Package ‘poolfstat’

February 23, 2026

Version 3.1.0

License GPL (>= 2)

Title Computing f-Statistics and Building Admixture Graphs Based on
Allele Count or Pool-Seq Read Count Data

Description Functions for the computation of F-, f- and D-statistics (e.g., Fst, hierarchical F-statistics, Patterson's F2, F3, F3*, F4 and D parameters) in population genomics studies from allele count or Pool-Seq read count data and for the fitting, building and visualization of admixture graphs. The package also includes several utilities to manipulate Pool-Seq data stored in standard format (e.g., such as 'vcf' files or 'rsync' files generated by the the 'PoPoolation' software) and perform conversion to alternative format (as used in the 'BayPass' and 'SelEstim' software). As of version 2.0, the package also includes utilities to manipulate standard allele count data (e.g., stored in 'TreeMix', 'BayPass' and 'SelEstim' format, see the Package vignette for details).

LinkingTo Rcpp, RcppProgress

Imports Rcpp (>= 1.0.5), methods, data.table, utils, foreach,
doParallel, parallel, DiagrammeR, ape, stats, Ryacas, Matrix,
RcppProgress, progress, nnls

Depends R (>= 3.0)

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Mathieu Gautier [aut, cre]

Maintainer Mathieu Gautier <`mathieu.gautier@inrae.fr`>

Repository CRAN

Date/Publication 2026-02-23 18:30:02 UTC

Contents

add.leaf 3
bjack_cov 5
block_sum 6

block_sum2 6
compare.fitted.fstats 7
compute.f4ratio 8
compute.fstats 9
compute.pairwiseFST 11
computeFST 12
compute_blockDdenom 14
compute_F3fromF2 15
compute_F3fromF2samples 15
compute_F4DfromF2samples 16
compute_F4fromF2 17
compute_F4fromF2samples 18
compute_H1 18
compute_Q2 19
compute_QmatfromF2samples 20
compute_snpFstAov 21
compute_snpHierFstAov 21
compute_snpQ1 22
compute_snpQ1onepop 23
compute_snpQ1rw 24
compute_snpQ2 25
compute_snpQ2onepair 25
compute_snpQ2rw 26
countdata-class 27
countdata.subset 27
countdata2genobaypass 29
data.merge 30
extract_allele_names 32
extract_nonvscan_counts 33
extract_vscan_counts 34
find.tree.popset 35
find_indelneighbor_idx 36
fit.graph 37
fitted.graph-class 39
fstats-class 40
generate.graph.params 41
generate.jackknife.blocks 43
generateF3names 44
generateF4names 44
genobaypass2countdata 45
genobaypass2pooldata 46
genoselestim2pooldata 48
genotreemix2countdata 49
graph.builder 51
graph.params-class 52
graph.params2qpGraphFiles 54
graph.params2symbolic.fstats 55
heatmap,pairwisefst-method 56

is.countdata . . . 58
is.fitted.graph . . . 58
is.fstats . . . 58
is.graph.params . . . 59
is.pairwisefst . . . 59
is.pooldata . . . 59
make.example.files . . . 60
pairwisefst-class . . . 60
plot,fitted.graph-method . . . 61
plot,fstats-method . . . 61
plot,graph.params-method . . . 62
plot,pairwisefst-method . . . 62
plot_fstats . . . 63
pooldata-class . . . 64
pooldata.subset . . . 65
pooldata2diyabc . . . 67
pooldata2genobaypass . . . 68
pooldata2genoselestim . . . 69
poolify . . . 70
poppair_idx . . . 72
popsync2pooldata . . . 72
randomallele.pca . . . 74
rooted.njtree.builder . . . 75
scan_allele_info . . . 77
show,countdata-method . . . 77
show,fitted.graph-method . . . 78
show,fstats-method . . . 78
show,graph.params-method . . . 79
show,pairwisefst-method . . . 79
show,pooldata-method . . . 80
sim.readcounts . . . 80
simureads_mono . . . 82
simureads_poly . . . 83
sliding.windows.fstat . . . 84
vcf2pooldata . . . 86

Index **89**

`add.leaf` *Test all possible connection of a leaf to a graph with non-admixed and or admixed edges*

Description

Test all possible connection of a leaf to a graph with non-admixed and or admixed edges

Usage

```
add.leaf(
  x,
  leaf.to.add,
  fstats,
  only.test.non.admixed.edges = FALSE,
  only.test.admixed.edges = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	An object of class graph.params or fitted.graph (see details)
`leaf.to.add`	Name of the leaf to add
`fstats`	Object of class fstats that contains estimates of the fstats (see compute.fstats)
`only.test.non.admixed.edges`	If TRUE the function only test non.admixed edges (may be far faster)
`only.test.admixed.edges`	If TRUE the function only test admixed edges
`verbose`	If TRUE extra information is printed on the terminal
`...`	Some parameters to be passed the function fit.graph called internally

Details

The input object x needs to be of class graph.params (as generated by the function generate.graph.params) or fitted.graph (as generated by the function fit.graph or by the function add.leaf itself in the graphs.fit.res elements of the output list). This is to ensure that the matrix describing the structure of the graph (graph slot of these objects) is valid (note that it can be plotted for checks). Hence graph.params objects may have been generated without fstats information (that should be supplied independently to the add.leaf function to obtain information on the fstats involving the candidate leaf defined with the leaf.to.add argument). By default the function tests all the possible positions of a newly added edge connecting the candidate leaf to the graph with both non-admixed (including a new rooting with the candidate leaf as an outgroup) and admixed edges. If n_e is the the number of non-admixed edges of the original graph, the number of tested graphs for non-admixed edges equals n_e+1. The newly added node is named "N-"[name of the leaf to add] (or with more N if the name already exists). For admixed edges, the number of tested graphs equals n_e*(n_e-1)/2 and for a given tested graph, three nodes named "S-"[name of the leaf to add], "S1-"[name of the leaf to add] and "S2-"[name of the leaf to add] (or with more S if the name already exists) are added and the admixture proportions are named with a letter (A to Z depending on the number of admixed nodes already present in the graph).

Value

A list with the following elements:

1. "n.graphs": The number of tested graphs

2. "fitted.graphs.list": a list of fitted.graph objects (indexed from 1 to n.graphs and in the same order as the list "graphs") containing the results of fitting of each graph.
3. "best.fitted.graph": The graph (object of class fitted.graph) with the minimal BIC (see function fit.graph) among all the graphs within fitted.graphs.list
4. "bic": a vector of the n.graphs BIC (indexed from 1 to n.graphs and in the same order as the "fitted.graphs.list" list) (see fit.graph details for the computation of the scores).

See Also

see `fit.graph` and `generate.graph.params`.

`bjack_cov` *bjack_cov*

Description

Compute the block-jackknife covariance between two stats

Arguments

`stat1`	Vector of block-jackknife values for the first stat
`stat2`	Vector of block-jackknife values for the second stat

Details

Compute the block-jackknife covariance between two stats with correction

Value

Covariance values

Examples

```
#
```

block_sum *block_sum*

Description

Sugar to compute the sum of a stat per block

Usage

```
.block_sum(stat, snp_bj_id)
```

Arguments

`stat`	vector of n stat values
`snp_bj_id`	integer n-length vector with block index (from 0 to nblock-1) of the stat value

Details

Sugar to compute the sum of a stat per block

Value

Return a vector of length nblocks containing the per-block sums of the input stat

Examples

```
#
```

block_sum2 *block_sum2*

Description

Sugar to compute the sum of a stat per block defined by a range of SNPs (allow treating overlapping blocks)

Usage

```
.block_sum2(stat, snp_bj_id)
```

Arguments

`stat`	vector of n stat values
`snp_bj_id`	integer matrix of dim nblocks x 2 giving for each block the start and end stat value index

Details

Sugar to compute the sum of a stat per block defined by a range of SNPs (allow treating overlapping blocks)

Value

Return a vector of length nblocks containing the per-block sums of the input stat

Examples

```
#
```

`compare.fitted.fstats` *Compare fitted f2, f3 and f4 f-statistics of an admixture graph with estimated ones*

Description

Compare fitted f2, f3 and f4 f-statistics of an admixture graph with estimated ones

Usage

```
compare.fitted.fstats(fstats, fitted.graph, n.worst.stats = 5)
```

Arguments

`fstats`	Object of class fstats containing estimates of fstats (as obtained with compute.fstats)
`fitted.graph`	Object of class fitted graph (as obtained with fit.graph function).
`n.worst.stats`	The number of worst statistics to be displayed in the terminal

Details

Compare fitted and estimated f-statistics may allow identifying problematic edges on the graph.

Value

A matrix with 3 columns for each test (row names of the matrix corresponding to the test):

1. The estimated f-statistics (mean across block-Jackknife samples)
2. The fitted f-statistics (obtained from the fitted grah parameters
3. A Z-score measuring the deviation of the fitted values from the estimated values in units of standard errors (i.e., Z=(fitted.value-target.value)/se(target.value))

See Also

See `compute.fstats` and `fit.graph`

`compute.f4ratio` *Compute F4ratio (estimation of admixture rate) from an fstats object*

Description

Compute F4ratio (estimation of admixture rate) from an fstats object

Usage

```
compute.f4ratio(x, num.quadruplet, den.quadruplet)
```

Arguments

`x`	A fstats object containing estimates of fstats
`num.quadruplet`	A character string for the F4 quadruplet used in the F4ratio numerator (should be of the form "A,O;C,X" where A, O, C and X are the names of the population as defined in the countdata or pooldata object used to obtain fstats, see details)
`den.quadruplet`	A character string for the F4 quadruplet used in the F4ratio denominator (should be of the form "A,O;C,B" where A, O, C and B are the names of the populations as defined in the countdata or pooldata object used to obtain fstats, see details))

Details

Assuming a 4 population phylogeny rooted with an outgroup O of the form (((A,B);C);O) and an admixed population X with two source populations related to B and C, the admixture rate alpha of the B-related ancestry is obtained using the ratio F4(A,O;C,X)/F4(A,O;C,B) (see Patterson et al., 2012 for more details).

Value

A vector with 5 elements corresponding. The first element is always the estimated value. If F2 block-jackknife samples are available in the input fstats object (i.e., compute.fstats was run with return.F2.blockjackknife.samples = TRUE), the four other elements are the block-jackknife mean; the block-jackknife s.e.; and the lower and upper bound of the 95

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`,`genobaypass2pooldata` or `genoselestim2pooldata`. To generate coundata object, see `genobaypass2countdata` or `genotreemix2countdata`.

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 res.fstats=compute.fstats(pooldata)
```

`compute.fstats` *Estimate the F-statistics (F2, F3, F3star, F4, Dstat) and within and across population diversity*

Description

Estimate the F-statistics (F2, F3, F3star, F4, Dstat) and within and across population diversity

Usage

```
compute.fstats(
  x,
  nsnp.per.bjack.block = 0,
  computeDstat = FALSE,
  computeF3 = TRUE,
  computeF4 = TRUE,
  output.pairwise.fst = TRUE,
  output.pairwise.div = TRUE,
  computeQmat = TRUE,
  return.F2.blockjackknife.samples = FALSE,
  return.F4.blockjackknife.samples = FALSE,
  verbose = TRUE
)
```

Arguments

`x`	A pooldata object containing Pool-Seq information or a countdata object containing allele count information
`nsnp.per.bjack.block`	Number of consecutive SNPs within a block for block-jackknife (default=0, i.e., no block-jackknife sampling)
`computeDstat`	If TRUE compute Dstatistics (i.e. scaled F4). This may add some non negligible computation time if the number of population is large (n>15)
`computeF3`	If TRUE (default) compute all F3 and all F3star (i.e. scaled F3).
`computeF4`	If TRUE (default) compute all F4.
`output.pairwise.fst`	If TRUE (default), output the npopxnpop matrix of pairwise-population Fst estimates (corresponding to the "Identity" method implemented in `compute.pairwiseFST`) in the pairwise.fst slot of the fstats output object (see help(fstats) for details) that may be visualized with e.g. heatmap function or used with a clustering function (e.g., hclust).
`output.pairwise.div`	If TRUE (default), output the npopxnpop matrix of pairwise-population divergence (1-Q2) estimates in the pairwise.div slot of the fstats output object (see help(fstats) for details) that may be visualized with e.g. heatmap function or used with a clustering function (e.g., hclust).

`computeQmat`	If TRUE, compute the error covariance matrix between all F3 and F2 statistics (needed for admixture graph construction). This matrix may be very large if the number of pops is large. It is recommended to estimate it on a reduced sample of pops.
`return.F2.blockjackknife.samples`	If TRUE (and nsnp.per.bjack.block>0) return an array of dimension (npopxnpopxnblocks) in an admixtools2 compatible format
`return.F4.blockjackknife.samples`	Deprecated options (since v. 2.2.0)
`verbose`	If TRUE extra information is printed on the terminal

Details

The function estimates for the n populations (or pools) represented in the input object x:

1. The F2 statistics for all the $n(n-1)/2$ pairs of populations (or pools) and their scaled version (equivalent, but faster, than Fst estimated with `compute.pairwiseFST` when method="Identity")
2. If n>2, The F3 statistics for all the $npools(npools-1)(npools-2)/2$ possible triplets of populations (or pools) and their scaled version (named F3star after Patterson et al., 2012)
3. If n>3, The F4 statistics and the D-statistics (a scaled version of the F4) for all the $npools(npools-1)(npools-2)*(npools-3)/8$ possible quadruplets of populations
4. The estimated within population heterozygosities (=1-Q1)
5. The estimated divergence for each pair of populations (=1-Q2)

Value

An object of class fstats (see help(fstats) for details)

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`,`genobaypass2pooldata` or `genoselestim2pooldata`. To generate coundata object, see `genobaypass2countdata` or `genotreemix2countdata`.

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 res.fstats=compute.fstats(pooldata)
```

`compute.pairwiseFST` *Compute pairwise population population FST matrix (and possibly all pairwise SNP-specific FST)*

Description

Compute pairwise population population FST matrix (and possibly all pairwise SNP-specific FST)

Usage

```
compute.pairwiseFST(
  x,
  method = "Anova",
  min.cov.per.pool = -1,
  max.cov.per.pool = 1e+06,
  min.indgeno.per.pop = -1,
  min.maf = -1,
  output.snp.values = FALSE,
  nsnp.per.bjack.block = 0,
  verbose = TRUE
)
```

Arguments

`x`	A pooldata object containing Pool-Seq information or a countdata object containing allele count information
`method`	Either "Anova" (default method as described in the manuscript) or "Identity" (relies on an alternative modeling consisting in estimating unbiased Probability of Identity within and across pairs of pools)
`min.cov.per.pool`	For Pool-Seq data (i.e., pooldata objects) only: minimal allowed read count (per pool). If at least one pool is not covered by at least min.cov.perpool reads, the position is discarded in the corresponding pairwise comparisons
`max.cov.per.pool`	For Pool-Seq data (i.e., pooldata objects) only: maximal allowed read count (per pool). If at least one pool is covered by more than min.cov.perpool reads, the position is discarded in the corresponding pairwise comparisons.
`min.indgeno.per.pop`	For allele count data (i.e., countdata objects) only: minimal number of overall counts required in each population. If at least one pop is not genotyped for at least min.indgeno.per.pop (haploid) individual, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio over all read counts for the reference allele over the read coverage) in the pairwise comparisons.

`output.snp.values`
: If TRUE, provide SNP-specific pairwise FST for each comparisons (may lead to a huge result object if the number of pools and/or SNPs is large)

`nsnp.per.bjack.block`
: Number of consecutive SNPs within a block for block-jackknife (default=0, i.e., no block-jackknife sampling)

`verbose`
: If TRUE extra information is printed on the terminal

Value

An object of class pairwisefst (see help(pairwisefst) for details)

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`,`genobaypass2pooldata` or `genoselestim2pooldata`. To generate coundata object, see `genobaypass2countdata` or `genotreemix2countdata`.

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 PairwiseFST=compute.pairwiseFST(pooldata)
```

`computeFST` *Compute Fst from Pool-Seq data or Count data*

Description

Compute Fst from Pool-Seq data or Count data

Usage

```
computeFST(
  x,
  method = "Anova",
  struct = NULL,
  weightpid = FALSE,
  nsnp.per.bjack.block = 0,
  sliding.window.size = 0,
  verbose = TRUE
)
```

Arguments

`x`	A pooldata object containing Pool-Seq information or countdata object containing allele counts information
`method`	Either "Anova" (default method) or "Identity" (relying on unbiased estimators of Probability of Identity within and across pairs of pools/populations)
`struct`	Vector of length equal to the number of pop. sample that give the pop. sample group name of index (i.e., structure)
`weightpid`	When method="Identity", if TRUE weighting averages of pop. Q1 and pairwise Q2 are performed (see eq. A46 and A47 in Hivert et al., 2018 for PoolSeq and Rousset 2007 for count data) to compute overall Q1 and Q2. If not, unweighted averages are performed.
`nsnp.per.bjack.block`	Number of consecutive SNPs within a block for block-jackknife (default=0, i.e., no block-jackknife sampling)
`sliding.window.size`	Number of consecutive SNPs within a window for multi-locus computation of Fst over sliding window with half-window size step (default=0, i.e., no sliding-window scan)
`verbose`	If TRUE extra information is printed on the terminal

Value

A list with the four following elements:

1. "FST": estimate of genome-wide Fst over all the populations. The element is a vector with 5 elements corresponding to i) the estimated value over all SNPs; ii) the block-jackknife mean; iii) the block-jackknife s.e.; iv) the lower; and v) the upper bound of the 95
2. "FSG": under the hierarchical Fst model (i.e., when struct vector is non-null); estimates estimate of genome-wide within-group differentiation (Fsg). The element is a vector with 5 elements corresponding to i) the estimated value over all SNPs; ii) the block-jackknife mean; iii) the block-jackknife s.e.; iv) the lower; and v) the upper bound of the 95
3. "FGT": under the hierarchical Fst model (i.e., when struct vector is non-null); estimates estimate of genome-wide between-group differentiation (Fgt). The element is a vector with 5 elements corresponding to i) the estimated value over all SNPs; ii) the block-jackknife mean; iii) the block-jackknife s.e.; iv) the lower; and v) the upper bound of the 95
4. "snp.Fstats": a data frame containing SNP-specific estimates of Fst and also under the hierarchical (i.e., when struct vector is non-null) SNP-specific estimates Fsg and Fgt
5. "snp.Q": a data frame containing SNP-specific estimates of Q1 (within-population) and Q2 (between-population) probability of identity and also under the hierarchical (i.e., when struct vector is non-null) SNP-specific estimates of Q3, the probability of identity between populations from different groups (under this model Q2 is then the Pid between populations from the same group).
6. "sliding.windows.fvalues" (if sliding.window.size>0): a 4 or 6 (under hierarchical Fst model) column data frame containing information on multi-locus Fst (and Fsg and Fgt under the hierarchical Fst model) computed for sliding windows of SNPs over the whole genome with

i) column with the chromosome/contig of origin of each window; ii) the mid-position of each window; iii) the cumulated mid-position of each window (to facilitate further plotting); iv) the estimated multi-locus Fst; and under the hierarchical Fst model v) the estimated multi-locus Fsg and ; vi) the estimated multi-locus Fgt

See Also

To generate pooldata object, see vcf2pooldata, popsync2pooldata,genobaypass2pooldata or genoselestim2pooldata. To generate coundata object, see genobaypass2countdata or genotreemix2countdata.

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 res.fst=computeFST(pooldata)
 res.hierfst=computeFST(pooldata,struct=c(rep("A",5),rep("B",7),rep("C",3)))
```

---

compute_blockDdenom *compute_blockDdenom*

Description

Compute the denominator of the Dstat for all quadruplet configuration and each block-jackknife block (if any) and overall SNPs (within or outside blocks)

Usage

```
.compute_blockDdenom(refcount, totcount, nblocks, block_id, verbose)
```

Arguments

refcount	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
totcount	Matrix of nsnpxnpop with total counts or read coverages
nblocks	Integer giving the number of block-jackknife blocs (may be 0 if no block-jackknife)
block_id	Integer vector of length nsnps with the (0-indexed) id of the block to which each SNP belongs (-1 for SNPs outside blocks)
verbose	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute the denominator of the Dstat for all quadruplet configuration and each block-jackknife block (if any) and overall SNPs (within or outside blocks)

Value

Return a matrix with nf4=(npops*(npops-1)/2)*((npops-2)*(npops-3)/2)/2 rows and nblocks+1 columns giving the mean Dstat-denominator (1-Q2ab)(1-Q2cd) for all quadruplet configuration and within each block-jackknife sample and over all SNPs (last column)

Examples

```
#
```

`compute_F3fromF2` *compute_F3fromF2*

Description

Compute all F3 from overall F2 values

Usage

```
.compute_F3fromF2(F2val, Hval, npops)
```

Arguments

`F2val`	Numeric vector of length nF2=(npop*(npop-1))/2 with all pairwise F2 estimates
`Hval`	Numeric vector of length npop with all within pop heterozygosity estimates
`npops`	Integer giving the number of populations

Details

Compute F3 and F3star estimates from F2 (and heterozygosities)

Value

Return a matrix of length nF3=npops*(npops-1)*(npops-2)/2 rows and 2 columns corresponding to the F3 and F3star estimates

Examples

```
#
```

`compute_F3fromF2samples`
compute_F3fromF2samples

Description

Compute all F3 from F2 values obtained from each block-jackknife bloc

Usage

```
.compute_F3fromF2samples(blockF2, blockHet, npops, verbose)
```

Arguments

`blockF2`	Numeric Matrix with nF2=(npop*(npop-1))/2 rows and nblocks columns matrix containing pairwise-pop F2 estimates for each block-jackknife sample (l.o.o.)
`blockHet`	Numeric Matrix with npop rows and nblocks columns containing all within pop heterozygosity estimates for each block-jackknife sample (l.o.o.)
`npops`	Integer giving the number of populations
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute F3 and F3star estimates and their s.e. based on block-jackknife estimates of all F2 (and heterozygosities)

Value

Return a matrix with nF3=npops*(npops-1)*(npops-2)/2 rows and four columns corresponding to the mean and the s.e. of F3 and the mean and s.e. of F3star

Examples

```
#
```

`compute_F4DfromF2samples`

compute_F4DfromF2samples

Description

Compute all F4 and Dstat from F2 values obtained from each block-jackknife bloc

Usage

```
.compute_F4DfromF2samples(blockF2, blockDenom, npops, verbose)
```

Arguments

`blockF2`	Numeric Matrix with nF2=(npop*(npop-1))/2 rows and nblocks columns matrix containing pairwise-pop F2 estimates for each block-jackknife sample (l.o.o.)
`blockDenom`	Numeric Matrix with nF4=(npops*(npops-1)/2)*((npops-2)*(npops-3)/2)/2 rows and nblocks containing the estimates of the denominator of Dstat (see compute_blockDdenom) for each block-jackknife sample (l.o.o.)
`npops`	Integer giving the number of populations
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute F4 and D estimates and their s.e. based on block-jackknife estimates of all F2 (and heterozygosities)

Value

Return a matrix with nF4=(npops*(npops-1)/2)*((npops-2)*(npops-3)/2)/2 rows and four columns corresponding to the mean and the s.e. of F4 and the mean and s.e. of Dstat

Examples

```
#
```

`compute_F4fromF2` *compute_F4fromF2*

Description

Compute all F4 from overall F2 and Q2 values

Usage

```
.compute_F4fromF2(F2val, npops)
```

Arguments

`F2val`	Numeric vector of length nF2=(npop*(npop-1))/2 with all pairwise F2 estimates
`npops`	Integer giving the number of populations

Details

Compute F4 from F2 (and heterozygosities)

Value

Return a vector of length nF4=(npops*(npops-1)/2) * ((npops-2)*(npops-3)/2) / 2 rows corresponding to all the F4 estimates for all possible configurations

Examples

```
#
```

`compute_F4fromF2samples` *compute_F4fromF2samples*

Description

Compute all F4 from F2 values obtained from each block-jackknife bloc

Usage

```
.compute_F4fromF2samples(blockF2, npops, verbose)
```

Arguments

`blockF2`	Numeric Matrix with nF2=(npop*(npop-1))/2 rows and nblocks columns matrix containing pairwise-pop F2 estimates for each block-jackknife sample (l.o.o.)
`npops`	Integer giving the number of populations
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute F4 estimates and their s.e. based on block-jackknife estimates of all F2 (and heterozygosities)

Value

Return a matrix with nF4=(npops*(npops-1)/2) * ((npops-2)*(npops-3)/2) / 2 rows and two columns corresponding to the mean and the s.e. of F4 estimates for all possible configurations

Examples

```
#
```

`compute_H1` *compute_H1*

Description

Compute (uncorrected) 1-Q1 for each block-jackknife block (if any) and over all the SNPs (i.e., either within or outside blocks)

Usage

```
.compute_H1(refcount, totcount, nblocks, block_id, verbose)
```

Arguments

`refcount`	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
`totcount`	Matrix of nsnpxnpop with total counts or read coverages
`nblocks`	Integer giving the number of block-jackknife blocs (may be 0 if no block-jackknife)
`block_id`	Integer vector of length nsnps with the (0-indexed) id of the block to which each SNP belongs (-1 for SNPs outside blocks)
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute all the (uncorrected) H1=1-Q1 for each block-jackknife block (if any) and overall SNPs (within or outside blocks). It is indeed more convenient to compute H1 (rather than Q1) to apply corrections afterwards within R function

Value

Return a matrix with npops rows and nblocks+1 column giving the mean H1 of each pop within each block and for all SNPs (last column)

Examples

```
#
```

`compute_Q2` *compute_Q2*

Description

Compute all Q2 for each block-jackknife block (if any) and overall SNPs (within or outside blocks)

Usage

```
.compute_Q2(refcount, totcount, nblocks, block_id, verbose)
```

Arguments

`refcount`	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
`totcount`	Matrix of nsnpxnpop with total counts or read coverages
`nblocks`	Integer giving the number of block-jackknife blocs (may be 0 if no block-jackknife)
`block_id`	Integer vector of length nsnps with the (0-indexed) id of the block to which each SNP belongs (-1 for SNPs outside blocks)
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute all Q2 for each block-jackknife block (if any) and overall SNPs (within or outside blocks).

Value

Return a matrix with npops*(npops-1)/2 and nblocks+1 column giving the mean Q2 of each pairwise pop comp. within each block and for all SNPs (last column)

Examples

```
#
```

---

compute_QmatfromF2samples

compute_QmatfromF2samples

Description

Compute the Qmat matrix (error covariance between all F2 and F3 measures) from F2 block-jackknife estimates

Usage

```
.compute_QmatfromF2samples(blockF2, npops, verbose)
```

Arguments

`blockF2`	Numeric Matrix with nF2=(npop*(npop-1))/2 rows and nblocks columns matrix containing pairwise-pop F2 estimates for each block-jackknife sample (l.o.o.)
`npops`	Integer giving the number of populations
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute the error covariance matrix Qmat (between all F2 and F3 measures) from F2 block-jackknife estimates (by recomuting all F3 for all blocks)

Value

Return the (nF2+nF3)*(nF2+nF3) error covariance (symmetric) matrix

Examples

```
#
```

`compute_snpFstAov` *compute_snpFstAov*

Description

Compute SNP-specific MSG, MSP and nc used to derived the Anova estimator of Fst for allele count or read count data (Pool-Seq)

Usage

```
.compute_snpFstAov(refcount, totcount, hapsize, verbose)
```

Arguments

`refcount`	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
`totcount`	Matrix of nsnpxnpop with total counts or read coverages
`hapsize`	Vector of length npop giving the haploid size of each pool (if one element <=0, counts are interpreted as count data)
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute SNP-specific Q1 and Q2 based on Anova estimator of Fst for allele count or read count data (Pool-Seq). For allele count data, the implemented estimator corresponds to that described in Weir, 1996 (eq. 5.2) For read (Pool-Seq) data, the implemented estimator corresponds to that described in Hivert et al., 2016

Value

Return a nsnpsx3 matrix with SNP-specific MSG, MSP and nc

Examples

```
#
```

`compute_snpHierFstAov` *compute_snpHierFstAov*

Description

Compute SNP-specific MSI, MSP, MSG, nc, nc_p and nc_pp used to derived the Anova estimator of hier. Fst for allele count or read count data (Pool-Seq)

Usage

```
.compute_snpHierFstAov(refcount, totcount, hapsize, popgrpidx, verbose)
```

Arguments

`refcount`	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
`totcount`	Matrix of nsnpxnpop with total counts or read coverages
`hapsize`	Vector of length npop giving the haploid size of each pool (if one element <=0, counts are interpreted as count data)
`popgrpidx`	Vector of length npop giving the index (coded from 0 to ngrp-1) of the group of origin
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute SNP-specific MSI, MSP, MSG, nc, nc_p and nc_pp used to derived the Anova estimator of hier. Fst for allele count or read count data (Pool-Seq)

Value

Return a nsnpsx6 matrix with SNP-specific MSI, MSP, MSG, nc, nc_p and nc_pp

Examples

```
#
```

`compute_snpQ1` *compute_snpQ1*

Description

Compute SNP-specific Q1 by averaging over all samples

Usage

```
.compute_snpQ1(refcount, totcount, weight, verbose)
```

Arguments

`refcount`	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
`totcount`	Matrix of nsnpxnpop with total counts or read coverages
`weight`	Vector of length npop giving the weighting scheme (w=1 for allele count data and w=poolsize/(poolsize-1) for PoolSeq data)
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute all the SNP-specific Q1 over all pop. samples (useful for Fst computation with method Identity).

Value

Return a vector of length nsnps with SNP-specific Q1

Examples

```
#
```

`compute_snpQ1onepop` *compute_snpQ1onepop*

Description

Compute SNP-specific Q1 for one pop

Usage

```
.compute_snpQ1onepop(refcount, totcount, weight)
```

Arguments

`refcount`	Vector of nsnp counts (genotype or reads) for the reference allele
`totcount`	Vector of nsnp total counts or read coverages
`weight`	Numeric (w=1 for allele count data and w=poolsize/(poolsize-1) for PoolSeq data)

Details

Compute SNP-specific Q1 for one pop. samples.

Value

Return a vector of length nsnps with SNP-specific Q1

Examples

```
#
```

compute_snpQ1rw *compute_snpQ1rw*

Description

Compute SNP-specific Q1 over all samples using weighting averages of pop. Q1 (eq. A46 in Hivert et al., 2018)

Usage

```
.compute_snpQ1rw(refcount, totcount, weight, sampsize, readcount, verbose)
```

Arguments

refcount	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
totcount	Matrix of nsnpxnpop with total counts or read coverages
weight	Vector of length npop giving the weighting scheme (w=1 for allele count data and w=poolsize/(poolsize-1) for PoolSeq data)
sampsize	Vector of length npop giving the haploid sample size (not used for count data)
readcount	Logical (if TRUE PoolSeq data assumed i.e. weights depending on haploid size, otherwise weights depend on total counts)
verbose	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute all the SNP-specific Q1 over all pop. samples using weighting averages of pop. Q1 as in eq. A46 of Hivert et al., 2018 (useful for Fst computation with method Identity).

Value

Return a vector of length nsnps with SNP-specific Q1

Examples

```
#
```

`compute_snpQ2` *compute_snpQ2*

Description

Compute SNP-specific Q2 by averaging over all pairs of samples

Usage

```
.compute_snpQ2(refcount, totcount, pairs, verbose)
```

Arguments

`refcount`	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
`totcount`	Matrix of nsnpxnpop with total counts or read coverages
`pairs`	Matrix of npoppairsx2 giving the index for all the pairs of pops included in the computation
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute all the SNP-specific Q2 over all pop. pairs (useful for Fst computation with method Identity).

Value

Return a vector of length nsnps with SNP-specific Q2

Examples

```
#
```

`compute_snpQ2onepair` *compute_snpQ2onepair*

Description

Compute SNP-specific Q2 for a single pair of samples

Usage

```
.compute_snpQ2onepair(refcount1, refcount2, totcount1, totcount2)
```

Arguments

`refcount1`	Vector of count (genotype or reads) for the reference allele in the first sample
`refcount2`	Vector of count (genotype or reads) for the reference allele in the second sample
`totcount1`	Vector of total count or read coverages in the first sample
`totcount2`	Vector of total count or read coverages in the second sample

Details

Compute SNP-specific Q2 for a single pair of samples

Value

Return a vector of length nsnps with SNP-specific Q1

Examples

```
#
```

`compute_snpQ2rw` *compute_snpQ2rw*

Description

Compute SNP-specific Q2 by averaging over all pairs of samples using weighting averages of pairwise Q2 (eq. A47 in Hivert et al., 2018)

Usage

```
.compute_snpQ2rw(refcount, totcount, pairs, sampsize, readcount, verbose)
```

Arguments

`refcount`	Matrix of nsnpxnpop with counts (genotype or reads) for the reference allele
`totcount`	Matrix of nsnpxnpop with total counts or read coverages
`pairs`	Matrix of npoppairsx2 giving the index for all the pairs of pops included in the computation
`sampsize`	Vector of length npop giving the haploid sample size (not used for count data)
`readcount`	Logical (if TRUE PoolSeq data assumed i.e. weights depending on haploid size, otherwise weights depend on total counts)
`verbose`	Logical (if TRUE progression bar is printed on the terminal)

Details

Compute SNP-specific Q2 by averaging over all pairs of samples using weighting averages of pairwise Q2 (eq. A47 in Hivert et al., 2018) (useful for Fst computation with method Identity).

Value

Return a vector of length nsnps with SNP-specific Q2

Examples

```
#
```

`countdata-class` *S4 class to represent a Count data set.*

Description

S4 class to represent a Count data set.

Slots

`npops` The number of populations

`nsnp` The number of SNPs

`refallele.count` A matrix (nsnp rows and npops columns) with the allele counts for the reference allele

`total.count` A matrix (nsnp rows and npops columns) with the total number of counts (i.e., twice the number of genotyped individual for diploid species and autosomal markers)

`snp.info` A data frame (nsnp rows and 4 columns) detailing for each SNP, the chromosome (or scaffold), the position, Reference allele name and Alternate allele name (if available)

`popnames` A vector of length npops with the corresponding population names

See Also

To generate countdata object, see `genobaypass2countdata` and `genotreemix2countdata`

`countdata.subset` *Create a subset of a countdata object that contains count data as a function of pop or SNP indexes*

Description

Create a subset of a countdata object that contains count data as a function of pop or SNP indexes

Usage

```
countdata.subset(
  countdata,
  pop.index = 1:countdata@npops,
  snp.index = 1:countdata@nsnp,
  min.indgeno.per.pop = -1,
  min.maf = -1,
  return.snp.idx = FALSE,
  verbose = TRUE
)
```

Arguments

`countdata`	A countdata object containing Allele count information
`pop.index`	Indexes of the pops (at least two), that should be selected to create the new pooldata object (default=all the pops)
`snp.index`	Indexes of the SNPs (at least two), that should be selected to create the new pooldata object (default=all the SNPs)
`min.indgeno.per.pop`	Minimal number of overall counts required in each population. If at least one pop is not genotyped for at least min.indgeno.per.pop (haploid) individual, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio overall counts for the reference allele over the overall number of (haploid) individual genotyped)
`return.snp.idx`	If TRUE, the row.names of the snp.info slot of the returned pooldata object are named as "rsx" where x is the index of SNP in the initial pooldata object (default=FALSE)
`verbose`	If TRUE return some information

Details

This function allows subsetting a pooldata object by selecting only some pools and/or some SNPs (e.g., based on their position on the genome). Additional filtering steps on SNPs can be carried out on the resulting subset to discard SNP with low polymorphism or poorly or too highly covered. In addition, coverage criteria can be applied on a per-pool basis with the cov.qthres.per.pool argument. 'more specific SNP selection based on their positions on the genome or their characteristics. For instance if qmax=0.95, a position is discarded if in a given pool it has a number of reads higher than the 95-th percentile of the empirical coverage distribution in this same pool (defined over the SNPs selected by snp.index). Similarly, if qmax=0.05, a position is discarded if in a given pool it has a number of reads lower than the 5-th percentile of the empirical coverage distribution in this same pool. This mode of selection may be more relevant when considering pools with heterogeneous read coverages.

Value

A countdata object with 6 elements:

1. "refallele.count": a matrix (nsnp rows and npops columns) with the allele counts for the reference allele
2. "total.count": a matrix (nsnp rows and npops columns) with the total number of counts (i.e., twice the number of genotyped individual for diploid species and autosomal markers)
3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele taken as reference in the refallele.count matrix (3rd column); and the alternative allele (4th column)
4. "popnames": a vector of length npops containing the names of the pops
5. "nsnp": a scalar corresponding to the number of SNPs
6. "npops": a scalar corresponding to the number of populations

See Also

To generate countdata object, see genobaypass2countdata, genotreemix2countdata

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
pooldata2genobaypass(pooldata=pooldata,writing.dir=tempdir())
##NOTE: This example is just for the sake of illustration as it amounts to
##interpret read count as allele count which must not be done in practice!
countdata=genobaypass2countdata(genobaypass.file=paste0(tempdir(),"/genobaypass"))
subset.by.snps=countdata.subset(countdata,snp.index=10:100)
subset.by.pops.and.snps=countdata.subset(countdata,pop.index=c(1,2),snp.index=10:100)
```

---

countdata2genobaypass *Convert a countdata object into BayPass input files.*

Description

Convert a countdata object into BayPass allele count file. A file containing SNP details is also printed out. Options to generate sub-samples (e.g., for large number of SNPs) are also available.

Usage

```
countdata2genobaypass(
  countdata,
  writing.dir = getwd(),
  prefix = "",
  subsamplesize = -1,
  subsamplingmethod = "thinning"
)
```

Arguments

`countdata`	A countdata object
`writing.dir`	Directory where to create the files (e.g., set writing.dir=getwd() to copy in the current working directory)
`prefix`	Prefix used for output file names
`subsamplesize`	Size of the sub-samples. If <=1 (default), all the SNPs are considered in the output
`subsamplingmethod`	If sub-sampling is activated (argument subsamplesize), define the method used for subsampling that might be either i) "random" (A single data set consisting of randmly chosen SNPs is generated) or ii) "thinning", sub-samples are generated by taking SNPs one every nsub=floor(nsnp/subsamplesize) in the order of the map (a suffix ".subn" is added to each sub-sample files where n varies from 1 to nsub).

Value

Files containing allele count (in BayPass format) and SNP details (as in the snp.info matrix from the countdata object)

See Also

To generate countdata object, see `genotreemix2countdata`, `genobaypass2countdata`

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 ##NOTE: This example is just for the sake of illustration as it amounts to
 ##interpret read count as allele count which must not be done in practice!
 countdata=genobaypass2countdata(genobaypass.file=paste0(tempdir(),"/genobaypass"))
 countdata2genobaypass(countdata=countdata,writing.dir=tempdir())
```

`data.merge` *Merge two pooldata or countdata objects*

Description

Merge two pooldata or countdata objects

Usage

```
data.merge(x1, x2, fake.pool.size = 1e+06, verbose = TRUE)
```

Arguments

`x1`	First pooldata or countdata object to merge
`x2`	Second pooldata or countdata object to merge
`fake.pool.size`	Specifies the haploid sample size used when merging a ‘countdata‘ object with a ‘pooldata‘ object to create a pseudo pooldata object containing all samples (default = 1e6), see details.
`verbose`	If TRUE return some information

Details

This function merges two objects of class ‘pooldata‘ and/or ‘countdata‘, automatically checking their structure for consistency. The merging behavior depends on the relationship between sample names and SNP identifiers:

1. Merging different samples (same SNPs): If SNP names are identical but (pool or population) sample names differ, the function merges data from the distinct samples into a single ‘pooldata‘ or ‘countdata‘ object that includes all samples.

2. Merging different SNPs (same sample): If sample names are identical but SNP names differ, the SNP data from each object are merged for each shared sample, effectively combining the variant information into one object.

3. Merging a ‘countdata‘ object with a ‘pooldata‘ object: In this case, the function returns a ‘pooldata‘ object. Allele counts from the ‘countdata‘ object are converted into pseudo read counts. To ensure compatibility, the haploid sample size for the sample originally contained in the ‘countdata‘ object is set to the value specified by the ‘fake.haploid.size‘ argument (default = `1e6`). Setting this value to a very large number (as in the default) ensures that each read count is treated as originating from a distinct haploid individual— mimicking Pool-Seq data where read coverage is much lower than the haploid sample size. This effectively disables Pool-Seq-specific bias corrections in downstream statistical analyses. Importantly, when merging objects of different types, only SNP-level merging is permitted. In this context, population samples are indeed expected to be necessarily distinct (at least in terms of effective haploid sample sizes).

Value

A new ‘pooldata‘ or ‘countdata‘ object, depending on the input types.

See Also

To obtain description of the ‘countdata‘ and ‘pooldata‘ objects, see `countdata` and `pooldata`

Examples

```
make.example.files(writing.dir=tempdir())
pooldata1=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
pooldata2=pooldata1
#Merge pooldata1 and pooldata2 by SNP
pooldata2@poolnames=paste0(pooldata2@poolnames,"_2") #pool names must be different
data.merged=data.merge(pooldata1,pooldata2)
#Merge pooldata1 and pooldata2 by POP
pooldata2=pooldata1
```

```
pooldata2@snp.info[,1]=paste0(pooldata2@snp.info[,1],"_2") #SNP info must be different
data.merged=data.merge(pooldata1,pooldata2)
#Merge pooldata1 with a countdata object
#create a countdata object (NOTE: This example is just for the sake of illustration)
pooldata2genobaypass(pooldata=pooldata1,writing.dir=tempdir())
countdata=genobaypass2countdata(genobaypass.file=paste0(tempdir(),"/genobaypass"))
countdata@snp.info=pooldata1@snp.info
countdata@popnames=paste0(countdata@popnames,"_2") #pop names must be different
data.merged=data.merge(pooldata1,countdata)
```

extract_allele_names *extract_allele_names*

Description

Extract the alleles from the REF and ALT fields

Usage

```
.extract_allele_names(allele_info, allele_idx)
```

Arguments

allele_info	a character string vector (concatenated REF and ALT field of the vcf)
allele_idx	Matrix with indexes of the two alleles of interest for the different markers

Details

Extract the alleles from the REF and ALT fields

Value

Return a matrix with the two alleles after parsing the alleles info

Examples

```
.extract_allele_names(c("A,C","A,C,T"),rbind(c(1,2),c(1,3)))
```

`extract_nonvscan_counts` *extract_nonvscan_counts*

Description

Extract counts from vcf produced by other caller than VarScan (e.g., bcftools, FreeBayes, GATK)

Usage

```
.extract_nonvscan_counts(vcf_data, nb_all, ad_idx, min_rc)
```

Arguments

`vcf_data`	a matrix of String containing count information
`nb_all`	a vector containing the number of alleles for the different markers
`ad_idx`	the index of the FORMAT AD field
`min_rc`	Minimal allowed read count per base (same as min.rc option in `vcf2pooldata`)

Details

Extract VarScan counts and return read counts for the reference and alternate allele

Value

A numeric matrix of read count with nsnp rows and 2*npools+6 columns. The first npools columns consist of read count for the reference allele, columns npools+1 to 2*npools consist of read coverage. The last 6 columns correspond to the index of the two most frequent alleles (idx_all1 and idx_all2) and their count (cnt_all1 and cnt_all2); the min_rc filtering criterion and count of variant (cnt_bases) other than two first most frequent. The min_rc crit is set to -1 for polymorphisms with more than 2 alleles and with the third most frequent alleles having more than min_rc count

Examples

```
.extract_nonvscan_counts(rbind(c("0/0:20,0","1/1:1,18"),c("0/2:12,1,15","1/1:27,1,0")),c(2,3),2,0)
.extract_nonvscan_counts(rbind(c("0/0:20,0","1/1:1,18"),c("0/2:12,1,15","1/1:27,1,0")),c(2,3),2,2)
```

`extract_vscan_counts` *extract_vscan_counts*

Description

Extract VarScan counts

Usage

```
.extract_vscan_counts(vcf_data, ad_idx, rd_idx)
```

Arguments

`vcf_data`	a matrix of String containing count information in VarScan format
`ad_idx`	the index of the FORMAT AD field
`rd_idx`	the index of the FORMAT RD field

Details

Extract VarScan counts and return read counts for the reference and alternate allele. For VarScan generated vcf, SNPs with more than one alternate allele are discarded (because only a single count is then reported in the AD fields) making the min.rc unavailable (of vcf2pooldata). The VarScan –min-reads2 option might replace to some extent the min.rc functionality although SNP where the two major alleles in the Pool-Seq data are different from the reference allele (e.g., expected to be more frequent when using a distantly related reference genome for mapping) will be disregarded.

Value

A numeric matrix of read count with nsnp rows and 2*npools columns. The first npools columns consist of read count for the reference allele (RD), columns npools+1 to 2*npools consist of read coverage (RD+AD)

Examples

```
.extract_vscan_counts(rbind(c("0/0:0:20","1/1:18:1"),c("0/1:12:15","1/1:27:2")),3,2)
```

find.tree.popset *Find sets of populations that may used as scaffold tree*

Description

Find sets of populations that may used as scaffold tree

Usage

```
find.tree.popset(
  fstats,
  f3.zcore.threshold = -1.65,
  f4.zscore.absolute.threshold = 1.96,
  excluded.pops = NULL,
  nthreads = 1,
  verbose = TRUE
)
```

Arguments

`fstats`	Object of class fstats containing estimates of fstats (see the function compute.fstats)
`f3.zcore.threshold`	The significance threshold for Z-score of formal test of admixture based on the F3-statistics (default=-2)
`f4.zscore.absolute.threshold`	The significance threshold for |Z-score| of formal test of treeness based on the F4-statistics (default=2)
`excluded.pops`	Vector of pop names to be exclude from the exploration
`nthreads`	Number of available threads for parallelization of some part of the parsing (default=1, i.e., no parallelization)
`verbose`	If TRUE extra information is printed on the terminal

Details

The procedure first discards all the populations P that shows a significant signal of admixture with a Z-score for F3 statistics of the form F3(P;Q,R) < f3.zscore.thresholds. It then identifies all the sets of populations that pass the F4-based treeness with themselves. More precisely, for a given set E containing n populations, the procedure ensure that all the n(n-1)(n-2)(n-3)/8 possible F4 quadruplets have a |Z-score|<f4.zscore.absolute.threshold. The function aims at maximizing the size of the sets.

Value

A list with the following elements:

1. "n.sets": The number of sets of (scaffold) unadmixed populations identified

2. "set.size": The number of populations included in each set
3. "pop.sets": A character matrix of n.sets rows and set.size columns giving for each set identified the names of the included populations.
4. "Z_f4.range": A matrix of n.sets rows and 2 columns reported for each set the range of variation (min and max value) of the absolute F4 Z-scores for the quadruplets passing the treeness test. More precisely, for a given set consisting of n=set.size populations, a total of n(n-1)(n-2)(n-3)/8 quadruplets can be formed. Yet, any set of four populations A, B, C and D is represented by three quadruplets A,B;C,D (or one of its seven other equivalent combinations formed by permuting each pairs); A,C;B,D (or one of its seven other equivalent combinations) and A,D;B,C (or one of its seven other combinations). Among these three, only a single quadruplet is expected to pass the treeness test (i.e., if the correct unrooted tree topology is (A,C;B,D), then the absoulte value of the Z-scores associated to F4(A,B;C,D) and F4(A,D;B,C) or their equivalent will be high.
5. "passing.quadruplets": A matrix of n.sets rows and set.size columns reporting for each sets the n(n-1)(n-2)(n-3)/24 quadruplets that pass the treeness test (see Z_f4.range detail).

See Also

see `compute.fstats`.

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
res.fstats=compute.fstats(pooldata,nsnp.per.bjack.block = 50)
#NOTE: toy example (in practice nsnp.per.bjack.block should be higher)
popsets=find.tree.popset(res.fstats,f3.zcore.threshold=-3)
```

`find_indelneighbor_idx` *find_indelneighbor_idx*

Description

Search for the closest indels of the markers

Usage

```
.find_indelneighbor_idx(contig, position, indels_idx, min_dist, indels_size)
```

Arguments

`contig`	a character string vector corresponding to the CHR field value of the vcf for the markers
`position`	an integer vector corresponding to the POSITION value for the markers
`indels_idx`	vector of (0-indexed) indices of indels
`min_dist`	same as min.dist.from.indels option in `vcf2pooldata`
`indels_size`	size of the indels (associated to indels_idx)

Details

Identify if the SNPs are close to an indel

Value

Return a vector consisting of 1 (if the marker is close to an indel) or 0 (if not)

Examples

```
.find_indelneighbor_idx(c("chr1","chr1","chr1"),c(1000,1004,1020),1,5,2)
```

`fit.graph` *Estimate parameters of an admixture graph*

Description

Estimate parameters of an admixture graph

Usage

```
fit.graph(
  graph.params,
  Q.lambda = 0,
  eps.admix.prop = 1e-06,
  edge.fact = 1000,
  admix.fact = 100,
  compute.ci = F,
  drift.scaling = F,
  outfileprefix = NULL,
  verbose = TRUE
)
```

Arguments

`graph.params`	An object of class graph.params containing graph information and relevant Fstats estimates (see the function generate.graph.params)
`Q.lambda`	A scalar (usually small) to add to the diagonal elements of the error covariance matrix of fstats estimates (may improve numerical stability of its decomposition for large number of populations)
`eps.admix.prop`	A scalar defining admixture proportion domain (eps.admix.prop vary between eps.admix.prop and 1-eps.admix.prop)
`edge.fact`	The multiplying factor of edges length in graph representation
`admix.fact`	The multiplying factor of admixture proportion in graph representation
`compute.ci`	Derive 95% Confidence Intervals for the parameters of the admixture graph (edge lengths and admixture rates)

`drift.scaling`	If TRUE scale edge lengths in drift units (require estimates of leave heterozygosities)
`outfileprefix`	The prefix of the dot file that will represent the graph (with extension ".dot"). If NULL, no graph file generated
`verbose`	If TRUE extra information is printed on the terminal

Details

Let f represent the n-length vector of basis target (i.e., observed) F2 and F3 statistics and $g(e;a) = X(a) * e$ the vector of their expected values given the vector of graph edges lengths e and the incidence matrix $X(a)$ that depends on the structure of the graph and the admixture rates a (if there is no admixture in the graph, $X(a)$ only contains 0 or 1). The function attempts to find the e and a graph parameter values that minimize a cost (score of the model) defined as $S(e;a) = (f - g(e;a))'Q^{-1}(f - g(e;a))$. Assuming $f\ N(g(e;a), Q)$ (i.e., the observed f-statistics vector is multivariate normal distributed around an expected g vector specified by the admixture graph and a covariance structure empirically estimated), $S = -2log(L) - K$ where L is the likelihood of the fitted graph and $K = n * log(2 * pi) + log(|Q|)$. Also, for model comparison purpose, a standard BIC is then derived from S as $BIC = S + p * log(n) - K$ (p being the number of graph parameters, i.e., edge lengths and admixture rates). As mentioned by Patterson et al. (2012), the score $S(e;a)$ is quadratic in edge lengths e given a. The function uses the Lawson-Hanson non-negative linear least squares algorithm implemented in the nnls function (package nnls) to estimate e (subject to the constraint of positive edge lengths) by finding the vector e that minimize $S(e;a) = (f - X(a) * e)'Q^{-1}(f - X(a) * e) = ||G * f - G * X(a) * e||$ (where G results from the Cholesky decomposition of Q^{-1}, i.e., $Q^{-1} = G'G$). Note that the *Q.lambda* argument may be used to add a small constant (e.g., $1e - 4$) to the diagonal elements of Q to avoid numerical problems (see Patterson et al., 2012). Yet *Q.lambda* is always disregarded when computing the final score S and BIC. Minimization of $S(e;a)$ is thus reduced to the identification of the admixture rates (a vector) which is performed using the L-BFGS-B method (i.e., Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm with box constraints) implemented in the optim function (stats package). The *eps.admix.prop* argument allows specifying the lower and upper bound of the admixture rates to *eps.admix.prop* and *1-eps.admix.prop* respectively. Scaling of the edges lengths in drift units (i.e., in units of $t/2N$ where t is time in generations and N is the effective population size) is performed as described in Lipson et al. (MBE, 2013) by dividing the estimated edges lengths by half the estimated heterozygosity of their parental nodes (using the property $hp = hc+2e(C,P)$ where hp and hc are the heterozygosities of a child C and its parent P node and $e(C,P)$ is the estimated length of the branch relating C and P. Finally, if compute.ci=TRUE, a (rough) 95% confidence intervals is computed using a bisection method (with a $1e - 4$ precision) for each parameters in turn (all others being set to their estimated value). Note that 95% CI are here defined as the set of values associated to a score S such that $Sopt < S < Sopt + 3.84$ (where $Sopt$ is the optimized score), i.e., with a likelihood-ratio test statistic with respect to the fitted values < 3.84 (the 95% threshold of a one ddl Chi-square distribution).

Value

An object of class fitted.graph (see help(fitted.graph) for details)

See Also

To generate a graph.params object, see `generate.graph.params`. The fitted graph may be plotted directly using plot that calls grViz() function and the resulting fitted fstats may be compared to the estimated ones with `compare.fitted.fstats`.

`fitted.graph-class` *S4 class to represent a population tree or admixture graph and its underlying fitted parameter.*

Description

S4 class to represent a population tree or admixture graph and its underlying fitted parameter.

Details

The dot.graph element allows to plot the graph using grViz() from the DiagrammeR package or with the dot program after writing the files (e.g., dot -Tpng inputgraph.dot in terminal). Note that the dot file may be customized (e.g., to change leave color, parameter names...).

Slots

`graph` The graph in 3 column format originated from the fitted graph.params object

`dot.graph` The fitted graph in dot format

`score` the score of the model (squared Mahalanobis distance between the observed and fitted basis F-statistics vectors)

`bic` The Bayesian Information Criterion associated to the model

`fitted.outstats` a matrix containing the target values of the fstats, the fitted values and the Z-score measuring the deviation of the fitted values from the target values in units of standard errors (i.e., Z=(fitted.value-target.value)/se(target.value))

`edges.length` a vector containing the estimated edges.length. Note finally, that the (two) edges coming from the roots are assumed of equal length (i.e., unrooted branch) as these are non-identifiable by the method.

`edges.length.scaled` If drift.scaling=TRUE, the estimated edges.length in units of t/2N

`edges.length.ci` A matrix with two columns (or four columns if drift scaled lengths are computed) containing for each edge length (in a row) the 95% CI lower and higher bounds (columns 3 and 4 containing 95% CI lower and higher bounds of drift scaled lengths, if any)

`admix.prop` a vector containing the estimated admixture proportions (if any)

`admix.prop.ci` a matrix with two columns containing for each admixture proportion (in a row) the 95% CI lower and higher bounds

`nodes.het` The estimated heterozygosities for all nodes (if available; see drift.scaling argument in fit.graph)

`fitted.f2.mat` the matrix of all the fitted F2 statistics (obtained from fitted admixture graph parameter values) from which all the fitted fstats can be derived.

`optim.results` list containing results of the optim call

See Also

To generate fitted.graph object, see `fit.graph`.

`fstats-class` *S4 class to represent fstats results obtained with computeFstats.*

Description

S4 class to represent fstats results obtained with computeFstats.

Slots

`f2.values` A data frame with npop(npop-1)/2 rows and 1 (or 3 if blockjackknife is TRUE) columns containing estimates of the f2-statistics over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.)

`fst.values` A data frame with npop(npop-1)/2 rows and 1 (or 3 if blockjackknife is TRUE) columns containing estimates of the scaled f2.values (same as obtained with compute.pairwiseFST with method="Identity") over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.). The F2 scaling factor is equal to 1-Q2 (where Q2 is the AIS probability between the two populations)

`f3.values` A data frame with npops(npops-1)(npops-2)/2 rows and 1 (or 4 if blockjackknife is TRUE) columns containing estimates of the f3-statistics over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.) and Z-score measuring the deviation of the f3-statistics from 0 in units of s.e.

`f3star.values` A data frame with npops(npops-1)(npops-2)/2 rows and 1 (or 4 if blockjackknife is TRUE) columns containing estimates of the scaled f3-statistics over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.) and Z-score measuring the deviation of the f3-statistics from 0 in units of s.e. The F3 scaling factor is equal to 1-Q1 (where Q1 is the AIS probability within the target population, i.e., population C for F3(C;A,B))

`f4.values` A data frame with npops(npops-1)(npops-2)(npops-3)/8 rows and 1 (or 4 if blockjackknife is TRUE) columns containing estimates of the f4-statistics over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.) and Z-score measuring the deviation of the f4-statistics from 0 in units of s.e.

`Dstat.values` A data frame with npops(npops-1)(npops-2)(npops-3)/8 rows and 1 (or 4 if blockjackknife is TRUE) columns containing estimates of the D-statistics (scaled f4-statistics) over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.) and Z-score measuring the deviation of the f3-statistics from 0 in units of s.e. For a given quadruplet (A,B;C,D), the parameter D corresponds to F4(A,B;C,D) scaled by (1-Q2(A,B))*(1-Q2(C,D)) where Q2(X,Y) is the is the AIS probability between the X and Y populations.

`F2.bjack.samples` If blockjackknife=TRUE and options return.F2.blockjackknife.samples is actived in compute.fstats, an array of dimension (npop x npop x nblocks) in an admixtools2 compatible format `comparisons` A list containing matrices with population names associated to the different test comparisons (e.g., the "F2" elements of the list is a npop(npop-1)/2 rows x 2 columns with each row containing the name of the two populations compared)

`Q.matrix` The estimated error covariance matrix for all the F2 and F3 estimates (required by graph fitting functions to compute graph scores)

`heterozygosities` A data frame with npop rows and 1 (or 3 if blockjackknife is TRUE) columns containing estimates of the within population heterozygosities (1-Q1) over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.)

`divergence` A data frame with npop(npop-1)/2 rows and 1 (or 3 if blockjackknife is TRUE) column(s) containing estimates of each population pairwise (absolute) divergence (1-Q2) over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.). This statistic is related to dXY (a.k.a. PiXY) but it is computed on the ascertained SNPs that were included in the original pooldata or countdata objects.

`pairwise.fst` A npop x npop (symmetric) matrix containing the pairwise-population Fst estimates (same as in the fst.values object) that may directly be visualized with e.g. heatmap function or used with a clustering function (e.g., hclust).

`pairwise.div` A npop x npop (symmetric) matrix containing the pairwise-population divergence (1-Q2) estimates (same as in the fst.values object) that may directly be visualized with e.g. heatmap function or used with a clustering function (e.g., hclust).

`blockjacknife` A logical indicating whether block-jackknife estimates of standard errors are available (TRUE) or not (FALSE)

See Also

To generate pairwise object, see `compute.pairwiseFST`

`generate.graph.params` *Generate a graph parameter object to fit admixture graph to observed fstats*

Description

Generate a graph parameter object to fit admixture graph to observed fstats

Usage

```
generate.graph.params(
  graph,
  fstats = NULL,
  popref = NULL,
  outfileprefix = NULL,
  verbose = TRUE
)
```

Arguments

`graph`	A three columns matrix containing graph information in a simple format (see details)
`fstats`	A fstats object containing estimates of fstats
`popref`	Reference population of the fstats basis used to fit the graph.
`outfileprefix`	The prefix of the dot file that will represent the graph (with extension ".dot"). If NULL, no graph file generated
`verbose`	If TRUE some information is printed on the terminal

Details

The graph needs to be specified by a three column (character) matrix corresponding for each edge (wether admixed or not) to i) the child node; ii) the parent node; iii) the admixture proportion. For non-admixed edge, the third column must be blank. An admixed node should be referred two times as a child node with two different parent node and two different admixture proportions coded as alpha and (1-alpha) (Note that the parentheses are mandatory) if alpha is the name of the admixture proportion. The root is automatically identified as a node only present in the parent node column. Several checks are made within the function but it is recommended to check the graph by plotting the resulting dot file named [outfileprefix].dot using for instance the grViz() from the DiagrammeR package that may be called directly with plot or with the dot program (e.g., dot -Tpng inputgraph.dot in terminal). Note that the dot file may be easily customized (e.g., to change leave color, parameter names...). The fstats object should be of class fstats (see help(fstats) for details) containing estimates of F2 and F3 statistics and block jackknife as generated with the `compute.fstats` function with computeF3 set to TRUE. If no fstats object is provided, only graph parameters will be generated.

Value

An object of class graph.params (see help(graph.params) for details)

See Also

The object may be used to estimate graph parameters with the function `fit.graph` or to generate files for the qpGraph software with `graph.params2qpGraphFiles`. See also `graph.params2symbolic.fstats` to obtain symbolic representation of Fstats.

Examples

```
graph=rbind(c("P1","P7",""),c("P2","s1",""),c("P3","s2",""),c("P6","S",""),
            c("S","s1","a"),c("S","s2","(1-a)"),c("s2","P8",""),c("s1","P7",""),
            c("P4","P9",""),c("P5","P9",""),c("P7","P8",""),
            c("P8","R",""),c("P9","R",""))
graph.params=generate.graph.params(graph)
plot(graph.params)
##NOTE: this calls grViz from DiagrammeR which cannot easily be plotted
#within pdf or other device. To that end the easiest is to output
#the graph in a dot file (using the outfileprefix argument) and
#then to use the dot program out of R in a terminal: dot -Tpng inputgraph.dot
```

generate.jackknife.blocks
Generate block coordinates for block-jackknife

Description

Generate block coordinates for block-jackknife

Usage

```
generate.jackknife.blocks(x, nsnp.per.bjack.block, verbose = TRUE)
```

Arguments

`x`	A pooldata or countdata object containing SNP positions (snp.info slot)
`nsnp.per.bjack.block`	Number of consecutive SNPs of each block-jackknife block
`verbose`	If TRUE extra information is printed on the terminal

Value

A list with the two following elements:

1. "blocks.det": A matrix with three columns containing for each identified block (in row) the index of the start SNP, the index of the end SNP and the block Size in bp
2. "snp.block.id": A vector containing the blocks assigned to each SNP eligible for block-Jacknife (non eligible SNPs ares assigned NA)
3. "nblocks": A scalar corresponding to the number of blocks
4. "nsnps": Number of SNPs eligible for block-jackknife 'i.e., included in one block

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
bjack.blocks=generate.jackknife.blocks(pooldata,nsnp.per.bjack.block=50)
```

generateF3names *generateF3names*

Description

Generate all names for F3 stats (same order as computation)

Usage

```
.generateF3names(popnames)
```

Arguments

`popnames`	String vector with the names of all the pops

Details

Generate all the npops*(npops-1)*(npops-2)/2 names for F3 stats (same order as computation)

Value

Return a string matrix with 4 columns including the complete F3 configuration names (of the form Px;P1,P2), and the names of each pop involved in the configuration

Examples

```
#
```

generateF4names *generateF4names*

Description

Generate all names for F4 stats (same order as computation)

Usage

```
.generateF4names(popnames)
```

Arguments

`popnames`	String vector with the names of all the pops

Details

Generate all the nf4=(npops*(npops-1)/2)*((npops-2)*(npops-3)/2)/2 names for F4 stats (same order as computation)

Value

Return a string matrix with 5 columns including the complete F4 configuration names (of the form P1,P2;P3,P4), and the names of each pop involved in the configuration

#

`genobaypass2countdata` *Convert BayPass allele count input files into a coundata object*

Description

Convert BayPass allele count input files into a coundata object

Usage

```
genobaypass2countdata(
  genobaypass.file = "",
  snp.pos = NA,
  popnames = NA,
  min.indgeno.per.pop = -1,
  min.maf = -1,
  verbose = TRUE
)
```

Arguments

`genobaypass.file`	The name (or a path) of the BayPass allele count file (see the BayPass manual https://forge.inrae.fr/mathieu.gautier/baypass_public/)
`snp.pos`	An optional two column matrix with nsnps rows containing the chromosome (or contig/scaffold) of origin and the position of each markers
`popnames`	A character vector with the names of pool
`min.indgeno.per.pop`	Minimal number of overall counts required in each population. If at least one pop is not genotyped for at least min.indgeno.per.pop (haploid) individual, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio overall counts for the reference allele over the overall number of (haploid) individual genotyped)
`verbose`	If TRUE extra information is printed on the terminal

Details

Information on SNP position is only required for some graphical display or to carried out block-jacknife sampling estimation of confidence intervals. If no mapping information is given (default), SNPs will be assumed to be ordered on the same chromosome and separated by 1 bp. As blocks are defined with a number of consecutive SNPs (rather than a length), the latter assumption has actually no effect (except in the reported estimated block sizes in Mb).

Value

A countdata object containing 6 elements:

1. "refallele.count": a matrix (nsnp rows and npops columns) with the allele counts for the reference allele
2. "total.count": a matrix (nsnp rows and npops columns) with the total number of counts (i.e., twice the number of genotyped individual for diploid species and autosomal markers)
3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele taken as reference in the refallele.count matrix (3rd column); and the alternative allele (4th column)
4. "popnames": a vector of length npops containing the names of the pops
5. "nsnp": a scalar corresponding to the number of SNPs
6. "npops": a scalar corresponding to the number of populations

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
pooldata2genobaypass(pooldata=pooldata,writing.dir=tempdir())
##NOTE: This example is just for the sake of illustration as it amounts
##to interpret read count as allele count which must not be done in practice!
countdata=genobaypass2countdata(genobaypass.file=paste0(tempdir(),"/genobaypass"))
```

---

genobaypass2pooldata *Convert BayPass read count and haploid pool size input files into a pooldata object*

Description

Convert BayPass read count and haploid pool size input files into a pooldata object

Usage

```
genobaypass2pooldata(
  genobaypass.file = "",
  poolsize.file = "",
  snp.pos = NA,
  poolnames = NA,
  min.cov.per.pool = -1,
  max.cov.per.pool = 1e+06,
  min.maf = -1,
  verbose = TRUE
)
```

Arguments

`genobaypass.file`	The name (or a path) of the BayPass read count file (see the BayPass manual `https://forge.inrae.fr/mathieu.gautier/baypass_public/`)
`poolsize.file`	The name (or a path) of the BayPass (haploid) pool size file (see the BayPass manual `https://forge.inrae.fr/mathieu.gautier/baypass_public/`)
`snp.pos`	An optional two column matrix with nsnps rows containing the chromosome (or contig/scaffold) of origin and the position of each markers
`poolnames`	A character vector with the names of pool
`min.cov.per.pool`	Minimal allowed read count (per pool). If at least one pool is not covered by at least min.cov.perpool reads, the position is discarded
`max.cov.per.pool`	Maximal allowed read count (per pool). If at least one pool is covered by more than min.cov.perpool reads, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio overall read counts for the reference allele over the read coverage)
`verbose`	If TRUE extra information is printed on the terminal

Details

Information on SNP position is only required for some graphical display or to carried out block-jacknife sampling estimation of confidence intervals. If no mapping information is given (default), SNPs will be assumed to be ordered on the same chromosome and separated by 1 bp. As blocks are defined with a number of consecutive SNPs (rather than a length), the latter assumption has actually no effect (except in the reported estimated block sizes in Mb).

Value

A pooldata object containing 7 elements:

1. "refallele.readcount": a matrix with nsnp rows and npools columns containing read counts for the reference allele (chosen arbitrarily) in each pool
2. "readcoverage": a matrix with nsnp rows and npools columns containing read coverage in each pool
3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele taken as reference in the refallele.readcount matrix (3rd column); and the alternative allele (4th column)
4. "poolsizes": a vector of length npools containing the haploid pool sizes
5. "poolnames": a vector of length npools containing the names of the pools
6. "nsnp": a scalar corresponding to the number of SNPs
7. "npools": a scalar corresponding to the number of pools

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
pooldata2genobaypass(pooldata=pooldata,writing.dir=tempdir())
pooldata=genobaypass2pooldata(genobaypass.file=paste0(tempdir(),"/genobaypass"),
                              poolsize.file=paste0(tempdir(),"/poolsize"))
```

---

genoselestim2pooldata *Convert SelEstim read count input files into a pooldata object*

Description

Convert SelEstim read count input files into a pooldata object

Usage

```
genoselestim2pooldata(
  genoselestim.file = "",
  poolnames = NA,
  min.cov.per.pool = -1,
  max.cov.per.pool = 1e+06,
  min.maf = -1,
  nlines.per.readblock = 1e+06,
  verbose = TRUE
)
```

Arguments

`genoselestim.file`	The name (or a path) of the SelEstim read count file (see the SelEstim manual https://www1.montpellier.inrae.fr/CBGP/software/selestim/)
`poolnames`	A character vector with the names of pool
`min.cov.per.pool`	Minimal allowed read count (per pool). If at least one pool is not covered by at least min.cov.perpool reads, the position is discarded
`max.cov.per.pool`	Maximal allowed read count (per pool). If at least one pool is covered by more than min.cov.perpool reads, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio overal read counts for the reference allele over the read coverage)
`nlines.per.readblock`	Number of Lines read simultaneously. Should be adapted to the available RAM.
`verbose`	If TRUE extra information is printed on the terminal

Value

A pooldata object containing 7 elements:

1. "refallele.readcount": a matrix with nsnp rows and npools columns containing read counts for the reference allele (chosen arbitrarily) in each pool
2. "readcoverage": a matrix with nsnp rows and npools columns containing read coverage in each pool
3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele taken as reference in the refallele.readcount matrix (3rd column); and the alternative allele (4th column)
4. "poolsizes": a vector of length npools containing the haploid pool sizes
5. "poolnames": a vector of length npools containing the names of the pools
6. "nsnp": a scalar corresponding to the number of SNPs
7. "npools": a scalar corresponding to the number of pools

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
pooldata2genoselestim(pooldata=pooldata,writing.dir=tempdir())
pooldata=genoselestim2pooldata(genoselestim.file=paste0(tempdir(),"/genoselestim"))
```

`genotreemix2countdata` *Convert allele count input files from the Treemix program into a coundata object*

Description

Convert allele count input files from the Treemix program into a coundata object

Usage

```
genotreemix2countdata(
  genotreemix.file = "",
  snp.pos = NA,
  min.indgeno.per.pop = -1,
  min.maf = -1,
  verbose = TRUE
)
```

Arguments

`genotreemix.file`	The name (or a path) of the Treemix allele count file (see the Treemix manual `https://bitbucket.org/nygcresearch/treemix/wiki/Home`)
`snp.pos`	An optional two column matrix with nsnps rows containing the chromosome (or contig/scaffold) of origin and the position of each markers
`min.indgeno.per.pop`	Minimal number of overall counts required in each population. If at least one pop is not genotyped for at least min.indgeno.per.pop (haploid) individual, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio overall counts for the reference allele over the overall number of (haploid) individual genotyped)
`verbose`	If TRUE extra information is printed on the terminal

Details

Information on SNP position is only required for some graphical display or to carried out block-jacknife sampling estimation of confidence intervals. If no mapping information is given (default), SNPs will be assumed to be ordered on the same chromosome and separated by 1 bp. As blocks are defined with a number of consecutive SNPs (rather than a length), the latter assumption has actually no effect (except in the reported estimated block sizes in Mb).

Value

A countdata object containing 6 elements:

1. "refallele.count": a matrix (nsnp rows and npops columns) with the allele counts for the reference allele
2. "total.count": a matrix (nsnp rows and npops columns) with the total number of counts (i.e., twice the number of genotyped individual for diploid species and autosomal markers)
3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele taken as reference in the refallele.count matrix (3rd column); and the alternative allele (4th column)
4. "popnames": a vector of length npops containing the names of the pops
5. "nsnp": a scalar corresponding to the number of SNPs
6. "npops": a scalar corresponding to the number of populations

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
##NOTE: This example is just for the sake of illustration as it amounts
##to interpret read count as allele count which must not be done in practice!
dum=matrix(paste(pooldata@refallele.readcount,
  pooldata@readcoverage-pooldata@refallele.readcount,sep=","),
  ncol=pooldata@npools)
```

```
colnames(dum)=pooldata@poolnames
write.table(dum,file=paste0(tempdir(),"/genotreemix"),quote=FALSE,row.names=FALSE)
countdata=genotreemix2countdata(genotreemix.file=paste0(tempdir(),"/genotreemix"))
```

graph.builder — *Implement a graph builder heuristic by successively adding leaves to an initial graph*

Description

Implement a graph builder heuristic by successively adding leaves to an initial graph

Usage

```
graph.builder(
  x,
  leaves.to.add,
  fstats,
  heap.dbic = 6,
  max.heap.size = 25,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	An object (or list of objects) of class graph.params or fitted.graph (see details)
`leaves.to.add`	Names of the leaves to successively add (in the given order)
`fstats`	Object of class fstats that contains estimates of the fstats (see compute.fstats)
`heap.dbic`	Maximal BIC distance from the best graph to be kept in the heap (heap.dbic=6 by default)
`max.heap.size`	Maximal number of graphs stored in the heap (max.heap.size=25 by default)
`verbose`	If TRUE extra information is printed on the terminal
`...`	Some parameters to be passed the function add.leaf called internally

Details

The input object x needs to be of class graph.params as generated by the function generate.graph.params; or fitted.graph as generated by the functions fit.graph, add.leaf (in the output list element named "fitted.graphs.list") or rooted.nj.builder (in the output element named "best.rooted.tree"). This is to ensure that the matrix describing the structure of the graph (graph slot of these objects) is valid (note that it can be plotted for checks). Hence graph.params objects may have been generated without fstats information (that should be supplied independently to the add.leaf function to obtain information on the fstats involving the candidate leaf defined with the leaf.to.add argument). The functions successively add each leaf given in the leaves.to.add vector to the list of fitted graph stored in a heap using the function add.leaf. For the first iteration (i.e., first tested leaf) the heap consists of the input graph or list of graph x. At each iteration, the function add.leaf is used to test the candidate leaf to each graph from the current heap in turn. A new heap of graphs is then built by each time including the fitted graphs with a BIC less than heap.dbic larger than the best resulting graphs (treating each graph independently). If the final number of graphs in the heap is larger than max.heap.size, the max.heap.size graphs with the lowest BIC are kept in the heap. After testing the latest leaf, graphs with a BIC larger than heap.dbic units of the best graph are discarded from the final list of graphs. In practice, it is recommended to test different orders of inclusion of the leaves (as specified in the vector leaves.to.add)

Value

A list with the following elements:

1. "n.graphs": The final number of fitted graphs
2. "fitted.graphs.list": a list of fitted.graph objects (indexed from 1 to n.graphs and in the same order as the list "graphs") containing the results of fitting of each graph.
3. "best.fitted.graph": The graph (object of class fitted.graph) with the minimal BIC (see function fit.graph) among all the graphs within fitted.graphs.list
4. "bic": a vector of the n.graphs BIC (indexed from 1 to n.graphs and in the same order as the "fitted.graphs.list" list) (see fit.graph details for the computation of the scores).

See Also

see `fit.graph`, `generate.graph.params` and `add.leaf`.

`graph.params-class` *S4 class to represent a population tree or admixture graph and its underlying parameter.*

Description

S4 class to represent a population tree or admixture graph and its underlying parameter.

Details

The graph is specified by a three column (character) matrix giving for each edge (whether admixed or not) to i) the child node; ii) the parent node; iii) the admixture proportion. For non-admixed edge, the third column must be blank. An admixed node should be referred two times as a child node with two different parent node and two different admixture proportions coded as alpha and (1-alpha) (parentheses are mandatory) if alpha is the name of the parameter for admixture proportion. The dot.graph element allows to plot the graph using grViz() from the DiagrammeR package or with the dot program after writing the files (e.g., dot -Tpng inputgraph.dot in terminal). Note that the dot file may be customized (e.g., to change leave color, parameter names...).

Slots

`graph` The graph in 3 column format (see details)

`dot.graph` The graph in dot format

`is.admgraph` If FALSE the graph is binary tree (i.e., no admixture events), if TRUE the graph is an admixture graph

`n.leaves` Number of leaves of the graph

`leaves` Name of the leaves

`root.name` Name of the root

`n.nodes` Number of nodes (including root)

`nodes.names` Name of the nodes

`n.edges` Number of edges (including admixture edges)

`edges.names` Names of the edges (coded as "Parent node Name"<->"Child node Name")

`n.adm.nodes` Number of admixed nodes (=0 if is.admgraph=FALSE). This is also the number of admixed parameters since only two-ways admixture are assumed for a given node

`adm.params.names` Names of the admixed parameters

`graph.matrix` The graph incidence matrix consisting of n.leaves rows and n.edges columns. The elements of the matrix are the weights of each edge (in symbolic representation) for the different possible paths from the leaves to the graph root.

`root.edges.idx` Indexes of the graph.matrix columns associated to the (two) edges connected to the root

`f2.target` The (n.leaves-1) stats F2 involving popref (i.e., of the form F2(popref;pop))

`f2.target.pops` A matrix of (n.leaves-1) rows and 2 columns containing the names of populations of the F2 stats. The first column is by construction always popref. The order is the same as in f2.target

`f3.target` The (n.leaves-1)(n.leaves-2)/2 stats F3 involving popref as a target (i.e., of the form F3(popref;popA,popB))

`f3.target.pops` A matrix of (n.leaves-1)(n.leaves-2)/2 rows and 3 columns containing the name of popref in the first column and the names of the two populations involved in the F3 stats. The order is the same as in f3.target

`popref` The name of the reference population defining the fstats basis

`f.Qmat` A square matrix of rank n.leaves(n.leaves-1)/2 corresponding to the error covariance matrix of the F2 and F3 estimates

`Het` Estimated leave heterozygosities (if present in the fstats object)

See Also

To generate graph.params object, see `generate.graph.params`. The object may be used to estimate graph parameters with the function `fit.graph` or to generate files for the qpGraph software with `graph.params2qpGraphFiles`. See also `graph.params2symbolic.fstats` to obtain symbolic representation of Fstats from the matrix "Omega".

graph.params2qpGraphFiles *Generate files for the qpGraph software from a graph.params object*

Description

Generate files for the qpGraph software from a graph.params object

Usage

```
graph.params2qpGraphFiles(
  graph.params,
  outfileprefix = "out",
  n.printed.dec = 4,
  verbose = TRUE
)
```

Arguments

`graph.params`	An object of class graph.params containing graph information with Fstats information (see the function generate.graph.params)
`outfileprefix`	The prefix of the qpGraph files
`n.printed.dec`	Number of decimal to be printed (if not enough may lead to fatalx error in qpGraph)
`verbose`	If TRUE extra information is printed on the terminal

Details

This function generates the three files required by qpGraph: i) a file named [outfileprefix].graph containing the graph in appropriate format; ii) a file named [outfileprefix].fstats file containing the fstats estimates of fstats (and their covariance); iii) a file named [outfileprefix].parqpGraph containing essential parameter information to run qpGraph (this may be edited by hand if other options are needed). The qpGraph software may then be run using the following options -p [outfileprefix].parqpGraph -g [outfileprefix].graph -o out.ggg -d out.dot.

Value

The three files described in the details section

See Also

To generate graph.params object, see `generate.graph.params`

graph.params2symbolic.fstats

Provide a symbolic representation of all the F-statistics and the model system of equations

Description

Provide a symbolic representation of all the F-statistics and the model system of equations

Usage

```
graph.params2symbolic.fstats(x, outfile = NULL)
```

Arguments

`x`	An object of class graph.params containing graph information and relevant Fstats estimates (see the function generate.graph.params)
`outfile`	The file where to print the equations (default=NULL, equations are not printed in a file)

Value

A list with the following elements:

1. "model.matrix": A symbolic representation of the matrix M relating the basis F-statistics and graph edge length as F=M*b where F is the vector of the basis Fstats (row names of model.matrix M) and b is the vector of graph edges (column names of model.matrix M).
2. "omega": A symbolic representation of the scaled covariance matrix of allele frequency with edge names and admixture parameter names as specified in the edges.names and adm.params.names slot of the input graph.params object x
3. "F2.equations": A symbolic representation of the nleaves(nleaves-1)/2 different F2 as a function of graph parameters
4. "F3.equations": A symbolic representation of the nleaves(nleaves-1)(nleaves-2)/2 different F3 as a function of graph parameters
5. "F4.equations": A symbolic representation of the npops(npops-1)(npops-2)(npops-3)/8 different F4 as a function of graph parameters

See Also

To generate a graph.params object, see `generate.graph.params`.

Examples

```
graph=rbind(c("P1","P7",""),c("P2","s1",""),c("P3","s2",""),c("P6","S",""),
           c("S","s1","a"),c("S","s2","(1-a)"),c("s2","P8",""),c("s1","P7",""),
           c("P4","P9",""),c("P5","P9",""),c("P7","P8",""),
           c("P8","R",""),c("P9","R",""))
graph.params=generate.graph.params(graph)
graph.equations=graph.params2symbolic.fstats(graph.params)
```

---

heatmap,pairwisefst-method

Show pairwisefst object

Description

Show pairwisefst object

Usage

```
## S4 method for signature 'pairwisefst'
heatmap(
  x,
  Rowv = NULL,
  Colv = if (symm) "Rowv" else NULL,
  distfun = as.dist,
  hclustfun = hclust,
  reorderfun = function(d, w) reorder(d, w),
  add.expr,
  symm = FALSE,
  revC = identical(Colv, "Rowv"),
  scale = c("row", "column", "none"),
  na.rm = TRUE,
  margins = c(5, 5),
  ColSideColors,
  RowSideColors,
  cexRow = 0.2 + 1/log10(nrow(x@PairwiseFSTmatrix)),
  cexCol = 0.2 + 1/log10(ncol(x@PairwiseFSTmatrix)),
  labRow = NULL,
  labCol = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  keep.dendro = FALSE,
  verbose = getOption("verbose"),
  ...
)
```

Arguments

`x`	Object of class pairwisefst
`Rowv`	determines if and how the row dendrogram should be computed and reordered. Either a dendrogram or a vector of values used to reorder the row dendrogram or NA to suppress any row dendrogram (and reordering) or by default, NULL, see 'Details' below.
`Colv`	determines if and how the column dendrogram should be reordered. Has the same options as the Rowv argument above and additionally when x is a square matrix, Colv = "Rowv" means that columns should be treated identically to the rows (and so if there is to be no row dendrogram there will not be a column one either).
`distfun`	function used to compute the distance (dissimilarity) between both rows and columns. Defaults to as.dist.
`hclustfun`	function used to compute the hierarchical clustering when Rowv or Colv are not dendrograms. Defaults to hclust. Should take as argument a result of distfun and return an object to which as.dendrogram can be applied.
`reorderfun`	function(d, w) of dendrogram and weights for reordering the row and column dendrograms. The default uses reorder.dendrogram.
`add.expr`	expression that will be evaluated after the call to image. Can be used to add components to the plot.
`symm`	logical indicating if x should be treated symmetrically; can only be true when x is a square matrix.
`revC`	logical indicating if the column order should be reversed for plotting, such that e.g., for the symmetric case, the symmetry axis is as usual.
`scale`	character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. The default is "row" if symm false, and "none" otherwise.
`na.rm`	logical indicating whether NA's should be removed.
`margins`	numeric vector of length 2 containing the margins (see par(mar = *)) for column and row names, respectively.
`ColSideColors`	(optional) character vector of length ncol(x) containing the color names for a horizontal side bar that may be used to annotate the columns of x.
`RowSideColors`	(optional) character vector of length nrow(x) containing the color names for a vertical side bar that may be used to annotate the rows of x.
`cexRow`, `cexCol`	positive numbers, used as cex.axis in for the row or column axis labeling. The defaults currently only use number of rows or columns, respectively.
`labRow`, `labCol`	character vectors with row and column labels to use; these default to rownames(x) or colnames(x), respectively.
`main`, `xlab`, `ylab`	main, x- and y-axis titles; defaults to none.
`keep.dendro`	logical indicating if the dendrogram(s) should be kept as part of the result (when Rowv and/or Colv are not NA).
`verbose`	logical indicating if information should be printed.
`...`	additional arguments passed on to image, e.g., col specifying the colors.

is.countdata *Check countdata objects*

Description

Check countdata objects

Usage

```
is.countdata(x)
```

Arguments

x	The name of the object to be tested

is.fitted.graph *Check fitted.graph objects*

Description

Check fitted.graph objects

Usage

```
is.fitted.graph(x)
```

Arguments

x	Object to be tested

is.fstats *Check fstats objects*

Description

Check fstats objects

Usage

```
is.fstats(x)
```

Arguments

x	The name of the object to be tested

`is.graph.params` *Check graph.params objects*

Description

Check graph.params objects

Usage

```
is.graph.params(x)
```

Arguments

`x`	The name (or a path) of the graph.params objet

`is.pairwisefst` *Check pairwisefst objects*

Description

Check pairwisefst objects

Usage

```
is.pairwisefst(x)
```

Arguments

`x`	The name (or a path) of the pairwisefst object

`is.pooldata` *Check pooldata objects*

Description

Check pooldata objects

Usage

```
is.pooldata(x)
```

Arguments

`x`	The name of the object to be tested

`make.example.files` *Create example files*

Description

Write in the current directory example files corresponding to a sync (as obtained when parsing mpileup files with PoPoolation) and vcf (as obtained when parsing mpileup files with VarScan) gzipped files

Usage

```
make.example.files(writing.dir = "")
```

Arguments

`writing.dir`	Directory where to copy example files (e.g., set writing.dir=getwd() to copy in the current working directory)

Examples

```
make.example.files(writing.dir=tempdir())
```

`pairwisefst-class` *S4 class to represent a pairwise Fst results obtained with the compute.pairwiseFST*

Description

S4 class to represent a pairwise Fst results obtained with the compute.pairwiseFST

Slots

`values` A data frame with npop*(npop-1)/2 rows and 3 (or 7 if blockjackknife is TRUE) columns containing for both the Fst and Q2, estimates over all the SNPs and if blockjackknife=TRUE, the estimated block-jackknife and standard error (s.e.). The seventh (or third if blockjackknife=FALSE) column gives the number of SNPs.

`PairwiseFSTmatrix` A npxnp matrix containing the pairwise FST estimates

`PairwiseSnpFST` A matrix (nsnp rows and npops columns) with read count data for the reference allele

`PairwiseSnpQ1` A matrix (nsnp rows and npops columns) with overall read coverage

`PairwiseSnpQ2` A matrix (nsnp rows and 4 columns) detailing for each SNP, the chromosome (or scaffold), the position, allele 1 and allele 2

`blockjacknife` A logical indicating whether block-jackknife estimates of standard errors are available (TRUE) or not (FALSE)

See Also

To generate pairwise object, see `compute.pairwiseFST`

`plot,fitted.graph-method`
plot pairwisefst object

Description

plot pairwisefst object

Usage

```
## S4 method for signature 'fitted.graph'
plot(x, y)
```

Arguments

`x`	Object of class fitted.graph
`y`	dummy argument

`plot,fstats-method` *plot fstats object*

Description

plot fstats object

Usage

```
## S4 method for signature 'fstats'
plot(x, y, ...)
```

Arguments

`x`	Object of class fstats
`y`	dummy argument
`...`	Other arguments to be passed to plot_fstats

See Also

see `plot_fstats` for details on plot_fstats arguments

plot,graph.params-method

plot graph in graph.params object

Description

plot graph in graph.params object

Usage

```
## S4 method for signature 'graph.params'
plot(x, y)
```

Arguments

x	Object of class fitted.graph
y	dummy argument

plot,pairwisefst-method

plot pairwisefst object

Description

plot pairwisefst object

Usage

```
## S4 method for signature 'pairwisefst'
plot(x, y, ...)
```

Arguments

x	Object of class pairwisefst
y	dummy argument
...	Some arguments to be passed to plot_fstats

See Also

see `plot_fstats` for details on plot_fstats arguments

plot_fstats *Plot F2, F3, F3star, F4, D or pairwise Fst values with their Confidence Intervals*

Description

Plot F2, F3, F3star, F4, D or pairwise Fst values with their Confidence Intervals

Usage

```
plot_fstats(
  x,
  stat.name = "F2",
  ci.perc = 95,
  value.range = c(NA, NA),
  pop.sel = NA,
  pop.f3.target = NA,
  highlight.signif = TRUE,
  main = stat.name,
  ...
)
```

Arguments

`x`	An object of class fstats (to plot heterozygosities, divergence, F2, F3, F3star, F4 or D statistics) or pairwisefst (to plot pairwise fst)
`stat.name`	For fstats object, the name of the stat (either heterozygosities, divergence, F2, F3, F3star, F4 or Dstat)
`ci.perc`	Percentage of the Confidence Interval in number of standard errors (default=95%)
`value.range`	Range of test values (x-axis) to be plotted (default=NA,NA: i.e., all test values are plotted)
`pop.sel`	Only plot test values involving these populations (default=NA: i.e., all test values are plotted)
`pop.f3.target`	For F3-statistics, only plot F3 involving pop.f3.target as a target
`highlight.signif`	If TRUE highlight significant tests in red (see details)
`main`	Main title of the plot (default=stat.name)
`...`	Some other graphical arguments to be passed

Details

Data will only be plotted if jackknife estimates of the estimator s.e. have been performed i.e. if the functions compute.fstats or compute.pairwiseFST were run with nsnp.per.block>0

Value

A plot of the Fstats of interest. Significant F3 statistics (i.e., showing formal evidence for admixture of the target population) are highlighted in red. Significant F4 statistics (i.e., showing formal evidence against treeness of the pop. quadruplet) are highlighted in red.

See Also

To generate x object, see `compute.pairwiseFST` (for pairwisefst object) or `compute.fstats` (for fstats object)

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),
                  poolsizes=rep(50,15),poolnames=paste0("P",1:15))
res.fstats=compute.fstats(pooldata,nsnp.per.bjack.block=25)
plot_fstats(res.fstats,stat.name="F3",cex=0.5)
plot_fstats(res.fstats,stat.name="F3",value.range=c(NA,0.001),
            pop.f3.target=c("P7","P5"),cex.axis=0.7)
plot_fstats(res.fstats,stat.name="F4",cex=0.5)
#allow to reduce the size of the test name (y-axis)
plot_fstats(res.fstats,stat.name="F4",cex=0.5,
            pop.sel=c("P1","P2","P3","P4","P5"))
plot_fstats(res.fstats,stat.name="F4",cex=0.5,
            pop.sel=c("P1","P2","P3","P4","P5"),highlight.signif=FALSE)
```

`pooldata-class` *S4 class to represent a Pool-Seq data set.*

Description

S4 class to represent a Pool-Seq data set.

Slots

`npools` The number of pools

`nsnp` The number of SNPs

`refallele.readcount` A matrix (nsnp rows and npools columns) with read count data for the reference allele

`readcoverage` A matrix (nsnp rows and npools columns) with overall read coverage

`snp.info` A data frame (nsnp rows and 4 columns) detailing for each SNP, the chromosome (or scaffold), the position, Reference allele name and Alternate allele name (if available)

`poolsizes` A vector of length npools with the corresponding haploid pool sizes

`poolnames` A vector of length npools with the corresponding haploid pool names

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`, `genobaypass2pooldata` and `genoselestim2pooldata`

`pooldata.subset` *Create a subset of the pooldata object that contains Pool-Seq data as a function of pool and/or SNP indexes*

Description

Create a subset of the pooldata object that contains Pool-Seq data as a function of pool and/or SNP indexes

Usage

```
pooldata.subset(
  pooldata,
  pool.index = 1:pooldata@npools,
  snp.index = 1:pooldata@nsnp,
  min.maf = -1,
  min.cov = 0,
  max.cov = 1e+09,
  cov.qthres = c(0, 1),
  min.cov.per.pool = -1,
  max.cov.per.pool = 1e+06,
  cov.qthres.per.pool = c(0, 1),
  return.snp.idx = FALSE,
  verbose = TRUE
)
```

Arguments

`pooldata`	A pooldata object containing Pool-Seq information
`pool.index`	Indexes of the pools (at least two), that should be selected to create the new pooldata object (default=all the pools)
`snp.index`	Indexes of the SNPs (at least two), that should be selected to create the new pooldata object (default=all the SNPs)
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio over all read counts for the reference allele over the read coverage)
`min.cov`	Minimal allowed read count (over all the pools).
`max.cov`	Maximal allowed read count (over all the pools).
`cov.qthres`	A two-elements vector containing the minimal (qmin) and maximal (qmax) quantile thresholds (0<=qmin<qmax<=1) for the overall coverage (i.e., summing over all pools). See details below

`min.cov.per.pool`	Minimal allowed read count (per pool). If at least one pool is not covered by at least min.cov.perpool reads, the position is discarded
`max.cov.per.pool`	Maximal allowed read count (per pool). If at least one pool is covered by more than min.cov.perpool reads, the position is discarded
`cov.qthres.per.pool`	A two-elements vector containing the minimal (qmin) and maximal (qmax) quantile coverage thresholds applied to each pools (0<=qmin<qmax<=1). See details below
`return.snp.idx`	If TRUE, the row.names of the snp.info slot of the returned pooldata object are named as "rsx" where x is the index of SNP in the initial pooldata object (default=FALSE)
`verbose`	If TRUE return some information

Details

This function subsets a `pooldata` object by selecting a subset of pools and/or SNPs (e.g., based on genomic position).

Additional SNP-level filtering can be applied to the resulting subset to remove poorly polymorphic SNPs using `min.maf`, or SNPs with low or excessively high coverage using `min.cov`, `max.cov`, and `cov.qthres`.

Coverage filtering can also be performed on a per-pool basis with `min.cov.per.pool`, `max.cov.per.pool`, and `cov.qthres.per.pool`.

For the `cov.qthres` and `cov.qthres.per.pool` arguments, empirical coverage quantiles are computed and used as filtering thresholds. SNPs with coverage outside the specified quantile range are discarded.

For example, if `qmax = 0.95` in `cov.qthres.per.pool`, a SNP is removed in a given pool if its coverage exceeds the 95th percentile of the empirical coverage distribution for that pool (computed over SNPs selected by `snp.index`). Conversely, if `qmin = 0.05`, SNPs with coverage below the 5th percentile are discarded.

Quantile-based filtering is particularly useful when pools display heterogeneous sequencing depth.

Value

A pooldata object with 7 elements:

1. "refallele.readcount": a matrix with nsnp rows and npools columns containing read counts for the reference allele (chosen arbitrarily) in each pool
2. "readcoverage": a matrix with nsnp rows and npools columns containing read coverage in each pool
3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele in the reference assembly (3rd column); the allele taken as reference in the refallele matrix.readcount matrix (4th column); and the alternative allele (5th column)
4. "poolsizes": a vector of length npools containing the haploid pool sizes

5. "poolnames": a vector of length npools containing the names of the pools
6. "nsnp": a scalar corresponding to the number of SNPs
7. "npools": a scalar corresponding to the number of pools

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 subset.by.pools=pooldata.subset(pooldata,pool.index=c(1,2))
 subset.by.snps=pooldata.subset(pooldata,snp.index=10:100)
 subset.by.pools.and.snps=pooldata.subset(pooldata,pool.index=c(1,2),snp.index=10:100)
subset.by.pools.qcov.thr=pooldata.subset(pooldata,pool.index=1:8,cov.qthres.per.pool=c(0.05,0.95))
```

`pooldata2diyabc` *Convert a pooldata object into DIYABC input files.*

Description

Convert a pooldata object into DIYABC data file for pool-seq data. A file containing SNP details is also printed out. Options to generate sub-samples (e.g., for large number of SNPs) are also available. Note that DIYABC SNP filtering criterion is based on MRC (minimal read count) which may be more stringent than usual MAF-based filtering criterion. It is recommended to parse vcf files and pooldata objects without any MAF criterion or to prefilter pooldata objects with the desired MRC (using option snp.index `pooldata.subset`).

Usage

```
pooldata2diyabc(
  pooldata,
  writing.dir = getwd(),
  prefix = "",
  diyabc.mrc = 1,
  subsamplesize = -1,
  subsamplingmethod = "thinning"
)
```

Arguments

`pooldata`	A pooldata object containing Pool-Seq information (see `vcf2pooldata` and `popsync2pooldata`)
`writing.dir`	Directory where to create the files (e.g., set writing.dir=getwd() to copy in the current working directory)
`prefix`	Prefix used for output file names

`diyabc.mrc`	MRC to be applied by DIYABC (note that no filtering based on MRC is done by the function)
`subsamplesize`	Size of the sub-samples. If <=1 (default), all the SNPs are considered in the output
`subsamplingmethod`	If sub-sampling is activated (argument subsamplesize), define the method used for subsampling that might be either i) "random" (A single data set consisting of randmly chosen SNPs is generated) or ii) "thinning", sub-samples are generated by taking SNPs one every nsub=floor(nsnp/subsamplesize) in the order of the map (a suffix ".subn" is added to each sub-sample files where n varies from 1 to nsub).

Value

DIYABC data file for pool-seq data

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 pooldata2diyabc(pooldata=pooldata,writing.dir=tempdir())
```

`pooldata2genobaypass` *Convert a pooldata object into BayPass input files.*

Description

Convert a pooldata object into BayPass allele read count and haploid pool size files. A file containing SNP details is also printed out. Options to generate sub-samples (e.g., for large number of SNPs) are also available.

Usage

```
pooldata2genobaypass(
  pooldata,
  writing.dir = getwd(),
  prefix = "",
  subsamplesize = -1,
  subsamplingmethod = "thinning"
)
```

Arguments

`pooldata`	A pooldata object containing Pool-Seq information (see `vcf2pooldata` and `popsync2pooldata`)
`writing.dir`	Directory where to create the files (e.g., set writing.dir=getwd() to copy in the current working directory)
`prefix`	Prefix used for output file names
`subsamplesize`	Size of the sub-samples. If <=1 (default), all the SNPs are considered in the output
`subsamplingmethod`	If sub-sampling is activated (argument subsamplesize), define the method used for subsampling that might be either i) "random" (A single data set consisting of randmly chosen SNPs is generated) or ii) "thinning", sub-samples are generated by taking SNPs one every nsub=floor(nsnp/subsamplesize) in the order of the map (a suffix ".subn" is added to each sub-sample files where n varies from 1 to nsub).

Value

Files containing allele count (in BayPass format), haploid pool size (in BayPass format), and SNP details (as in the snp.info matrix from the pooldata object)

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 pooldata2genobaypass(pooldata=pooldata,writing.dir=tempdir())
```

`pooldata2genoselestim` *Convert a pooldata object into SelEstim input files.*

Description

Convert a pooldata object into SelEstim allele read count. A file containing SNP details is also printed out. Options to generate sub-samples (e.g., for large number of SNPs) are also available.

Usage

```
pooldata2genoselestim(
  pooldata,
  writing.dir = getwd(),
  prefix = "",
  subsamplesize = -1,
  subsamplingmethod = "thinning"
)
```

Arguments

`pooldata`	A pooldata object containing Pool-Seq information (see `vcf2pooldata` and `popsync2pooldata`)
`writing.dir`	Directory where to create the files (e.g., set writing.dir=getwd() to copy in the current working directory)
`prefix`	Prefix used for output file names
`subsamplesize`	Size of the sub-samples. If <=1 (default), all the SNPs are considered in the output
`subsamplingmethod`	If sub-sampling is activated (argument subsamplesize), define the method used for subsampling that might be either i) "random" (A single data set consisting of randmly chosen SNPs is generated) or ii) "thinning", sub-samples are generated by taking SNPs one every nsub=floor(nsnp/subsamplesize) in the order of the map (a suffix ".subn" is added to each sub-sample files where n varies from 1 to nsub).

Value

Files containing allele count (in SelEstim Pool-Seq format) and SNP details (as in the snp.info matrix from the pooldata object)

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
pooldata2genoselestim(pooldata=pooldata,writing.dir=tempdir())
```

---

poolify — *Create a (pseudo-)pooled sample from a set of samples (using random-allele sampling)*

Description

Create a (pseudo-)pooled sample from a set of samples (using random-allele sampling)

Usage

```
poolify(x, sample.index = NULL, out.samplename = "PoolSample")
```

Arguments

`x`	Pooldata or countdata object containing samples to pool
`sample.index`	Indexes of the pools or pops (at least two), that should be selected to create the pool sample (default: all)
`out.samplename`	Name of the poolified sample (default: PoolSample)

Details

This function generates a ‘countdata‘ with a single sample combining read or allele counts from different samples stored in either a ‘pooldata‘ or a ‘countdata‘ object. To avoid introducing bias in downstream analyses, different strategies are applied depending on the type of input and the desired output:

1. The input is a ‘countdata‘ object (allele counts): The output consists of the sum of allele counts across the selected samples (specified via ‘sample.index‘). The resulting "poolified" ‘countdata‘ object can be used to simulate Pool-Seq data using the function `sim.readcounts` or directly merged with some other samples using `data.merge`.

2. The input is a ‘pooldata‘ object (read counts): A random allele approach is used. For each SNP, one read is randomly sampled per sample to be pooled. This downsampling strategy—although it reduces information—ensures that all reads in the pooled sample originate from different chromosomes (i.e., no reads come from the same individual chromosome). The resulting ‘countdata‘ object can then be merged with other samples stored in either ‘countdata‘ or ‘pooldata‘ objects using the function `data.merge`.

Value

A ‘countdata‘ object containing the "poolified" sample.

See Also

To merge the object with other ‘countdata‘ and ‘pooldata‘ objects, see `data.merge`. To simulate PoolSeq data from allele count data, see `sim.readcounts`

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
#random allele pooling of sample P2 to P10
P2toP10pseudopool=poolify(pooldata,sample.index=2:10,out.samplename="P2toP10")
#merge other sample
newpooldata=data.merge(pooldata.subset(pooldata,pool.index=c(1,11:15)),P2toP10pseudopool)
newpooldata
#Working with allele count data
#create a countdata object (NOTE: This example is just for the sake of illustration)
pooldata2genobaypass(pooldata,writing.dir=tempdir())
countdata=genobaypass2countdata(genobaypass.file=paste0(tempdir(),"/genobaypass"))
countdata@snp.info=pooldata@snp.info
#merge counts (no random allele sampling) from sample P2 to P10
P2toP10mergecounts=poolify(countdata,sample.index=2:10,out.samplename="P2toP10")
#merge other sample from the original countdata
newcountdata=data.merge(countdata.subset(countdata,pop.index=c(1,11:15)),P2toP10mergecounts)
newcountdata
#simulate a pool-seq sample from the pooled counts of P2 to P10 sample
#and merge it with the original Pool-Seq
poolP2toP10=sim.readcounts(P2toP10mergecounts,min.rc=0,seq.eps=0)
#Merge with other PoolSeq sample (First ensure that the same SNPs with the same alleles are merged)
snp.sel.idx=which(paste0(pooldata@snp.info$Chromosome,pooldata@snp.info$Position)%in%
                  paste0(poolP2toP10@snp.info$Chromosome,poolP2toP10@snp.info$Position))
```

```
newpooldata=data.merge(pooldata.subset(pooldata,pool.index=c(1,11:15),snp.index=snp.sel.idx),
                       poolP2toP10)
newpooldata
```

poppair_idx *poppair_idx*

Description

Compute the index of the pairwise comparison from the idx of each pop

Arguments

idx_pop1	Integer giving the (0-indexed) index of the first pop
idx_pop2	Integer giving the (0-indexed) index of the second pop
nidx	Integer giving the total number of indexes (i.e., number of pops)

Details

If idx_pop2 < idx_pop1, indexes are reversed

Value

Return the (0-indexed) index for the row associated to the pairwise comparison in the ordered flat list of all (npop*(npop-1))/2 pairwise stats

Examples

```
#
```

popsync2pooldata *Convert Popoolation Sync files into a pooldata object*

Description

Convert Popoolation Sync files into a pooldata object

Usage

```
popsync2pooldata(
  sync.file = "",
  poolsizes = NA,
  poolnames = NA,
  min.rc = 1,
  min.cov.per.pool = -1,
  max.cov.per.pool = 1e+06,
  min.maf = 0.01,
  noindel = TRUE,
  nlines.per.readblock = 1e+06,
  nthreads = 1
)
```

Arguments

`sync.file`	The name (or a path) of the Popoolation sync file (might be in compressed format)
`poolsizes`	A numeric vector with haploid pool sizes
`poolnames`	A character vector with the names of pool
`min.rc`	Minimal allowed read count per base. Bases covered by less than min.rc reads are discarded and considered as sequencing error. For instance, if nucleotides A, C, G and T are covered by respectively 100, 15, 0 and 1 over all the pools, setting min.rc to 0 will lead to discard the position (the polymorphism being considered as tri-allelic), while setting min.rc to 1 (or 2, 3..14) will make the position be considered as a SNP with two alleles A and C (the only read for allele T being disregarded).
`min.cov.per.pool`	Minimal allowed read count (per pool). If at least one pool is not covered by at least min.cov.perpool reads, the position is discarded
`max.cov.per.pool`	Maximal allowed read count (per pool). If at least one pool is covered by more than min.cov.perpool reads, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio overal read counts for the reference allele over the read coverage)
`noindel`	If TRUE, positions with at least one indel count are discarded
`nlines.per.readblock`	Number of Lines read simultaneously. Should be adapted to the available RAM.
`nthreads`	Number of available threads for parallelization of some part of the parsing (default=1, i.e., no parallelization)

Value

A pooldata object containing 7 elements:

1. "refallele.readcount": a matrix with nsnp rows and npools columns containing read counts for the reference allele (chosen arbitrarily) in each pool

2. "readcoverage": a matrix with nsnp rows and npools columns containing read coverage in each pool
3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele taken as reference in the refallele.readcount matrix (3rd column); and the alternative allele (4th column)
4. "poolsizes": a vector of length npools containing the haploid pool sizes
5. "poolnames": a vector of length npools containing the names of the pools
6. "nsnp": a scalar corresponding to the number of SNPs
7. "npools": a scalar corresponding to the number of pools

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
```

---

randomallele.pca *PCA of a pooldata or countdata object using a random allele approach*

Description

PCA of a pooldata or countdata object using a random allele approach

Usage

```
randomallele.pca(
  x,
  scale = TRUE,
  return.snploadings = FALSE,
  plot.pcs = c(1, 2),
  ...
)
```

Arguments

x	A pooldata object containing Pool-Seq information or a countdata object containing allele count information
scale	If FALSE the random allele data matrix is not scaled (default=TRUE)
return.snploadings	If TRUE return the SNP loadings (may be large)
plot.pcs	A vector with two-elements giving the two PCs to plot. If NULL, no plotting is done.
...	graphical parameters (see plot function)

Details

PCA is performed by singular-value decomposition (SVD) of a npop (or npools) x nsnp matrix of a single randomly sampled allele (i.e. or read for pooldata object) for each SNP and for each population (inspired by Skoglund and Jakobsson, 2011, https://doi.org/10.1073/pnas.1108181108). Although this approach leads to information loss, it allows to efficiently account for unequal sample size (and read coverages for pool-seq data) and have little impact on the resulting representation when the number of SNPs is large. Note also that the implemented approach is similar to that implemented in the PCA_MDS module of the software ANGSD by Korneliussen et al. (2014) (see http://www.popgen.dk/angsd/index.php/PCA_MDS).

Value

An object of class fstats (see help(fstats) for details)

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`,`genobaypass2pooldata` or `genoselestim2pooldata`. To generate coundata object, see `genobaypass2countdata` or `genotreemix2countdata`.

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata<-popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
 res.pca<-randomallele.pca(pooldata)
```

`rooted.njtree.builder` *Construct and root an Neighbor-Joining tree of presumably nonadmixed leaves*

Description

Construct and root an Neighbor-Joining tree of presumably nonadmixed leaves

Usage

```
rooted.njtree.builder(
  fstats,
  pop.sel,
  edge.fact = 1000,
  plot.nj = FALSE,
  verbose = TRUE
)
```

Arguments

`fstats`	Object of class fstats that contains estimates of the fstats (see compute.fstats)
`pop.sel`	Names of the leaves (pops) used to build the nj tree (at least 3 required)
`edge.fact`	The multiplying factor of edges length in graph representation
`plot.nj`	If TRUE plot the Neighbor-Joining tree
`verbose`	If TRUE extra information is printed on the terminal

Details

A Neighbor-Joining tree is first built (using nj function from the package ape) based on the F2-distance matrix of the leaves in pop.sel which are presumably non-admixed (see the function find.tree.popset to find such groups of scaffold populations using estimated F3 and F4 test statistics). For non-admixed leaves, F2 are indeed expected to be additive along the resulting binary tree (see Lipson et al., 2013). The resulting tree is then rooted using the method described in Lipson et al. (2013) which is based on the property that the estimated heterozygosity of the root h_R equals h_R=1-Q2(A,B) if A and B are two populations sharing R as the only common ancestor in the tree. This estimator should then be consistent across all the possible pairs of populations A and B that are only connected through R in the tree (i.e., that each belong to one of the two partitions of the tree defined by a root position R). Note that 1-Q2(A,B)=(1-Q1(A))/2 + (1-Q1(B))/2 + F2(A,B)=(h_A+h_B)/2+F2(A,B) where h_A, h_B and F2(A,B) are estimated with the function compute.fstats.

Value

A list with the following elements:

1. "n.rooted.trees": The number of possible rooted binary trees that were evaluated
2. "fitted.rooted.trees.list": a list of objects of class fitted.graph containing information on all the possible graphs (indexed from 1 to n.rooted.trees). Each tree may be visualized or further used using functions applied to objects of class fitted.graph (e.g., plot, add.leave)
3. best.rooted.tree The tree (object of class fitted.graph) among all the graphs within fitted.rooted.trees.list displaying the minimal the minimal sd over estimates of h_P (see details)
4. "root.het.est.var": For a matrix of n.tree rows (same order as in the list rooted.tree) and 4 columns with i) the average estimated root heterozygosity h_R across all the pairs of population leave that are relevant for estimation (see details); ii) the size of the range of variation and iii) the s.d. of the estimates of h_R, and iv) the number of population pairs relevant for estimation
5. "nj.tree.eval": If n.edges>3, gives the five worst configuration fit (by calling the compare.fitted.fstats function) which are the same irrespective of rooting

See Also

see `fit.graph`, `generate.graph.params` and `add.leaf`.

`scan_allele_info` *scan_allele_info*

Description

Scan allele information in ALT field of a vcf

Usage

```
.scan_allele_info(allele_info)
```

Arguments

`allele_info`	a character string vector (ALT field of the vcf)

Details

Scan allele information in ALT field of a vcf to identify the number of alleles and if there is indels

Value

Return a vector with two elements consisting i) the number of alleles (1+number of comma) and ii) 0 or 1 if an indel is detected

Examples

```
.scan_allele_info(c("A,C","T","AAT"))
```

`show,countdata-method` *Show countdata object*

Description

Show countdata object

Usage

```
## S4 method for signature 'countdata'
show(object)
```

Arguments

`object`	Object of class countdata

show,fitted.graph-method *Show fitted.graph object*

Description

Show fitted.graph object

Usage

```
## S4 method for signature 'fitted.graph'
show(object)
```

Arguments

`object`	Object of class fitted.graph

show,fstats-method *Show fstats object*

Description

Show fstats object

Usage

```
## S4 method for signature 'fstats'
show(object)
```

Arguments

`object`	Object of class fstats

`show,graph.params-method` *Show graph.params object*

Description

Show graph.params object

Usage

```
## S4 method for signature 'graph.params'
show(object)
```

Arguments

`object`	Object of class graph.params

`show,pairwisefst-method` *Show pairwisefst object*

Description

Show pairwisefst object

Usage

```
## S4 method for signature 'pairwisefst'
show(object)
```

Arguments

`object`	Object of class pairwisefst

show,pooldata-method *Show pooldata object*

Description

Show pooldata object

Usage

```
## S4 method for signature 'pooldata'
show(object)
```

Arguments

`object`	Object of class pooldata

sim.readcounts *Simulate read counts from count data and return a pooldata object*

Description

Simulate read counts from count data and return a pooldata object

Usage

```
sim.readcounts(
  x,
  lambda.cov = rep(50, x@npops),
  overdisp = 1,
  seq.eps = 0,
  exp.eps = 0,
  maf.thr = 0,
  min.rc = 2,
  genome.size = 0,
  verbose = TRUE
)
```

Arguments

`x`	A countdata object containing allele count information
`lambda.cov`	Numeric vector of length npop giving the expected coverage of each pool
`overdisp`	Numeric value giving overdispersion of coverages (see details)
`seq.eps`	Numeric value giving the sequencing error rate
`exp.eps`	Numeric value giving the experimental error leading to unequal contribution of individual to the pool reads

`maf.thr`	Float giving the MAF threshold for SNP filtering
`min.rc`	Integer giving the minimal read count for an allele to be considered as true allele
`genome.size`	Size of the genome (only considered when seq.eps>0 to simulated spurious SNPs generated at monomorphic position)
`verbose`	If TRUE extra information is printed on the terminal

Details

The function implements a simulation approach similar to that described in Gautier et al. (2021). Read coverages are sampled from a distribution specified by the lambda.cov vector and the overdisp scalar. Note that overdisp is the same for all pop sample but the expected coveragese (specified in the lambda.cov vector) may vary across pool. If overdisp=1 (default), coverages are assumed Poisson distributed with mean (and variance) equal to the value specified in the lambda.cov vector. If overdisp>1, coverages follows a Negative Binomial distribution with a mean equal to lambda and variance equal to overdisp*lambda. Finally, if overdisp<1, no variation in coverage is introduced and all coverages are equal to the value specified in the lambda vector although they may (slightly) vary in the output when seq.eps>0 due to the removal of error reads. The seq.eps parameter control sequencing error rate. Sequencing errors are modeled following Gautier et al. (2021) i.e. read counts for the four possible bases are sampled from a multinomial distribution Multinom(c,{f*(1-eps)+(1-f)*eps/3;f*eps/3+(1-f)*(1-eps),eps/3,eps/3}) where c is the read coverage and f the reference allele frequencies (obtained from the count data). When seq.eps>0, spurious SNPs may be generated at monomorphic positions (the number of which being equal to the size of the genome, provided with the genome.size argument, minus the number of SNPs in the countdata object). These spurious SNPs are simulated using the same error model (Multinom(c,{1-eps,eps/3,eps/3,eps/3}). Only bi-allelic SNPs passing filtering conditions specified by min.rc (which controls the minimal read count for an allele to be deemed as true, i.e. if more than two alleles have >= min.rc counts then the SNP is excluded because non-bi-allelic) and maf.thr (threshold on the major allele frequency computed over all read counts) are included in the output. Experimental error exp.eps control the contribution of individual (assumed diploid) to the pools following the model described in Gautier et al. (2013). The parameter exp.eps corresponds to the coefficient of variation of the individual contributions. For example, in a pool of 10 individuals and a Poisson distributed coverage of mean 100, exp.eps=0.5 correspond to a situation where the 5 most contributing individuals contribute $>2$ times reads than the others. When exp.eps tends toward 0, all individuals contribute equally to the pool and there is no experimental error. Note that the number of (diploid) individuals for each SNP and pop. sample is deduced from the input total count (it may thus differ over SNP when the total counts are not the same).

Value

A pooldata object containing simulated read counts

See Also

To generate coundata object, see `genobaypass2countdata` or `genotreemix2countdata`.

Examples

```
#not run
```

`simureads_mono` *simureads_mono*

Description

Simulate read counts for monomorphic position when there is sequencing error

Usage

```
.simureads_mono(npos, npop, lambda, overdisp, min_rc, min_maf, eps)
```

Arguments

`npos`	Integer giving the number of positions (close to genome size)
`npop`	Integer giving the number of population samples
`lambda`	Numeric Vector of length npop giving the expected coverage of each pool
`overdisp`	Numeric value giving overdispersion of coverages and their distribution (see details)
`min_rc`	Integer giving the minimal read count for an allele to be considered as true allele
`min_maf`	Float giving the MAF threshold for SNP filtering
`eps`	Numeric value giving the sequencing error

Details

The function implements a simulation approach similar to that described in Gautier et al. (2021). Read coverages are sampled from a distribution specified by the lambda and overdisp vectors. Note that overdisp is the same for all pop sample but lambda (expected coverages) may vary across pool. If overdisp=1 (default in the R function), coverages are assumed Poisson distributed and the mean and variance of the coverages for the pool are both equal to the value specified in the lambda vector. If overdisp>1, coverages follows a Negative Binomial distribution with a mean equal the lamda but a variance equal to overdisp*lambda. Finally, if overdisp<1, no variation in coverage is introduced and all coverages are equal to the value specified in the lambda vector although they may (slightly) vary in the output when eps>0 due to the removal of error reads. The eps parameter control sequencing error rate. Sequencing errors are modeled following Gautier et al. (2021) i.e. read counts for the four possible bases are sampled from a multinomial distribution Multinom(c,{1-eps;eps/3,eps/3,eps/3}) where c is the read coverage. Only bi-allelic SNPs (after considering min_rc) satisfying with MAF>min_maf are included in the output.

Value

Return an Integer matrix with nsnp rows and 2*npop columns (1:npop=ref allele readcount; (npop+1):2*npop=coverage)

Examples

```
#
```

`simureads_poly` *simureads_poly*

Description

Simulate read counts from count data

Usage

```
.simureads_poly(
  y_count,
  n_count,
  lambda,
  overdisp,
  min_rc,
  min_maf,
  eps,
  eps_exp
)
```

Arguments

`y_count`	Integer Matrix with nsnp rows and npop columns giving allele counts at the reference allele
`n_count`	Integer Matrix with nsnp rows and npop columns giving total counts
`lambda`	Numeric Vector of length npop giving the expected coverage of each pool
`overdisp`	Numeric value giving overdispersion of coverages and their distribution (see details)
`min_rc`	Integer giving the minimal read count for an allele to be considered as true allele
`min_maf`	Float giving the MAF threshold for SNP filtering
`eps`	Numeric value giving the sequencing error
`eps_exp`	Numeric value giving the experimental error leading to unequal contribution of individual to the pool reads

Details

The function implements a simulation approach similar to that described in Gautier et al. (2021). Read coverages are sampled from a distribution specified by the lambda and overdisp vectors. Note that overdisp is the same for all pop sample but lambda (expected coverages) may vary across pool. If overdisp=1 (default in the R function), coverages are assumed Poisson distributed and the mean and variance of the coverages for the pool are both equal to the value specified in the lambda vector. If overdisp>1, coverages follows a Negative Binomial distribution with a mean equal the lamda but a variance equal to overdisp*lambda. Finally, if overdisp<1, no variation in coverage is introduced and all coverages are equal to the value specified in the lambda vector although they may (slightly) vary in the output when eps>0 due to the removal of error reads. The eps parameter control sequencing error rate. Sequencing errors are modeled following Gautier et al. (2021) i.e. read counts for the four possible bases are sampled from a multinomial distribution Multinom(c,{f*(1-eps)+(1-f)*eps/3;f*eps/3+(1-f)*(1-eps),eps/3,eps/3}) where c is the read coverage and f the reference allele frequencies (obtained from the count data). Experimental error eps_exp control the contribution of individual (assumed diploid) to the pools following the model described in Gautier et al. (2013). The parameter eps_exp corresponds to the coefficient of variation of the individual contributions When eps_exp tends toward 0, all individuals contribute equally to the pool and there is no experimental error. For example, with 10 individuals, eps_exp=0.5 correspond to a situation where 5 individuals contribute 2.8x more reads than the five others. Note that the number of (diploid) individuals for each SNP and pop. sample is deduced from the input total count (it may thus differ over SNP when the total counts are not the same).

Value

Return an Integer matrix with nsnp rows and 2*npop columns (1:npop=ref allele readcount; (npop+1):2*npop=coverage)

Examples

```
#
```

`sliding.windows.fstat` *Compute sliding window estimates of F-statistics or ratio of F-statistics over the genome*

Description

Compute sliding window estimates of F-statistics or ratio of F-statistics over the genome

Usage

```
sliding.windows.fstat(
  x,
  num.pop.idx = NULL,
  den.pop.idx = NULL,
  num.stat = NULL,
  den.stat = NULL,
  window.def = c("nsnp", "bp")[1],
  sliding.window.size = NULL,
  window.overlap.fact = 0.5,
  bp.start.first.snp = TRUE,
  verbose = TRUE
)
```

Arguments

`x`	A pooldata object containing Pool-Seq information or a countdata object containing allele count information
`num.pop.idx`	A vector of length 1 to 4 (depending on num.stat) giving the index (or names) of the populations samples. If num.stat="het", num.pop.idx must be of length 1: num.pop.idx=i specifies the ith pop in x. If num.stat="div", "F2" or "Fst", num.pop.idx must be of length 2: num.pop.idx=c(i,j) specifies the pairs of populations with indexes i and j in x. If num.stat="F3" or "F3star", num.pop.idx must be of length 3 (num.pop.idx=c(i,j,k) specifies the F3(pop_i;pop_j,pop_k) populations triplet). Finally, if num.stat="F4" or "Dstat", num.pop.idx must be of length 4: num.pop.idx=c(i,j,k) specifies the F4(pop_i,pop_j;pop_k,pop_l) populations quadruplet i.e. the computed (numerator) statistic computed is (F2(pop_i,pop_k)-F2(pop_i,pop_l)-F2(pop_j,pop_k)+F2(pop_j,pop_l))/2.
`den.pop.idx`	A vector of length 1 to 4 (see num.pop.idx description) giving the index of the populations specifying the F-statistic. If NULL, the computed statistic is the one specified by num.pop.idx.
`num.stat`	the name of the (numerator) stat which must be "het" (1-Q1), "div" (1-Q2), "F2", "Fst", "F3", "F3star", "F4", "Dstat", "Fh", "Fd", or "FdM"
`den.stat`	the name of the (numerator) stat which must be "het" (1-Q1), "div" (1-Q2), "F2", "Fst", "F3", "F3star", "F4", "Dstat", "Fh", "Fd", or "FdM"
`window.def`	Either "nsnp" or "bp" to define windows by either a number of SNPs or a size in bp, respectively
`sliding.window.size`	A numeric value giving the number of SNPs or the size (in bp) of the windows depending window.def
`window.overlap.fact`	A numeric value (between 0 and 1) giving the percentage of overlap between consecutive windows (default=0.5)
`bp.start.first.snp`	When window.def="bp", if TRUE (default) the windowing start at the first SNP position, if FALSE the windowing start at position 1
`verbose`	If TRUE extra information is printed on the terminal

Details

Compute sliding window estimates of F-statistics or ratio of F-statistics over the genome.

Value

A data frame with 7 columns with for each window in a row their i) chromosome/contig of origin; ii) start and iii) end position; iv) the mid-position of each window; v) the cumulated mid-position of each window (to facilitate further plotting); vi) the number of SNPs included in the computation of window value; and vii) the estimated value of the statistic

See Also

To generate pooldata object, see `vcf2pooldata`, `popsync2pooldata`,`genobaypass2pooldata` or `genoselestim2pooldata`. To generate coundata object, see `genobaypass2countdata` or `genotreemix2countdata`.

Examples

```
 make.example.files(writing.dir=tempdir())
pooldata=popsync2pooldata(sync.file=paste0(tempdir(),"/ex.sync.gz"),poolsizes=rep(50,15))
```

`vcf2pooldata` *Convert a VCF file into a pooldata object.*

Description

Convert VCF files into a pooldata object.

Usage

```
vcf2pooldata(
  vcf.file = "",
  poolsizes = NA,
  poolnames = NA,
  min.cov.per.pool = -1,
  min.rc = 1,
  max.cov.per.pool = 1e+06,
  min.maf = -1,
  remove.indels = FALSE,
  min.dist.from.indels = 0,
  nlines.per.readblock = 1e+06,
  verbose = TRUE
)
```

Arguments

`vcf.file`	The name (or a path) of the Popoolation sync file (might be in compressed format)
`poolsizes`	A numeric vector with haploid pool sizes (if NA haploid pools sizes are set to a default value of 100)
`poolnames`	A character vector with the names of pool (if NA sample names are retrieved from the vcf header)
`min.cov.per.pool`	Minimal allowed read count (per pool). If at least one pool is not covered by at least min.cov.perpool reads, the position is discarded

`min.rc`	Minimal allowed read count per base (options silenced for VarScan vcf). Bases covered by less than min.rc reads are discarded and considered as sequencing error. For instance, if nucleotides A, C, G and T are covered by respectively 100, 15, 0 and 1 over all the pools, setting min.rc to 0 will lead to discard the position (the polymorphism being considered as tri-allelic), while setting min.rc to 1 (or 2, 3..14) will make the position be considered as a SNP with two alleles A and C (the only read for allele T being disregarded). For VarScan vcf, markers with more than one alternative allele are discarded because the VarScan AD field only contains one alternate read count.
`max.cov.per.pool`	Maximal allowed read count (per pool). If at least one pool is covered by more than min.cov.perpool reads, the position is discarded
`min.maf`	Minimal allowed Minor Allele Frequency (computed from the ratio overall read counts for the reference allele over the read coverage)
`remove.indels`	Remove indels identified using the number of characters of the alleles in the REF or ALT fields (i.e., if at least one allele is more than 1 character, the position is discarded)
`min.dist.from.indels`	Remove SNPs within min.dist.from.indels from an indel i.e. SNP with position p verifying (indel.pos-min.dist)<=p<=(indel.pos+min.dist+l.indels-1) where l.indel=length of the ref. indel allele. If min.dist.from.indels>0, INDELS are also removed (i.e., remove.indels is set to TRUE).
`nlines.per.readblock`	Number of Lines read simultaneously. Should be adapted to the available RAM.
`verbose`	If TRUE extra information is printed on the terminal

Details

Genotype format in the vcf file for each pool is assumed to contain either i) an AD field containing allele counts separated by a comma (as produced by popular software such as GATK or samtools/bcftools) or ii) both a RD (reference allele count) and a AD (alternate allele count) as obtained with the VarScan mpileup2snp program (when run with the –output-vcf option). The underlying format is automatically detected by the function. For VarScan generated vcf, it should be noticed that SNPs with more than one alternate allele are discarded (because only a single count is then reported in the AD fields) making the min.rc unavailable. The VarScan –min-reads2 option might replace to some extent this functionality although SNP where the two major alleles in the Pool-Seq data are different from the reference allele (e.g., expected to be more frequent when using a distantly related reference genome for mapping) will be disregarded.

Value

A pooldata object containing 7 elements:

1. "refallele.readcount": a matrix with nsnp rows and npools columns containing read counts for the reference allele (chosen arbitrarily) in each pool
2. "readcoverage": a matrix with nsnp rows and npools columns containing read coverage in each pool

3. "snp.info": a matrix with nsnp rows and four columns containing respectively the contig (or chromosome) name (1st column) and position (2nd column) of the SNP; the allele taken as reference in the refallele.readcount matrix (3rd column); and the alternative allele (4th column)
4. "poolsizes": a vector of length npools containing the haploid pool sizes
5. "poolnames": a vector of length npools containing the names of the pools
6. "nsnp": a scalar corresponding to the number of SNPs
7. "npools": a scalar corresponding to the number of pools

Examples

```
make.example.files(writing.dir=tempdir())
pooldata=vcf2pooldata(vcf.file=paste0(tempdir(),"/ex.vcf.gz"),poolsizes=rep(50,15))
```

Index

.block_sum *(block_sum)*, 6
.block_sum2 *(block_sum2)*, 6
.compute_F3fromF2 *(compute_F3fromF2)*, 15
.compute_F3fromF2samples
    (compute_F3fromF2samples), 15
.compute_F4DfromF2samples
    (compute_F4DfromF2samples), 16
.compute_F4fromF2 *(compute_F4fromF2)*, 17
.compute_F4fromF2samples
    (compute_F4fromF2samples), 18
.compute_H1 *(compute_H1)*, 18
.compute_Q2 *(compute_Q2)*, 19
.compute_QmatfromF2samples
    (compute_QmatfromF2samples), 20
.compute_blockDdenom
    (compute_blockDdenom), 14
.compute_snpFstAov *(compute_snpFstAov)*,
    21
.compute_snpHierFstAov
    (compute_snpHierFstAov), 21
.compute_snpQ1 *(compute_snpQ1)*, 22
.compute_snpQ1onepop
    (compute_snpQ1onepop), 23
.compute_snpQ1rw *(compute_snpQ1rw)*, 24
.compute_snpQ2 *(compute_snpQ2)*, 25
.compute_snpQ2onepair
    (compute_snpQ2onepair), 25
.compute_snpQ2rw *(compute_snpQ2rw)*, 26
.extract_allele_names
    (extract_allele_names), 32
.extract_nonvscan_counts
    (extract_nonvscan_counts), 33
.extract_vscan_counts
    (extract_vscan_counts), 34
.find_indelneighbor_idx
    (find_indelneighbor_idx), 36
.generateF3names *(generateF3names)*, 44
.generateF4names *(generateF4names)*, 44
.scan_allele_info *(scan_allele_info)*, 77
.simureads_mono *(simureads_mono)*, 82
.simureads_poly *(simureads_poly)*, 83

add.leaf, 3, *52*, *76*

bjack_cov, 5
block_sum, 6
block_sum2, 6

compare.fitted.fstats, 7, *39*
compute.f4ratio, 8
compute.fstats, *7*, 9, *36*, *42*, *64*
compute.pairwiseFST, *9*, *10*, 11, *41*, *61*, *64*
compute_blockDdenom, 14
compute_F3fromF2, 15
compute_F3fromF2samples, 15
compute_F4DfromF2samples, 16
compute_F4fromF2, 17
compute_F4fromF2samples, 18
compute_H1, 18
compute_Q2, 19
compute_QmatfromF2samples, 20
compute_snpFstAov, 21
compute_snpHierFstAov, 21
compute_snpQ1, 22
compute_snpQ1onepop, 23
compute_snpQ1rw, 24
compute_snpQ2, 25
compute_snpQ2onepair, 25
compute_snpQ2rw, 26
computeFST, 12
countdata, *31*
countdata *(countdata-class)*, 27
countdata-class, 27
countdata.subset, 27
countdata2genobaypass, 29

data.merge, 30, *71*

extract_allele_names, 32
extract_nonvscan_counts, 33

extract_vscan_counts, 34

find.tree.popset, 35
find_indelneighbor_idx, 36
fit.graph, *5*, *7*, 37, *40*, *42*, *52*, *53*, *76*
fitted.graph *(*fitted.graph-class*)*, 39
fitted.graph-class, 39
fstats *(*fstats-class*)*, 40
fstats-class, 40

gen.fitted.graph *(*fitted.graph-class*)*, 39
generate.graph.params, *5*, *39*, 41, *52–55*, *76*
generate.jackknife.blocks, 43
generateF3names, 44
generateF4names, 44
genobaypass2countdata, *8*, *10*, *12*, *14*, *27*, *29*, *30*, 45, *75*, *81*, *86*
genobaypass2pooldata, *8*, *10*, *12*, *14*, 46, *65*, *75*, *86*
genoselestim2pooldata, *8*, *10*, *12*, *14*, 48, *65*, *75*, *86*
genotreemix2countdata, *8*, *10*, *12*, *14*, *27*, *29*, *30*, 49, *75*, *81*, *86*
graph.builder, 51
graph.params *(*graph.params-class*)*, 52
graph.params-class, 52
graph.params2qpGraphFiles, *42*, *53*, 54
graph.params2symbolic.fstats, *42*, *53*, 55

heatmap,pairwisefst-method, 56

is.countdata, 58
is.fitted.graph, 58
is.fstats, 58
is.graph.params, 59
is.pairwisefst, 59
is.pooldata, 59

make.example.files, 60

pairwisefst *(*pairwisefst-class*)*, 60
pairwisefst-class, 60
plot, *74*
plot,fitted.graph-method, 61
plot,fstats-method, 61
plot,graph.params-method, 62
plot,pairwisefst-method, 62
plot_fstats, *61*, *62*, 63
pooldata, *31*
pooldata *(*pooldata-class*)*, 64
pooldata-class, 64
pooldata.subset, 65, *67*
pooldata2diyabc, 67
pooldata2genobaypass, 68
pooldata2genoselestim, 69
poolify, 70
poppair_idx, 72
popsync2pooldata, *8*, *10*, *12*, *14*, *65*, *67–70*, 72, *75*, *86*

randomallele.pca, 74
rooted.njtree.builder, 75

scan_allele_info, 77
show,countdata-method, 77
show,fitted.graph-method, 78
show,fstats-method, 78
show,graph.params-method, 79
show,pairwisefst-method, 79
show,pooldata-method, 80
sim.readcounts, *71*, 80
simureads_mono, 82
simureads_poly, 83
sliding.windows.fstat, 84

vcf2pooldata, *8*, *10*, *12*, *14*, *33*, *36*, *65*, *67–70*, *75*, *86*, 86