and is suitable for different types of genomic data and research needs. Mean Imputation is best for quick, general estimates, kNN for datasets where similarity patterns are strong and computational resources are not an issue, and Random Forest for capturing complex relationships in the data while using an acceptable amount of computational ressources.

Population Genomics Metrics Overview

General Functionality

Each function in this part of GenoPop is designed to calculate specific population genomics metrics directly from VCF (Variant Call Format) files. These functions are designed for efficiency and handle large genomic datasets by processing data in batches or windows. Most functions share a set of common parameters:

- **vcf_path**: Path to the VCF file.
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **batch_size**: The number of variants to be processed in each batch.
- **window_size** (optional): Size of the window for windowed analysis in base pairs.
- **skip_size** (optional): Number of base pairs to skip between windows.
- **exclude_ind** (optional): Vector of individual IDs to exclude from the analysis.

In batch mode, the entire VCF file is processed at once to provide a general overview. In window mode, # the file is processed in sections to identify specific regions of interest. These functions typically return single metrics for batch mode or data frames detailing metrics per window.

Metrics Overview

- **FixedSites**: Counts the number of sites fixed for the alternative allele. It helps identify regions with a complete fixation of an allele, potentially indicating selective sweeps or other evolutionary pressures.
- **SegregatingSites**: Counts the number of polymorphic sites, which are not fixed for the alternative allele. It's a measure of genetic variability within the population.
- **SingletonSites**: Counts the number of singleton sites, where a minor allele occurs only once in the sample. It can be an indicator of recent mutations.

- **PrivateAlleles**: Calculates the number of private alleles in two populations. Private alleles are present in one population but absent in another, providing insight into population differentiation.
- **ObservedHeterozygosity (Ho)**: Calculates the observed heterozygosity for each variant. It's a measure of genetic diversity within a population.
- **ExpectedHeterozygosity (He)**: Calculates the expected heterozygosity based on allele frequencies. It's a theoretical measure of how genetically diverse a population should be under random mating.
- **NucleotideDiversity (Pi)**: Measures the average number of nucleotide differences per site between two sequences. It's a key indicator of genetic diversity within a population.
- **Tajima's D**: A neutrality test comparing the number of segregating sites to the average number of nucleotide differences. It can suggest population expansion, selection, or bottlenecks.
- **Watterson's Theta**: A measure of genetic diversity within a population, based on the number of segregating sites.
- **Average Nucleotide Differences (Dxy)**: Measures the average number of nucleotide differences per site between two populations. It's a measure of genetic differentiation.
- **Fst**: The fixation index, measuring genetic differentiation between populations. It ranges from 0 (no differentiation) to 1 (complete differentiation).
- **OneDimSFS**: Calculates a one-dimensional site frequency spectrum, either folded or unfolded. It provides insights into allele frequency distributions within a population.
- **TwoDimSFS**: Calculates a two-dimensional site frequency spectrum for two populations. It's used to infer demographic history and population relationships.

Please note that this summary provides an overview of the functions and their purposes. For complete understanding and appropriate usage, refer to the detailed documentation of GenoPop.

Installation Instructions for GenoPop

Prerequisites

Before installing GenoPop, make sure you have R installed on your system. You can download and install R from CRAN.

Installing GenoPop from GitHub

To install the GenoPop package directly from GitHub, you will need the `devtools` package in R. If you don't have `devtools` installed, you can install it by running the following command in R:

```
install.packages("devtools")
```

Once devtools is installed, you can install GenoPop using the install_github function. Run the following commands in your R console:

```
library(devtools)
install_github("https://github.com/MiGurke/GenoPop")
```

Then load the package..

```
library(GenoPop)
```

and your ready to go!

For the curta people (or others on HPC's)

R installations on curta are a bit tricky and annoying, so here is an easier way. (Which may also be easier in other HPC environments, if they have equivalent preinstalled modules.)

First download the package from github into your local directory using this command:

```
git clone https://github.com/MiGurke/GenoPop.git
```

Then, in all of your slurm scripts using the package load a preinstalled R environment, that already includes all the dependencies for GenoPop. This is command you need to add for this on curta:

```
module load R-bundle-Bioconductor/3.16-foss-2022b-R-4.2.2
```

Then at the very beginning of each R script or command your starting on curta using GenoPop, you add these two lines (of course update the path to your download of the package):

```
library(devtools)
devtools::load_all("/path/to/GenoPop/GenoPop.Rproj")
```

Then your also ready to go without the need to start a battle with conda or R to create your own installation ;)

Functions documentation

Imputation functions

meanImputation

Imputes missing data in VCF files using mean genotypes, either by variant or by individual. This function is designed for large-scale genomic datasets and utilizes efficient batch processing and parallel computing to handle VCF files directly. The function reads a VCF file in batches, performs mean imputation, and writes the imputed data to a new VCF file.

Parameters

- **vcf_path**: Path to the input VCF file.
- **output_vcf**: Path for the output VCF file with imputed data.
- **batch_size**: Number of variants to process per batch (default: 10000).
- **threads**: Number of threads to use for parallel processing (default: 1).
- **write_log**: If TRUE, writes a log of the imputation process (default: FALSE).
- **logfile**: Path to the log file, used if `write_log` is TRUE.
- **mode**: Method for calculating means, either "variant" or "individual" (default: "variant").

Returns

Path to the output VCF file with imputed data.

Examples

```
meanImputation("/path/to/input.vcf",
               "/path/to/output.vcf",
               batch_size = 1000,
               threads = 5,
               write_log = TRUE,
               logfile = "/path/to/logfile.txt",
               mode = "variant")
```

knn_imputeR

This is just the wrapper function for the kNN imputation algorithm that is written in C++. Please use the function kNNImputation().

Parameters

- **data**: NumericMatrix, the data matrix.
- **k**: Integer, the number of neighbors.

Returns

NumericMatrix, the imputed data matrix.

Examples

```
example_matrix <- matrix(c(0, 1, NA, 1, 1, NA, 0, 0, NA, 1, 1, NA, 0, 0,
0, 0, NA, 1, 0, 0, NA, 1, 1, NA, 0), nrow = 5, byrow = TRUE)
knn_imputeR(example_matrix, k = 3)
```

kNNImputation

Performs imputation of missing genomic data using a k-nearest-neighbor algorithm, optimized for large VCF files. This function reads VCF files in batches, applies kNN imputation, and writes the results to a new VCF file. It's designed for efficient processing through parallel computing and chunk-based handling of SNP data. The function uses the 'annoy' library for efficient neighbor detection. Note that neighboring SNPs lacking data will be excluded from the imputation process, and positions with large proportions of missing data may remain unimputed. The choice of batch size, number of neighbors (k), and the number of iterations (maxiter) is critical for balancing accuracy and computational demand.

Parameters

- **vcf_path**: Path to the input VCF file.
- **output_vcf**: Path for the output VCF file with imputed data.
- **k**: Number of nearest neighbors used for imputation (default: 10).
- **maxiter**: Maximum number of iterations for the kNN algorithm (default: 3).
- **batch_size**: Number of variants to process per batch (default: 1000).
- **threads**: Number of threads used for computation (default: 1).
- **write_log**: If TRUE, writes a log file of the process (advised for large datasets).
- **logfile**: Path to the log file, used if `write_log` is TRUE.

Returns

Path to the output VCF file with imputed data.

Examples

```
kNNImputation("/path/to/input.vcf",
              "/path/to/output.vcf",
              k = 10,
              maxiter = 3,
              batch_size = 1000,
              threads = 5,
              write_log = TRUE,
              logfile = "/path/to/logfile.txt")
```

rfImputation

Performs imputation of missing genomic data using a random forest algorithm, optimized for large-scale VCF files. This function reads VCF files in batches, applies random forest imputation via the missForest package, and writes the results to a new VCF file. The choice of batch size, number of improvement iterations, and the number of decision trees in the random forest is critical for balancing accuracy and computational demand.

Parameters

- **vcf_path**: Path to the input VCF file.
- **output_vcf**: Path for the output VCF file with imputed data.
- **batch_size**: Number of variants to process per batch (default: 1000).
- **maxiter**: Number of improvement iterations for the random forest algorithm (default: 10).
- **ntree**: Number of decision trees in the random forest (default: 100).
- **threads**: Number of threads used for computation (default: 1, or one less than available on the system).
- **write_log**: If TRUE, writes a log file of the process (advised for large datasets).
- **logfile**: Path to the log file, used if `write_log` is TRUE.

Returns

Path to the output VCF file with imputed data.

Examples

```
rfImputation("/path/to/input.vcf",
             "/path/to/output.vcf",
             batch_size = 1000,
             maxiter = 10,
             ntree = 100,
             threads = 5,
             write_log = TRUE,
             logfile = "/path/to/logfile.txt")
```

Populations genomics functions

FixedSites

This function counts the number of sites fixed for the alternative allele ("1") in a VCF file. It processes the file in two modes: the entire file at once or in specified windows across the genome. For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach but tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): A single integer representing the total number of fixed sites for the alternative allele across the entire VCF file.
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'FixedSites', representing the count of fixed sites within each window.

Details

The function has two modes of operation:

1. Batch Mode: Processes the entire VCF file in batches to count the total number of fixed sites for the alternative allele. Suitable for a general overview of the entire dataset.
2. Window Mode: Processes the VCF file in windows of a specified size and skip distance. This mode is useful for identifying regions with high numbers of fixed sites, which could indicate selective sweeps or regions of low recombination.

Examples

```
# Batch mode example
vcf_path <- "path/to/vcf/file"
num_fixed_sites <- FixedSites(vcf_path, threads = 4, write_log = TRUE,
logfile = "fixed_sites_log.txt")

# Window mode example
vcf_path <- "path/to/vcf/file"
fixed_sites_df <- FixedSites(vcf_path, threads = 4, write_log = TRUE,
logfile = "windowed_fixed_sites_log.txt", window_size = 100000, skip_size
= 50000)
```

SegregatingSites

This function counts the number of polymorphic or segregating sites (sites not fixed for the alternative allele) in a VCF file. It processes the file in batches or specified windows across the genome. For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): A single integer representing the total number of polymorphic sites across the entire VCF file. In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'PolymorphicSites', representing the count of polymorphic sites within each window.

Examples

```
# Batch mode example
vcf_path <- "path/to/vcf/file"
num_polymorphic_sites <- SegregatingSites(vcf_path, threads = 4,
write_log = TRUE, logfile = "polymorphic_sites_log.txt")

# Window mode example
vcf_path <- "path/to/vcf/file"
polymorphic_sites_df <- SegregatingSites(vcf_path, threads = 4, write_log
= TRUE, logfile = "windowed_polymorphic_sites_log.txt", window_size =
100000, skip_size = 50000)
```

SingletonSites

This function counts the number of singleton sites (sites where a minor allele occurs only once in the sample) in a VCF file. It processes the file in batches or specified windows across the genome. For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): A single integer representing the total number of singleton sites across the entire VCF file.
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'SingletonSites', representing the count of singleton sites within each window.

Examples

```
# Batch mode example
vcf_path <- "path/to/vcf/file"
num_singleton_sites <- SingletonSites(vcf_path, threads = 4, write_log =
TRUE, logfile = "singleton_sites_log.txt")

# Window mode example
vcf_path <- "path/to/vcf/file"
singleton_sites_df <- SingletonSites(vcf_path, threads = 4, write_log =
TRUE, logfile = "windowed_singleton_sites_log.txt", window_size = 100000,
skip_size = 50000)
```

PrivateAlleles

This function calculates the number of private alleles in two populations from a VCF file. (Alleles which are not present in the other population.) It processes the file in batches or specified windows across the genome. For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **pop1_individuals**: Vector of individual names belonging to the first population.
- **pop2_individuals**: Vector of individual names belonging to the second population.
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.

- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): A list containing the number of private alleles for each population.
In window mode (`window_size` and `skip_size` provided): A list of data frames, each with columns 'Chromosome', 'Start', 'End', 'PrivateAllelesPop1', and 'PrivateAllelesPop2', representing the count of private alleles within each window for each population.

Examples

```
# Batch mode example
vcf_path <- "path/to/vcf/file"
pop1_individuals <- c("8449", "8128", "8779")
pop2_individuals <- c("8816", "8823", "8157")
private_alleles <- PrivateAlleles(vcf_path, pop1_individuals,
pop2_individuals, threads = 4, write_log = TRUE, logfile =
"private_alleles_log.txt")

# Window mode example
private_alleles_windows <- PrivateAlleles(vcf_path, pop1_individuals,
pop2_individuals, threads = 4, write_log = TRUE, logfile =
"windowed_private_alleles_log.txt", window_size = 100000, skip_size =
50000)
```

ObservedHeterozygosity

This function calculates the observed heterozygosity (Ho) for a sample in a VCF file. (The proportion of heterozygote genotypes.) For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.

- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): Observed heterozygosity averaged over all loci.
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'Ho', representing the observed heterozygosity within each window.

Examples

```
vcf_path <- "path/to/your/vcf/file"
# Batch mode example
Ho <- ObservedHeterozygosity(vcf_path)
# Window mode example
Ho_windows <- ObservedHeterozygosity(vcf_path, window_size = 100000,
skip_size = 50000)
```

ExpectedHeterozygosity

This function calculates the expected heterozygosity (He) for a sample in a VCF file. The expected heterozygosity is the proportion of heterozygote genotypes expected in the sample, given its allele frequencies, under Hardy-Weinberg Equilibrium. For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): Expected heterozygosity averaged over all loci.
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'He', representing the expected heterozygosity within each window.

Examples

```
vcf_path <- "path/to/your/vcf/file"
# Batch mode example
He <- ExpectedHeterozygosity(vcf_path)
# Window mode example
He_windows <- ExpectedHeterozygosity(vcf_path, window_size = 100000,
skip_size = 50000)
```

Pi

This function calculates the nucleotide diversity (Pi) for a sample in a VCF file as defined by Nei & Li, 1979. The formula used for this is equivalent to the one used in vcftools --window-pi. Handling missing alleles at one site is equivalent to Korunes & Samuk, 2021. The function calculates the number of monomorphic sites using the sequence length and the number of variants in the VCF file. This assumes that all sites not present in the VCF file are invariant sites, which will underestimate Pi because of commonly done (and necessary) variant filtering. However, this calculation would only work with VCF files that include all monomorphic sites, which is quite unpractical for common use cases and will increase computational demands significantly. If you know the number of filtered out sites vs the number of monomorphic sites, use the number of monomorphic + the number of polymorphic (number of variants in your VCF) sites as the sequence length to get the most accurate estimation of Pi. (This does not work for the window mode of this function, which assumes the sequence length to be the window size.) For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **seq_length**: Total length of the sequence in number of bases (used in batch mode only).
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.

- **logfile**: Path to the log file where progress will be logged.
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): Nucleotide diversity (Pi) across the sequence.
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'Pi', representing the nucleotide diversity within each window.

Examples

```
vcf_path <- "path/to/your/vcf/file"
total_sequence_length <- 265392  # Total length of the sequence
# Batch mode example
pi_value <- Pi(vcf_path, total_sequence_length)
# Window mode example
pi_windows <- Pi(vcf_path, seq_length = total_sequence_length,
window_size = 100000, skip_size = 50000)
```

TajimasD

This function calculates Tajima's D statistic for a given dataset (Tajima, 1989). The formula used for this is equivalent to the one used in vcftools --TajimaD. The function calculates the number of monomorphic sites using the sequence length and the number of variants in the VCF file. This assumes that all sites not present in the VCF file are invariant sites, which will underestimate Pi, because of commonly done (and necessary) variant filtering. However, this calculation would only work with VCF files that include all monomorphic sites, which is quite unpractical for common use cases and will increase computational demands significantly. If you know the number of filtered out sites vs the number of monomorphic sites, use the number of monomorphic + the number of polymorphic (number of variants in your VCF) sites as the sequence length to get the most accurate estimation of Pi. (This does not work for the window mode of this function, which assumes the sequence length to be the window size.) For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **seq_length**: Total length of the sequence in number of bases (used in batch mode only).
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): Tajima's D value.
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'TajimasD', representing Tajima's D within each window.

Examples

```
vcf_path <- "path/to/your/vcf/file"
total_sequence_length <- 265392  # Total length of the sequence
# Batch mode example
tajimas_d <- TajimasD(vcf_path, total_sequence_length)
# Window mode example
tajimas_d_windows <- TajimasD(vcf_path, seq_length =
total_sequence_length, window_size = 100000, skip_size = 50000)
```

WattersonsTheta

This function calculates Watterson's Theta, a measure for neutrality, from a VCF file (Watterson, 1975). The function calculates the number of monomorphic sites using the sequence length and the number of variants in the VCF file. This assumes that all sites not present in the VCF file are invariant sites, which will underestimate Pi, because of commonly done (and necessary) variant filtering. However, this calculation would only work with VCF files that include all monomorphic sites, which is quite unpractical for common use cases and will increase computational demands significantly. If you know the number of filtered out sites vs the number of monomorphic sites, use the number of monomorphic + the number of polymorphic

(number of variants in your VCF) sites as the sequence length to get the most accurate estimation of Pi. (This does not work for the window mode of this function, which assumes the sequence length to be the window size.) For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **seq_length**: The length of the sequence in the data set (used in batch mode only).
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): Watterson's theta value normalized by the sequence length.
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'WattersonsTheta', representing Watterson's theta within each window normalized by the window length.

Examples

```
vcf_path <- "path/to/your/vcf/file"
total_sequence_length <- 265392  # Total length of the sequence
# Batch mode example
wattersons_theta <- WattersonsTheta(vcf_path, total_sequence_length)
# Window mode example
wattersons_theta_windows <- WattersonsTheta(vcf_path, seq_length =
total_sequence_length, window_size = 100000, skip_size = 50000)
```

Dxy

This function calculates the average number of nucleotide differences per site (Dxy) between two populations from a VCF file (Nei & Li, 1979). Handling missing alleles at one site is equivalent to Korunes & Samuk, 2021. The function calculates the number of monomorphic sites using the sequence length and the number of variants in the VCF file. This assumes that all sites not present in the VCF file are invariant sites, which will underestimate Pi, because of commonly done (and necessary) variant filtering. However, this calculation would only work with VCF files that include all monomorphic sites, which is quite unpractical for common use cases and will increase computational demands significantly. If you happen to know the number of filtered out sites vs the number of monomorphic sites, please use the number of monomorphic + the number of polymorphic (number of variants in your VCF) sites as the sequence length to get the most accurate estimation of Pi. (This does not work for the window mode of this function, which assumes the sequence length to be the window size.) For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **pop1_individuals**: Vector of individual names belonging to the first population.
- **pop2_individuals**: Vector of individual names belonging to the second population.
- **seq_length**: Length of the sequence in number of bases, including monomorphic sites (used in batch mode only).
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.
- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): The average number of nucleotide substitutions per site between the individuals of two populations (Dxy). In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'Dxy', representing the average nucleotide differences within each window.

Examples

```
vcf_path <- "path/to/your/vcf/file"
pop1_individuals <- c("8449", "8128", "8779")
pop2_individuals <- c("8816", "8823", "8157")
total_sequence_length <- 265392  # Total length of the sequence
# Batch mode example
dxy_value <- Dxy(vcf_path, pop1_individuals, pop2_individuals,
total_sequence_length)
# Window mode example
dxy_windows <- Dxy(vcf_path, pop1_individuals, pop2_individuals,
seq_length = total_sequence_length, window_size = 100000, skip_size =
50000)
```

Fst

This function calculates the fixation index (Fst) between two populations from a VCF file using the method of Weir and Cockerham (1984). The formula used for this is equivalent to the one used in vcftools --weir-fst-pop. For batch processing, it uses `process_vcf_in_batches`. For windowed analysis, it uses a similar approach tailored to process specific genomic windows (`process_vcf_in_windows`).

Parameters

- **vcf_path**: Path to the VCF file.
- **pop1_individuals**: Vector of individual names belonging to the first population.
- **pop2_individuals**: Vector of individual names belonging to the second population.
- **weighted**: Logical, whether weighted Fst or mean Fst is returned (Default = FALSE (mean Fst is returned)).
- **batch_size**: The number of variants to be processed in each batch (used in batch mode only, default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **window_size**: Size of the window for windowed analysis in base pairs (optional). When specified, `skip_size` must also be provided.

- **skip_size**: Number of base pairs to skip between windows (optional). Used in conjunction with `window_size` for windowed analysis.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

In batch mode (no `window_size` or `skip_size` provided): Fst value (either mean or weighted).
In window mode (`window_size` and `skip_size` provided): A data frame with columns 'Chromosome', 'Start', 'End', and 'Fst', representing the fixation index within each window.

Examples

```
vcf_path <- "path/to/your/vcf/file"
pop1_individuals <- c("8449", "8128", "8779")
pop2_individuals <- c("8816", "8823", "8157")
# Batch mode example
fst_value <- Fst(vcf_path, pop1_individuals, pop2_individuals, weighted =
TRUE)
# Window mode example
fst_windows <- Fst(vcf_path, pop1_individuals, pop2_individuals, weighted
= TRUE, window_size = 100000, skip_size = 50000)
```

OneDimSFS

This function calculates a one-dimensional site frequency spectrum from a VCF file. It processes the file in batches for efficient memory usage. The user can decide between a folded or unfolded spectrum.

Parameters

- **vcf_path**: Path to the VCF file.
- **folded**: Logical, deciding if folded (TRUE) or unfolded (FALSE) SFS is returned.
- **batch_size**: The number of variants to be processed in each batch (default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

Site frequency spectrum as a named vector.

Examples

```
vcf_path <- "path/to/your/vcf/file"
sfs <- OneDimSFS(vcf_path, folded = FALSE)
```

TwoDimSFS

This function calculates a two-dimensional site frequency spectrum from a VCF file for two populations. It processes the file in batches for efficient memory usage. The user can decide between a folded or unfolded spectrum.

Parameters

- **vcf_path**: Path to the VCF file.
- **pop1_individuals**: Vector of individual names belonging to the first population.
- **pop2_individuals**: Vector of individual names belonging to the second population.
- **folded**: Logical, deciding if folded (TRUE) or unfolded (FALSE) SFS is returned.
- **batch_size**: The number of variants to be processed in each batch (default of 10,000 should be suitable for most use cases).
- **threads**: Number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: Path to the log file where progress will be logged.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis.

Returns

Two-dimensional site frequency spectrum as a matrix.

Examples

```
vcf_path <- "path/to/your/vcf/file"
pop1_individuals <- c("8449", "8128", "8779")
pop2_individuals <- c("8816", "8823", "8157")
sfs_2d <- TwoDimSFS(vcf_path, pop1_individuals, pop2_individuals, folded
= TRUE)
```

Internal Batch Processing of VCF Data

This function is designed for efficient internal processing of large VCF files. It reads the VCF file in batches, processes each batch in parallel, and applies a custom function to the processed data. It's optimized for performance with large genomic datasets and is not intended to be used directly by end users.

Parameters

- **vcf_path**: The path to the VCF file.
- **batch_size**: The number of variants to be processed in each batch.
- **custom_function**: A custom function that takes two arguments: a data frame similar to the `@fix` slot of a `vcfR` object and a genotype matrix similar to the `@sep_gt` slot. This function is applied to each batch of data.
- **threads**: The number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: The path to the log file.
- **exclude_ind**: Optional vector of individual IDs to exclude from the analysis. If provided, the function will remove these individuals from the genotype matrix before applying the custom function. Default is NULL, meaning no individuals are excluded.

Details

The function divides the VCF file into batches based on the specified `batch_size`. Each batch is processed to extract fixed information and genotype data, filter for biallelic SNPs, and then apply the `custom_function`. The function uses parallel processing to improve efficiency and can handle large genomic datasets.

This function is part of the internal workings of the package and is not intended to be called directly by the user. It is documented for the sake of completeness and to assist in maintenance and understanding of the package's internal mechanics.

Return

A data frame that is the combined result of applying the `custom_function` to each batch. The structure of this data frame depends on the `custom_function` used.

Internal Window-Based Processing of VCF Data

This function is designed for efficient internal processing of large VCF files based on specified window sizes and genomic skips. It reads the VCF file in windows, processes each window in parallel, and applies a custom function to the processed data. It's optimized for performance with large genomic datasets and is not intended to be used directly by end users.

######## Parameters

- **vcf_path**: The path to the VCF file.
- **window_size**: The genomic length of each window in base pairs.
- **skip_size**: The size of the genomic region to skip between windows in base pairs.

- **custom_function**: A custom function that takes two arguments: a data frame similar to the `@fix` slot of a `vcfR` object and a genotype matrix similar to the `@sep_gt` slot. This function is applied to each window of data.
- **threads**: The number of threads to use for parallel processing.
- **write_log**: Logical, indicating whether to write progress logs.
- **logfile**: The path to the log file.

Details

The function divides the VCF file into windows based on the specified `window_size` and `skip_size`. Each window is processed to extract fixed information and genotype data, filter for biallelic SNPs, and then apply the `custom_function`. The function uses parallel processing to improve efficiency and can handle large genomic datasets.

This function is part of the internal workings of the package and is not intended to be called directly by the user. It is documented for the sake of completeness and to assist in maintenance and understanding of the package's internal mechanics.

Return

A list that is the combined result of applying the `custom_function` to each window. The structure of this list depends on the `custom_function` used.

Separate Genotype Matrix by Populations

This function separates a genotype matrix into two data frames based on population assignments. It's designed to work with the batches or windows processed by `process_vcf_in_batches` and `process_vcf_in_windows`.

Parameters

- **sep_gt**: A genotype matrix similar to the `@sep_gt` slot of a `vcfR` object.
- **pop1_names**: A character vector of individual names for the first population.
- **pop2_names**: A character vector of individual names for the second population.
- **rm_ref_alleles**: Logical, whether variants that only have the reference allele should be removed from the respective subpopulations data frame. (Default = TRUE)

Return

A list containing two data frames, one for each population.

Examples

```
sep_gt <- matrix(...) # Example genotype matrix
pop1_names <- c("Individual1", "Individual2", ...)
pop2_names <- c("Individual5", "Individual6", ...)

separated_data <- separateByPopulations(sep_gt, pop1_names, pop2_names)
```

Calculate Allele Frequencies from Genotype Matrix

This function calculates allele frequencies from a genotype matrix (sep_gt) for each variant. It is designed to be used within the batch or window processing framework of `process_vcf_in_batches` and `process_vcf_in_windows`.

Parameters

- **sep_gt**: Genotype matrix similar to the `@sep_gt` slot of a `vcfR` object.

Return

A data frame containing allele frequencies for each variant.

Examples

```
sep_gt <- ... # Obtain this from a batch processed by
`process_vcf_in_batches`
allele_freqs <- calculateAlleleFreqs(sep_gt)
```