Package: polyRAD (via r-universe)

October 8, 2024

Version 2.0.0.9003

Date 2023-12-16

Title Genotype Calling with Uncertainty from Sequencing Data in Polyploids and Diploids

Author Lindsay V. Clark [aut, cre] (<https://orcid.org/0000-0002-3881-9252>), U.S. National Science Foundation [fnd]

Maintainer Lindsay V. Clark <Lindsay.Clark@seattlechildrens.org>

Depends R (>= 3.5.0), methods

Imports fastmatch, pcaMethods, Rcpp, stringi

Suggests rrBLUP, Rsamtools, GenomeInfoDb, Biostrings, GenomicRanges, VariantAnnotation, SummarizedExperiment, S4Vectors, IRanges, BiocGenerics, knitr, rmarkdown, GenomicFeatures, ggplot2, adegenet

LinkingTo Rcpp

VignetteBuilder knitr, rmarkdown

Description Read depth data from genotyping-by-sequencing (GBS) or restriction site-associated DNA sequencing (RAD-seq) are imported and used to make Bayesian probability estimates of genotypes in polyploids or diploids. The genotype probabilities, posterior mean genotypes, or most probable genotypes can then be exported for downstream analysis. 'polyRAD' is described by Clark et al. (2019) <doi:10.1534/g3.118.200913>, and the Hind/He statistic for marker filtering is described by Clark et al. (2022) <doi:10.1186/s12859-022-04635-9>. A variant calling pipeline for highly duplicated genomes is also included and is described by Clark et al. (2020, Version 1) <doi:10.1101/2020.01.11.902890>.

License GPL (>= 2)

URL https://github.com/lvclark/polyRAD

Repository https://lvclark.r-universe.dev

RemoteUrl https://github.com/lvclark/polyRAD

RemoteRef HEAD

RemoteSha 6edf95b72ebf6962cc841fb9e6f089777029e6aa

Contents

Accessors *Accessor Functions for RADdata Objects*

Description

These functions can be used for accessing and replacing data within a "RADdata" object. Data slots that do not yet have accessors can be accessed and replaced using the $ operator or the attr function.

Usage

```
GetTaxa(object, ...)
GetLoci(object, ...)
GetLocDepth(object, ...)
GetContamRate(object, ...)
SetContamRate(object, value, ...)
nTaxa(object, ...)
nLoci(object, ...)
nAlleles(object, ...)
GetAlleleNames(object, ...)
GetTaxaPloidy(object, ...)
SetTaxaPloidy(object, value, ...)
GetTaxaByPloidy(object, ...)

## S3 method for class 'RADdata'
GetTaxaByPloidy(object, ploidy, ...)
```

Arguments

object	A "RADdata" object.
value	A value to assign. For SetContamRate, a number generally ranging from zero to 0.01 indicating the expected rate of sample cross-contamination. For SetTaxaPloidy, a vector of integers indicating ploidy, with one value for each taxon. If the vector for SetTaxaPloidy is named, the names should correspond to taxa names in the object.
ploidy	An integer indicating a single ploidy for which to return taxa.
...	Additional arguments (none currently supported).

Value

For GetTaxa and GetLoci, a character vector listing taxa names or loci names, respectively. For GetLocDepth, a named matrix with taxa in rows and loci in columns, giving the total read depth for each taxon and locus. For GetContamRate, a number indicating the expected contamination rate that is stored in the object. For SetContamRate, a "RADdata" object with an updated contamination rate. For nTaxa, the number of taxa in the object. For nLoci, the number of loci in the object. For nAlleles, the number of alleles across all loci in the object. For GetAlleleNames, the names of all alleles. For GetTaxaPloidy, a named integer vector indicating the ploidy of each taxon. For SetTaxaPloidy, a "RADdata" object with the taxa ploidies updated. For GetTaxaByPloidy, a character vector listing taxa.

Author(s)

Lindsay V. Clark

See Also

SetBlankTaxa for functions that assign taxa to particular roles.

Examples

```
data(exampleRAD)
GetTaxa(exampleRAD)
GetLoci(exampleRAD)
GetLocDepth(exampleRAD)
GetContamRate(exampleRAD)
exampleRAD <- SetContamRate(exampleRAD, 0.0000001)
GetContamRate(exampleRAD)
nTaxa(exampleRAD)
nAlleles(exampleRAD)
GetAlleleNames(exampleRAD)
GetTaxaPloidy(exampleRAD)
exampleRAD <- SetTaxaPloidy(exampleRAD, rep(c(2, 5), time = c(75, 25)))
GetTaxaByPloidy(exampleRAD, 2)
```

AddAlleleFreqByTaxa *Estimate Local Allele Frequencies for Each Taxon Based on Population Structure*

Description

This function estimates allele frequencies per taxon, rather than for the whole population. The best estimated genotypes (either object$depthRatio or GetWeightedMeanGenotypes(object)) are regressed against principal coordinate axes. The regression coefficients are then in turn used to predict allele frequencies from PC axes. Allele frequencies outside of a user-defined range are then adjusted so that they fall within that range.

Usage

```
AddAlleleFreqByTaxa(object, ...)
## S3 method for class 'RADdata'
AddAlleleFreqByTaxa(object, minfreq = 0.0001, ...)
```

Arguments

`object`	A `"RADdata"` object. `AddPCA` should have already been run.
`minfreq`	The minimum allowable allele frequency to be output. The maximum allowable allele frequency will be calculated as `1 - minfreq`.
`...`	Additional arguments (none implemented).

Details

For every allele, all PC axes stored in `object$PCA` are used for generating regression coefficients and making predictions, regardless of whether they are significantly associated with the allele.

`object$depthRatio` has missing data for loci with no reads; these missing data are omitted on a per-allele basis when calculating regression coefficients. However, allele frequencies are output for all taxa at all alleles, because there are no missing data in the PC axes. The output of `GetWeightedMeanGenotypes` has no missing data, so missing data are not an issue when calculating regression coefficients using that method.

After predicting allele frequencies from the regression coefficients, the function loops through all loci and taxa to adjust allele frequencies if necessary. This is needed because otherwise some allele frequencies will be below zero or above one (typically in subpopulations where alleles are near fixation), which interferes with prior genotype probability estimation. For a given taxon and locus, any allele frequencies below `minfreq` are adjusted to be equal to `minfreq`, and any allele frequencies above `1 - minfreq` are adjusted to be `1 - minfreq`. Remaining allele frequencies are adjusted so that all allele frequencies for the taxon and locus sum to one.

Value

A `"RADdata"` object identical to the one passed to the function, but with a matrix of allele frequencies added to the `$alleleFreqByTaxa` slot. Taxa are in rows and alleles in columns.

Author(s)

Lindsay V. Clark

See Also

`AddGenotypePriorProb_ByTaxa`

Examples

```
# load data
data(exampleRAD)
# do PCA
exampleRAD <- AddPCA(exampleRAD, nPcsInit = 3)
```

```
# get allele frequencies
exampleRAD <- AddAlleleFreqByTaxa(exampleRAD)

exampleRAD$alleleFreqByTaxa[1:10,]
```

AddAlleleFreqHWE *Estimate Allele Frequencies in a RADdata Object Assuming Hardy-Weinberg Equilibrium*

Description

Allele frequencies are estimated based on the best parameters available. `object$alleleFreqByTaxa` is used if available. If `object$alleleFreqByTaxa` is null, `GetWeightedMeanGenotypes` is used, and if that isn't possible `object$depthRatio` is used. From whichever of the three options is used, column means are taken, the output of which is stored as `object$alleleFreq`.

Usage

```
AddAlleleFreqHWE(object, ...)
## S3 method for class 'RADdata'
AddAlleleFreqHWE(object, excludeTaxa = GetBlankTaxa(object), ...)
```

Arguments

`object`	A "`RADdata`" object.
`excludeTaxa`	A character vector indicating taxa that should be excluded from the calculation.
`...`	Included to allow more arguments in the future, although none are currently used.

Value

A "`RADdata`" object identical to the one passed to the function, but with allele frequencies added to `object$alleleFreq`, and "`HWE`" as the value for the "`alleleFreqType`" attribute.

Author(s)

Lindsay V. Clark

See Also

`AddAlleleFreqMapping`, `AddGenotypePriorProb_HWE`

Examples

```
# load in an example dataset
data(exampleRAD)
exampleRAD

# add allele frequencies
exampleRAD <- AddAlleleFreqHWE(exampleRAD)
exampleRAD$alleleFreq
```

AddAlleleFreqMapping *Estimate Allele Frequencies in a Mapping Population*

Description

Estimate allele frequencies using data from a mapping population, assuming a fixed set of allele frequencies are possible.

Usage

```
AddAlleleFreqMapping(object, ...)
## S3 method for class 'RADdata'
AddAlleleFreqMapping(object, expectedFreqs = seq(0, 1, 0.25),
                     allowedDeviation = 0.05,
                     excludeTaxa = c(GetDonorParent(object),
                                     GetRecurrentParent(object),
                                     GetBlankTaxa(object)), ...)
```

Arguments

`object` A "`RADdata`" object. The donor and recurrent parent should have been assigned with `SetDonorParent` and `SetRecurrentParent`, respectively. If this is not a backcross population, it does not matter which is the donor or recurrent parent.

`expectedFreqs` A numeric vector listing all expected allele frequencies in the mapping population.

`allowedDeviation`
A value indicating how far an observed allele frequency can deviate from an expected allele frequency and still be categorized as that allele frequency. Must be no more than half the smallest interval seen in `expectedFreqs`.

`excludeTaxa` A character vector indicating taxa that should be excluded from the allele frequency estimate.

`...` Arguments to be passed to the method for "`RADdata`".

Details

Allele frequencies are first estimated as the column means of object$depthRatio (unless posterior genotype probabilities and ploidy chi-squared values have already been calculated, in which case GetWeightedMeanGenotypes is run and the column means of its output are taken), excluding any taxa listed in excludeTaxa. These are then categorized based on which, if any, expected allele frequency they match with, based on the intervals described by expectedFreqs and allowedDeviation. If an allele frequency does not fall within any of these intervals it is classified as NA; otherwise it is converted to the matching value in expectedFreqs.

Value

A "RADdata" object identical to the one passed to the function, but with allele frequencies added to object$alleleFreq, and "mapping" as the "alleleFreqType" attribute.

Author(s)

Lindsay V. Clark

See Also

AddAlleleFreqHWE

Examples

```
# load example dataset
data(exampleRAD_mapping)
exampleRAD_mapping

# specify parents
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")

# estimate allele frequencies in diploid BC1 population
exampleRAD_mapping <- AddAlleleFreqMapping(exampleRAD_mapping,
                        expectedFreqs = c(0.25, 0.75),
                        allowedDeviation = 0.08)

exampleRAD_mapping$alleleFreq
```

AddAlleleLinkages *Identify and Utilize Linked Alleles for Estimating Genotype Priors*

Description

AddAlleleLinkages finds alleles, if any, in linkage disequilibrium with each allele in a RADdata object, and computes a correlation coefficient representing the strength of the linkage. AddGenotypePriorProb_LD adds a second set of prior genotype probabilities to a RADdata object based on the genotype posterior probabilities at linked alleles.

Usage

```
AddAlleleLinkages(object, ...)
## S3 method for class 'RADdata'
AddAlleleLinkages(object, type, linkageDist, minCorr,
                  excludeTaxa = character(0), ...)

AddGenotypePriorProb_LD(object, ...)
## S3 method for class 'RADdata'
AddGenotypePriorProb_LD(object, type, ...)
```

Arguments

`object` A `RADdata` object with genomic alignment data stored in `object$locTable$Chr` and `object$locTable$pos`.

`type` A character string, either “mapping”, “hwe”, or “popstruct”, to indicate the type of population being analyzed.

`linkageDist` A number, indicating the distance in basepairs from a locus within which to search for linked alleles.

`minCorr` A number ranging from zero to one indicating the minimum correlation needed for an allele to be used for genotype prediction at another allele.

`excludeTaxa` A character vector listing taxa to be excluded from correlation estimates.

`...` Additional arguments (none implemented).

Details

These functions are primarily designed to be used internally by the pipeline functions.

`AddAlleleLinkages` obtains genotypic values using `GetWeightedMeanGenotypes`, then regresses those values for a given allele against those values for nearby alleles to obtain correlation coefficients. For the population structure model, the genotypic values for an allele are first regressed on the PC axes from `object$PCA`, then the residuals are regressed on the genotypic values at nearby alleles to obtain correlation coefficients.

`AddGenotypePriorProb_LD` makes a second set of priors in addition to `object$priorProb`. This second set of priors has one value per inheritance mode per taxon per allele per possible allele copy number. Where K is the ploidy, with allele copy number c ranging from 0 to K, i is an allele, j is a linked allele at a different locus out of J total alleles linked to i, r_{ij} is the correlation coefficient between those alleles, t is a taxon, $post_{cjt}$ is the posterior probability of a given allele copy number for a given allele in a given taxon, and $prior_{cit}$ is the prior probability for a given allele copy number for a given allele in a given taxon based on linkage alone:

$$prior_{cit} = \frac{\prod_{j=1}^{J} post_{cjt} * r_{ij} + (1 - r_{ij})/(K+1)}{\sum_{c=0}^{K} \prod_{j=1}^{J} post_{cjt} * r_{ij} + (1 - r_{ij})/(K+1)}$$

For mapping populations, `AddGenotypePriorProb_LD` uses the above formula when each allele only has two possible genotypes (i.e. test-cross segregation). When more genotypes are possible, `AddGenotypePriorProb_LD` instead estimates prior probabilities as fitted values when the posterior probabilities for a given allele are regressed on the posterior probabilities for a linked allele. This allows loci with different segregation patterns to be informative for predicting genotypes, and for cases where two alleles are in phase for some but not all parental copies.

Value

A `RADdata` object is returned. For `AddAlleleLinkages`, it has a new slot called `$alleleLinkages` that is a list, with one item in the list for each allele in the dataset. Each item is a data frame, with indices for linked alleles in the first column, and correlation coefficients in the second column.

For `AddGenotypePriorProb_LD`, the object has a new slot called `$priorProbLD`. This is a list much like `$posteriorProb`, with one list item per inheritance mode, and each item being an array with allele copy number in the first dimension, taxa in the second dimension, and alleles in the third dimension. Values indicate genotype prior probabilities based on linked alleles alone.

Author(s)

Lindsay V. Clark

See Also

`AddGenotypePriorProb_HWE`

Examples

```
# load example dataset
data(Msi01genes)

# Run non-LD pop structure pipeline
Msi01genes <- IteratePopStruct(Msi01genes, tol = 0.01, nPcsInit = 10)

# Add linkages
Msi01genes <- AddAlleleLinkages(Msi01genes, "popstruct", 1e4, 0.05)
# Get new posterior probabilities based on those linkages
Msi01genes <- AddGenotypePriorProb_LD(Msi01genes, "popstruct")

# Preview results
Msi01genes$priorProbLD[[1,2]][,1:10,1:10]
```

`AddGenotypeLikelihood` *Estimate Genotype Likelihoods in a RADdata object*

Description

For each possible allele copy number across each possible ploidy in each taxon, `AddGenotypeLikelihood` estimates the probability of observing the distribution of read counts that are recorded for that taxon and locus. `AddDepthSamplingPermutations` is called by `AddGenotypeLikelihood` the first time it is run, so that part of the likelihood calcluation is stored in the `RADdata` object and doesn't need to be re-run on each iteration of the pipeline functions.

Usage

```
AddGenotypeLikelihood(object, ...)

## S3 method for class 'RADdata'
AddGenotypeLikelihood(object, overdispersion = 9, ...)

AddDepthSamplingPermutations(object, ...)
```

Arguments

`object`	A "RADdata" object.
`overdispersion`	An overdispersion parameter. Higher values will cause the expected read depth distribution to more resemble the binomial distribution. Lower values indicate more overdispersion, *i.e.* sample-to-sample variance in the probability of observing reads from a given allele.
`...`	Other arguments; none are currently used.

Details

If allele frequencies are not already recorded in `object`, they will be added using `AddAlleleFreqHWE`. Allele frequencies are then used for estimating the probability of sampling an allele from a genotype due to sample contamination. Given a known genotype with x copies of allele i, ploidy k, allele frequency p_i in the population used for making sequencing libraries, and contamination rate c, the probabiity of sampling a read r_i from that locus corresponding to that allele is

$$P(r_i|x) = \frac{x}{k} * (1 - c) + p_i * c$$

To estimate the genotype likelihood, where nr_i is the number of reads corresponding to allele i for a given taxon and locus and nr_j is the number of reads corresponding to all other alleles for that taxon and locus:

$$P(nr_i, nr_j|x) = \binom{nr_i + nr_j}{nr_i} * \frac{B[P(r_i|x) * d + nr_i, [1 - P(r_i|x)] * d + nr_j]]}{B[P(r_i|x) * d, [1 - P(r_i|x)] * d]}$$

where

$$\binom{nr_i + nr_j}{nr_i} = \frac{(nr_i + nr_j)!}{nr_i! * nr_j!}$$

B is the beta function, and d is the overdispersion parameter set by `overdispersion`. $\binom{nr_i+nr_j}{nr_i}$ is calculated by `AddDepthSamplingPermutations`.

Value

A `"RADdata"` object identical to that passed to the function, but with genotype likelihoods stored in `object$genotypeLikelihood`. This item is a two dimensional list, with one row for each ploidy listed in `object$possiblePloidies`, ignoring differences between autopolyploids and allopolyploids, and one column for each ploidy listed in `object$taxaPloidy`. Each item in the list is a three-dimensional array with number of allele copies in the first dimension, taxa in the second dimension, and alleles in the third dimension.

Author(s)

Lindsay V. Clark

See Also

AddAlleleFreqMapping

Examples

```
# load example dataset and add allele frequency
data(exampleRAD)
exampleRAD <- AddAlleleFreqHWE(exampleRAD)

# estimate genotype likelihoods
exampleRAD <- AddGenotypeLikelihood(exampleRAD)

# inspect the results
# the first ten individuals and first two alleles, assuming diploidy
exampleRAD$alleleDepth[1:10,1:2]
exampleRAD$genotypeLikelihood[[1]][,1:10,1:2]
# assuming tetraploidy
exampleRAD$genotypeLikelihood[[2]][,1:10,1:2]
```

AddGenotypePosteriorProb *Estimate Posterior Probabilities of Genotypes*

Description

Given a "RADdata" object containing genotype prior probabilities and genotype likelihoods, this function estimates genotype posterior probabilities and adds them to the $posteriorProb slot of the object.

Usage

```
AddGenotypePosteriorProb(object, ...)
```

Arguments

object	A "RADdata" object. Prior genotype probabilities and genotype likelihood should have already been added.
...	Potential future arguments (none currently in use).

Value

A "RADdata" object identical to that passed to the function, but with a two-dimensional list added to the $posteriorProb slot. Rows of the list correspont to object$possiblePloidies, and columns to unique values in object$taxaPloidy, similarly to object$priorProb. Each item of the list is a three dimensional array, with allele copy number in the first dimension, taxa in the second dimension, and alleles in the third dimension. For each allele and taxa, posterior probabilities will sum to one across all potential allele copy numbers.

Author(s)

Lindsay V. Clark

See Also

AddGenotypeLikelihood, AddGenotypePriorProb_Mapping2Parents

Examples

```
# load dataset and set some parameters
data(exampleRAD_mapping)
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")
exampleRAD_mapping <- AddAlleleFreqMapping(exampleRAD_mapping,
                                     expectedFreqs = c(0.25, 0.75),
                                     allowedDeviation = 0.08)
exampleRAD_mapping <- AddGenotypeLikelihood(exampleRAD_mapping)
exampleRAD_mapping <- AddGenotypePriorProb_Mapping2Parents(exampleRAD_mapping,
                                                n.gen.backcrossing = 1)

# estimate posterior probabilities
exampleRAD_mapping <- AddGenotypePosteriorProb(exampleRAD_mapping)
# examine the results
exampleRAD_mapping$posteriorProb[[1,1]][,3,]
```

AddGenotypePriorProb_ByTaxa

Estimate Prior Genotype Probabilities on a Per-Taxon Basis

Description

Using local allele frequencies estimated by AddAlleleFreqByTaxa and assuming Hardy-Weinberg Equilibruim or inbreeding on a local scale, AddGenotypePriorProb_ByTaxa estimates prior genotype probabilities at each taxon, allele, and possible ploidy. These are then stored in the $priorProb slot of the "RADdata" object.

Usage

```
AddGenotypePriorProb_ByTaxa(object, ...)
## S3 method for class 'RADdata'
AddGenotypePriorProb_ByTaxa(object, selfing.rate = 0, ...)
```

Arguments

`object` A "RADdata" object. AddAlleleFreqByTaxa should have already been run.

`selfing.rate` A number ranging from zero to one indicating the frequency of self-fertilization in the species.

`...` Additional arguments (none implemented).

Value

A "RADdata" object identical to that passed to the function, but with a two-dimensional list added to the `$priorProb` slot. Each row in the list corresponds to one ploidy in `object$possiblePloidies`, and each column to a unique ploidy in `object$taxaPloidy`. Each item is a three-dimensional array with allele copy number in the first dimension, taxa in the second dimension, and alleles in the third dimension. The values in the array are prior genotype probabilities. Additionally, "taxon" is recorded in the "priorType" attribute.

Author(s)

Lindsay V. Clark

See Also

AddGenotypePriorProb_HWE for equations used for genotype prior probability estimation.

AddGenotypePriorProb_Mapping2Parents, AddGenotypeLikelihood

Examples

```
# load data
data(exampleRAD)
# do PCA
exampleRAD <- AddPCA(exampleRAD, nPcsInit = 3)
# get allele frequencies
exampleRAD <- AddAlleleFreqByTaxa(exampleRAD)

# add prior probabilities
exampleRAD <- AddGenotypePriorProb_ByTaxa(exampleRAD)

exampleRAD$priorProb[[1,1]][,1,]
exampleRAD$priorProb[[2,1]][,1,]
exampleRAD$priorProb[[1,1]][,2,]
exampleRAD$priorProb[[2,1]][,2,]
exampleRAD$priorProb[[1,2]][,1,]

# try it with inbreeding, for diploid samples only
exampleRAD2 <- SubsetByTaxon(exampleRAD, GetTaxa(exampleRAD)[exampleRAD$taxaPloidy == 2])
exampleRAD2 <- AddGenotypePriorProb_ByTaxa(exampleRAD2, selfing.rate = 0.5)

exampleRAD2$priorProb[[1,1]][,1,]
```

AddGenotypePriorProb_Even

Add Uniform Priors to a RADdata Object

Description

To estimate genotype posterior probabilities based on read depth alone, without taking any population parameters into account, this function can be used to set a uniform prior probability on all possible genotypes. This function is not part of any pipeline but can be used for very rough and quick genotype estimates, when followed by AddGenotypeLikelihood, AddGenotypePosteriorProb, AddPloidyChiSq, and GetWeightedMeanGenotypes or GetProbableGenotypes.

Usage

```
AddGenotypePriorProb_Even(object, ...)
```

Arguments

object	A RADdata object.
...	Additional arguments (none implemented).

Value

A "RADdata" object identical that passed to the function, but with data stored in one new slot:

priorProb	A two-dimensional list of matrices, with rows corresponding to object$possiblePloidies and columns corresponding to unique values in object$taxaPloidy. Each item in the list is a matrix. For each matrix, allele copy number (from zero to the total ploidy) is in rows, and alleles are in columns. Each value is $1/(ploidy + 1)$.

Note

Values in object$ploidyChiSq may not be particularly meaningful under uniform priors.

Author(s)

Lindsay V. Clark

See Also

AddGenotypePriorProb_HWE

Examples

```
data(exampleRAD)

exampleRAD <- AddGenotypePriorProb_Even(exampleRAD)
exampleRAD$priorProb

# finish protocol to get genotypes
exampleRAD <- AddGenotypeLikelihood(exampleRAD)
exampleRAD <- AddPloidyChiSq(exampleRAD)
exampleRAD <- AddGenotypePosteriorProb(exampleRAD)

genmat <- GetWeightedMeanGenotypes(exampleRAD)
genmat
```

AddGenotypePriorProb_HWE

Estimate Genotype Prior Probabilities In the Absence of Population Structure

Description

Assuming Hardy-Weinberg Equilibrium, this function uses allele frequencies and possible ploidies stored in a "RADdata" object to estimate genotype frequencies in the population, then stores these genotype frequencies in the `$priorProb` slot. Inbreeding can also be simulated using the `selfing.rate` argument.

Usage

```
AddGenotypePriorProb_HWE(object, ...)
## S3 method for class 'RADdata'
AddGenotypePriorProb_HWE(object, selfing.rate = 0, ...)
```

Arguments

`object`	A "RADdata" object that has had allele frequencies added with `AddAlleleFreqHWE`.
`selfing.rate`	A number ranging from zero to one indicating the frequency of self-fertilization in the species.
`...`	Additional arguments (none currently implemented).

Details

For an autopolyploid, or within one subgenome of an allopolyploid, genotype prior probabilities are estimated as:

$$P(G_i) = \binom{k}{i} p^i * (1-p)^{k-i}$$

where k is the ploidy, i is the copy number of a given allele, and p is the allele frequency in the population.

If the selfing rate is above zero and ploidy is even, genotype prior probabilities are adjusted according to Equation 6 of de Silva et al. (2005):

$$P(G_{self}) = (1 - s)(I - sA)^{-1}P(G)$$

where s is the selfing rate. A is a $k + 1 \times k + 1$ matrix, with each column representing the allele copy number from 0 to k of a parental genotype, and each row representing the allele copy number from 0 to k of a progeny genotype, and matrix elements representing the frequencies of progeny after self-fertilization (each column summing to one).

Value

A "RADdata" object identical that passed to the function, but with data stored in one new slot:

`priorProb`	A two-dimensional list of matrices, with rows corresponding to `object$possiblePloidies` and columns corresponding to unique values in `object$taxaPloidy`. Each item in the list is a matrix. For each matrix, allele copy number (from zero to the total ploidy) is in rows, and alleles are in columns. Each value is the probability of sampling an individual with that allele copy number from the population.

Author(s)

Lindsay V. Clark

References

De Silva, H. N., Hall, A. J., Rikkerink, E., and Fraser, L. G. (2005) Estimation of allele frequencies in polyploids under certain patterns of inheritance. *Heredity* **95**, 327–334. doi:10.1038/sj.hdy.6800728

See Also

`AddGenotypePriorProb_Mapping2Parents`, `AddGenotypeLikelihood`, `AddGenotypePriorProb_ByTaxa`

Examples

```
# load in an example dataset
data(exampleRAD)
# add allele frequencies
exampleRAD <- AddAlleleFreqHWE(exampleRAD)
# add inheritance modes
exampleRAD$possiblePloidies <- list(2L, 4L, c(2L, 2L))

# estimate genotype prior probabilities
exampleRAD <- AddGenotypePriorProb_HWE(exampleRAD)

# examine results
exampleRAD$alleleFreq
```

```
exampleRAD$priorProb

# try it with inbreeding, for diploids only
exampleRAD2 <- SubsetByTaxon(exampleRAD, GetTaxa(exampleRAD)[exampleRAD$taxaPloidy == 2])
exampleRAD2 <- AddGenotypePriorProb_HWE(exampleRAD2, selfing.rate = 0.5)
exampleRAD2$priorProb
```

AddGenotypePriorProb_Mapping2Parents

Expected Genotype Frequencies in Mapping Populations

Description

EstimateParentalGenotypes estimates the most likely genotypes of two parent taxa. Using those parental genotypes, AddGenotypePriorProb_Mapping2Parents estimates expected genotype frequencies for a population of progeny, which are added to the "RADdata" object in the $priorProb slot.

Usage

```
AddGenotypePriorProb_Mapping2Parents(object, ...)
## S3 method for class 'RADdata'
AddGenotypePriorProb_Mapping2Parents(object,
    donorParent = GetDonorParent(object),
    recurrentParent = GetRecurrentParent(object),
    n.gen.backcrossing = 0, n.gen.intermating = 0, n.gen.selfing = 0,
    minLikelihoodRatio = 10, ...)

EstimateParentalGenotypes(object, ...)
## S3 method for class 'RADdata'
EstimateParentalGenotypes(object,
    donorParent = GetDonorParent(object),
    recurrentParent = GetRecurrentParent(object),
    n.gen.backcrossing = 0, n.gen.intermating = 0, n.gen.selfing = 0,
    minLikelihoodRatio = 10, ...)
```

Arguments

object
: A "RADdata" object. Ideally this should be set up as a mapping population using SetDonorParent, SetRecurrentParent, and AddAlleleFreqMapping.

...
: Additional arguments, listed below, to be passed to the method for "RADdata" objects.

donorParent
: A character string indicating which taxon is the donor parent. If backcrossing was not performed, it does not matter which was the donor or recurrent parent.

recurrentParent
: A character string indicating which taxon is the recurrent parent.

`n.gen.backcrossing`
An integer, zero or greater, indicating how many generations of backcrossing to the recurrent parent were performed.

`n.gen.intermating`
An integer, zero or greater, indicating how many generations of intermating within the population were performed. (Values above one should not have an effect on the genotype priors that are output, *i.e.* genotype probabilities after one generation of random mating are identical to genotype probabilities after >1 generation of random mating, assuming no genetic drift or selection).

`n.gen.selfing` An integer, zero or greater, indicating how many generations of selfing were performed.

`minLikelihoodRatio`
The minimum likelihood ratio for determining parental genotypes with confidence, to be passed to `GetLikelyGen` for both parental taxa.

Details

`AddGenotypePriorProb_Mapping2Parents` examines the parental and progeny ploidies stored in `object$taxaPloidy` and throws an error if they do not meet expectations. In particular, all progeny must be the same ploidy, and that must be the ploidy that would be expected if the parents produced normal gametes. For example in an F1 cross, if one parent was diploid and the other tetraploid, all progeny must be triploid. If both parents are tetraploid, all progeny must be tetraploid.

The most likely genotypes for the two parents are estimated by `EstimateParentalGenotypes` using `GetLikelyGen`. If parental gentoypes don't match progeny allele frequencies, the function attempts to correct the parental genotypes to the most likely combination that matches the allele frequency.

For each ploidy being examined, F1 genotype probabilities are then calculated by `AddGenotypePriorProb_Mapping2Parent` Genotype probabilities are updated for each backcrossing generation, then each intermating generation, then each selfing generation.

The default, with `n.gen.backcrossing = 0`, `n.gen.intermating = 0` and `n.gen.selfing = 0`, will simulate an F1 population. A BC1F2 population, for example, would have `n.gen.backcrossing = 1`, `n.gen.intermating = 0` and `n.gen.selfing = 1`. A typical F2 population would have `n.gen.selfing = 1` and the other two parameters set to zero. However, in a self-incompatible species where many F1 are intermated to produce the F2, one would instead use `n.gen.intermating = 1` and set the other parameters to zero.

Value

A `"RADdata"` object identical to that passed to the function, but with data stored in three new slots:

`priorProb` A two-dimensional list of matrices, with rows corresponding to `object$possiblePloidies` and columns corresponding to unique values in `object$taxaPloidy`. Each item in the list is a matrix. For each matrix, allele copy number (from zero to the total ploidy) is in rows, and alleles are in columns. Each value is the probability of sampling an individual with that allele copy number from the population. For any taxa ploidies other than the progeny ploidy, even priors are returned.

`likelyGeno_donor`
: A matrix of the donor parent genotypes that were used for estimating genotype prior probabilities. Formatted like the output of `GetLikelyGen`.

`likelyGeno_recurrent`
: A matrix of the recurrent parent genotypes that were use for estimating gentoype prior probabilities.

Note

For the time being, in allopolyploids it is assumed that copies of an allele are distributed among as few isoloci as possible. For example, if an autotetraploid genotype had two copies of allele A and two copies of allele B, it is assumed to be AA BB rather than AB AB. This may be remedied in the future by examining distribution of genotype likelihoods.

Author(s)

Lindsay V. Clark

See Also

`AddGenotypeLikelihood`, `AddGenotypePriorProb_HWE`

Examples

```
# load dataset and set some parameters
data(exampleRAD_mapping)
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")
exampleRAD_mapping <- AddAlleleFreqMapping(exampleRAD_mapping,
                                           expectedFreqs = c(0.25, 0.75),
                                           allowedDeviation = 0.08)
exampleRAD_mapping <- AddGenotypeLikelihood(exampleRAD_mapping)

# examine the dataset
exampleRAD_mapping
exampleRAD_mapping$alleleFreq

# estimate genotype priors for a BC1 population
exampleRAD_mapping <- AddGenotypePriorProb_Mapping2Parents(exampleRAD_mapping,
                                                      n.gen.backcrossing = 1)
exampleRAD_mapping$priorProb
```

`AddPCA` *Perform Principal Components Analysis on “RADdata” Object*

Description

This function uses read depth ratios or posterior genotype probabilities (the latter preferentially) as input data for principal components analysis. The PCA scores are then stored in the `$PCA` slot of the `"RADdata"` object.

Usage

```
AddPCA(object, ...)
## S3 method for class 'RADdata'
AddPCA(object, nPcsInit = 10, maxR2changeratio = 0.05,
                          minPcsOut = 1, ...)
```

Arguments

`object` A "`RADdata`" object.

`nPcsInit` The number of principal component axes to initially calculate.

`maxR2changeratio`
This number determines how many principal component axes are retained. The difference in R^2 values between the first and second axes is multiplied by `maxR2changeratio`. The last axis retained is the first axis after which the R^2 value changes by less than this value. Lower values of `maxR2changeratio` will result in more axes being retained.

`minPcsOut` The minimum number of PC axes to output, which can override `maxR2changeratio`.

`...` Additional arguments to be passed to the `pca` function from the **pcaMethods** BioConductor package.

Details

The PPCA (probabalistic PCA) method from **pcaMethods** is used, due to the high missing data rate that is typical of genotyping-by-sequencing datasets.

Value

A "`RADdata`" object identical to the one passed to the function, but with a matrix added to the `$PCA` slot. This matrix contains PCA scores, with taxa in rows, and PC axes in columns.

Note

If you see the error

`Error in if (rel_ch < threshold & count > 5) { : missing value where TRUE/FALSE needed`

try lowering `nPcsInit`.

Author(s)

Lindsay V. Clark

See Also

`AddAlleleFreqByTaxa`

Examples

```
# load data
data(exampleRAD)
# do PCA
exampleRAD <- AddPCA(exampleRAD, nPcsInit = 3)

plot(exampleRAD$PCA[,1], exampleRAD$PCA[,2])
```

AddPloidyChiSq *Chi-Square Test on Genotype Likelihood Distributions*

Description

This function is intended to help identify the correct inheritance mode for each locus in a "RADdata" object. Expected genotype frequencies are taken from object$priorProb. Observed genotype frequencies are estimated from object$genotypeLikelihood, where each taxon has a partial assignment to each genotype, proportional to genotype likelihoods. A χ^2 statistic is then estimated.

Usage

```
AddPloidyChiSq(object, ...)
## S3 method for class 'RADdata'
AddPloidyChiSq(object, excludeTaxa = GetBlankTaxa(object),
               ...)
```

Arguments

object	A "RADdata" object. Genotype prior probabilities and likelihoods should have been added.
excludeTaxa	A character vector indicating names of taxa to exclude from calculations.
...	Additional arguments to be passed to other methods (none currently in use).

Details

Parents (in mapping populations) and blank taxa are automatically excluded from calculations.

Genotypes with zero prior probability would result in an infinite A χ^2 statistic and therefore are excluded from the calculation. However, the total number of observations (total number of taxa) remains the same, so that if there are many taxa with high likelihood for a genotype with zero prior probability, χ^2 will be high.

Value

A "RADdata" object identical to the one passed to the function, but with a matrix added to the $ploidyChiSq slot. This matrix has inheritance rows (matching object$priorProb) in rows and alleles in columns. object$ploidyChiSq contains the χ^2 values.

Author(s)

Lindsay V. Clark

See Also

AddGenotypeLikelihood, AddPloidyLikelihood

Examples

```
# load dataset and set some parameters
data(exampleRAD_mapping)
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")
exampleRAD_mapping <- AddAlleleFreqMapping(exampleRAD_mapping,
                                           expectedFreqs = c(0.25, 0.75),
                                           allowedDeviation = 0.08)
exampleRAD_mapping <- AddGenotypeLikelihood(exampleRAD_mapping)
exampleRAD_mapping <- AddGenotypePriorProb_Mapping2Parents(exampleRAD_mapping,
                                                      n.gen.backcrossing = 1)

# get chi-squared values
exampleRAD_mapping <- AddPloidyChiSq(exampleRAD_mapping)
# view chi-squared and p-values (diploid only)
exampleRAD_mapping$ploidyChiSq
```

AddPloidyLikelihood *Likelihoods for Possible Ploidies Based on Genotype Distributions*

Description

Given prior genotype probabilities, and a set of high-confidence genotypes estimated with GetLikelyGen, this function estimates the probability of observing that distribution of genotypes and stores the probability in the $ploidyLikelihood slot of the "RADdata" object.

Usage

```
AddPloidyLikelihood(object, ...)
## S3 method for class 'RADdata'
AddPloidyLikelihood(object, excludeTaxa = GetBlankTaxa(object),
                              minLikelihoodRatio = 50, ...)
```

Arguments

object	A "RADdata" object. Prior genotype probabilities and genotype likelihoods should have already been added using the appropriate functions.
...	Additional arguments to be passed to the method for "RADdata".
excludeTaxa	A character vector indicating taxa that should be excluded from calculations.
minLikelihoodRatio	A number, one or higher, to be passed to GetLikelyGen.

Details

The purpose of this function is to estimate the correct inheritance mode for each locus. This function may be deleted in the future in favor of better alternatives.

Value

A "RADdata" object identical to that passed to the function, but with results added to the $ploidyLikelihood slot. This has one row for each possible ploidy (each ploidy with data in $priorProb), and one column for each allele. Each element of the matrix is the multinomial probability of seeing that distribution of genotypes given the prior probabilities.

Author(s)

Lindsay V. Clark

See Also

AddPloidyChiSq

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##-- or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (object, ...)
{
    UseMethod("AddPloidyLikelihood", object)
  }
```

CanDoGetWeightedMeanGeno

Check Whether GetWeightedMeanGenotypes *Can Be Run*

Description

This function is used internally by AddPCA, AddAlleleFreqByTaxa, and the internal function .alleleFreq to test whether GetWeightedMeanGenotypes can be run on a "RADdata" object.

Usage

```
CanDoGetWeightedMeanGeno(object, ...)
```

Arguments

object	A "RADdata" object.
...	Additional arguments (none implemented).

Value

A single Boolean value. To be TRUE, object$posteriorProb must be non-null, and either there must be only one possible ploidy, or object$ploidyChiSq must be non-null.

Author(s)

Lindsay V. Clark

See Also

AddGenotypePosteriorProb, AddPloidyChiSq

Examples

```
data(exampleRAD)

CanDoGetWeightedMeanGeno(exampleRAD)

exampleRAD <- AddAlleleFreqHWE(exampleRAD)
exampleRAD <- AddGenotypePriorProb_HWE(exampleRAD)
exampleRAD <- AddGenotypeLikelihood(exampleRAD)
exampleRAD <- AddPloidyChiSq(exampleRAD)
exampleRAD <- AddGenotypePosteriorProb(exampleRAD)

CanDoGetWeightedMeanGeno(exampleRAD)
```

EstimateContaminationRate *Estimate Sample Contamination Using Blanks*

Description

Based on mean read depth at blank and non-blank taxa, estimate sample cross-contamination and add that information to the "RADdata" object.

Usage

```
EstimateContaminationRate(object, ...)
## S3 method for class 'RADdata'
EstimateContaminationRate(object, multiplier = 1, ...)
```

Arguments

object	A "RADdata" object where SetBlankTaxa has already been used to assign one or more taxa as blanks.
multiplier	A single numeric value, or a named numeric vector with one value per blank taxon in object, with names matching the blank taxa names. Read depth at blank taxa will be multiplied by this number when estimating sample cross-contamination. See example below.

`...`	Additional arguments (none implemented).

Details

This function estimates sample cross-contamination assuming that the only source of contamination is from adapter or sample spill-over between wells during library preparation, or contamination among the libraries themselves. If you anticipate a higher rate of contamination during DNA extraction before library preparation, you may wish to increase the value using `SetContamRate`.

It is important to set the contamination rate to a reasonably accurate value (*i.e.* the right order of magnitude) in order for **polyRAD** to be able to identify homozygotes that may otherwise appear heterozygous due to contamination.

Value

A `"RADdata"` object identical to `object` but with the `"contamRate"` attribute adjusted.

Author(s)

Lindsay V. Clark

Examples

```
# dataset for this example
data(Msi01genes)

# give the name of the taxon that is blank
Msi01genes <- SetBlankTaxa(Msi01genes, "blank")

# Fifteen libraries were done; blank is pooled over all of them, and
# most other samples are pooled over two libraries.
mymult <- 2/15

# estimate the contamination rate
Msi01genes <- EstimateContaminationRate(Msi01genes, multiplier = mymult)
```

`ExamineGenotype` *Plots to Examine Genotype Calling at a Single Taxon and Allele*

Description

For a given taxon and allele, this function generates barplots showing read depth ratio, posterior mean genotype, genotype prior probabilities, genotype likelihoods, and genotype posterior probabilities. It is intended as a sanity check on genotype calling, as well as a means to visually demonstrate the concept of Bayesian genotype calling.

Usage

```
ExamineGenotype(object, ...)

## S3 method for class 'RADdata'
ExamineGenotype(object, taxon, allele, pldindex = 1, ...)
```

Arguments

`object`	A `RADdata` object for which genotype calling has already been performed.
`taxon`	A single character string indicating the taxon to show.
`allele`	A single character string indicating the allele to show.
`pldindex`	An index of which inheritance mode to use within `object$possiblePloidies`.
`...`	Other arguments (none implemented).

Value

A barplot is generated. Invisibly, a list is returned:

`alleleDepth`	Sequence read depth for the selected allele.
`antiAlleleDepth`	Sequence read depth for all other alleles at the locus.
`depthRatio`	Proportion of reads at this taxon and locus belonging to this allele.
`priorProb`	A vector of genotype prior probabilities.
`genotypeLikelhood`	A vector of genotype likelihoods.
`posteriorProb`	A vector of genotype posterior probabilities.
`postMean`	The posterior mean genotype on a scale of 0 to 1.

Author(s)

Lindsay V. Clark

Examples

```
data(exampleRAD)

exampleRAD <- IterateHWE(exampleRAD)

eg <- ExamineGenotype(exampleRAD, "sample088", "loc1_T")
```

exampleRAD *Miniature Datasets for Testing polyRAD Functions*

Description

exampleRAD and exampleRAD_mapping are two very small simulated "RADdata" datasets for testing polyRAD functions. Each has four loci. exampleRAD is a natural population of 100 individuals with a mix of diploid and tetraploid loci, with 80 individuals diploid and 20 individuals triploid. exampleRAD_mapping is a diploid BC1 mapping population with two parents and 100 progeny. Msi01genes is a "RADdata" object with 585 taxa and 24 loci, containing real data from *Miscanthus sinensis*, obtained by using VCF2RADdata on the file Msi01genes.vcf. Most individuals in Msi01genes are diploid, with three haploids and one triploid.

Usage

```
data(exampleRAD)
data(exampleRAD_mapping)
data(Msi01genes)
```

Format

See the format described in "RADdata".

Source

Randomly generated using a script available in polyRAD/extdata/simulate_rad_data.R.

M. sinensis sequencing data available at https://www.ncbi.nlm.nih.gov//bioproject/PRJNA207721, with full genotype calls at doi:10.13012/B2IDB1402948_V1.

Examples

```
data(exampleRAD)
exampleRAD
data(exampleRAD_mapping)
exampleRAD_mapping
data(Msi01genes)
Msi01genes
```

ExpectedHindHe *Simulate Data to Get Expected Distribution of Hind/He*

Description

These functions were created to help users determine an appropriate cutoff for filtering loci based on H_{ind}/H_E after running HindHe and InbreedingFromHindHe. ExpectedHindHe takes allele frequencies, sample size, and read depths from a RADdata object, simulates genotypes and allelic read depths from these assuming Mendelian inheritance, and then estimates H_{ind}/H_E for each simulated locus. ExpectedHindHeMapping performs similar simulation and estimation, but in mapping populations based on parental genotypes and expected distribution of progeny genotypes. SimGenotypes, SimGenotypesMapping, and SimAlleleDepth are internal functions used by ExpectedHindHe and ExpectedHindHeMapping but are provided at the user level since they may be more broadly useful.

Usage

```
ExpectedHindHe(object, ploidy = object$possiblePloidies[[1]], inbreeding = 0,
               overdispersion = 20, contamRate = 0, errorRate = 0.001,
               reps = ceiling(5000/nLoci(object)),
               quiet = FALSE, plot = TRUE)

ExpectedHindHeMapping(object, ploidy = object$possiblePloidies[[1]],
                      n.gen.backcrossing = 0, n.gen.selfing = 0,
                      overdispersion = 20, contamRate = 0, errorRate = 0.001,
                      freqAllowedDeviation = 0.05,
                    minLikelihoodRatio = 10, reps = ceiling(5000/nLoci(object)),
                      quiet = FALSE, plot = TRUE)

SimGenotypes(alleleFreq, alleles2loc, nsam, inbreeding, ploidy)

SimGenotypesMapping(donorGen, recurGen, alleles2loc, nsam,
                    ploidy.don, ploidy.rec,
                    n.gen.backcrossing, n.gen.selfing)

SimAlleleDepth(locDepth, genotypes, alleles2loc, overdispersion = 20,
               contamRate = 0, errorRate = 0.001)
```

Arguments

object — A RADdata object.

ploidy — A single integer indicating the ploidy to use for genotype simulation. For ExpectedHindHe and ExpectedHindHeMapping, this number will be multiplied by the values in GetTaxaPloidy(object) then divided by two to determine the ploidy of each individual for simulation.

`inbreeding` A number ranging from 0 to 1 indicating the amount of inbreeding (F). This represents inbreeding from all sources (population structure, self-fertilization, etc.) and can be estimated with `InbreedingFromHindHe`.

`overdispersion` Overdispersion parameter as described in `AddGenotypeLikelihood`. Lower values will cause allelic read depth distributions to deviate further from expectations based on allele copy number.

`contamRate` Sample cross-contamination rate to simulate. Although 0 is the default, 0.001 is also reasonable.

`errorRate` Sequencing error rate to simulate. For Illumina reads, 0.001 is a reasonable value. An error is assumed to have an equal chance of converting an allele to any other allele at the locus, although this is somewhat of an oversimplification.

`reps` The number of times to simulate the data and estimate H_{ind}/H_E. This can generally be left at the default, but set it higher than 1 if you want to see within-locus variance in the estimate.

`quiet` Boolean indicating whether to suppress messages and results printed to console.

`plot` Boolean indicating whether to plot a histogram of H_{ind}/H_E values.

`n.gen.backcrossing`
An integer indicating the number of generations of backcrossing.

`n.gen.selfing` An integer indicating the number of generations of self-fertilization.

`freqAllowedDeviation`
The amount by which allele frequencies are allowed to deviate from expected allele frequencies. See `AddAlleleFreqMapping`.

`minLikelihoodRatio`
Minimum likelihood ratio for determining the most likely parental genotypes. See `GetLikelyGen`.

`alleleFreq` A vector of allele frequencies, as can be found in the `$alleleFreq` slot of a `RADdata` object after running `AddAlleleFreqHWE`.

`alleles2loc` An integer vector assigning alleles to loci, as can be found in the `$alleles2loc` slot of a `RADdata` object.

`nsam` An integer indicating the number of samples (number of taxa) to simulate.

`donorGen` A vector indicating genotypes of the donor parent (which can be either parent if backcrossing was not performed), with one value for each allele in the dataset, and numbers indicating the copy number of each allele.

`recurGen` A vector indicating genotypes of the recurrent parent, as with `donorGen`.

`ploidy.don` A single integer indicating the ploidy of the donor parent.

`ploidy.rec` A single integer indicating the ploidy of the recurrent parent.

`locDepth` An integer matrix indicating read depth at each taxon and locus. Formatted as the `$locDepth` slot of a `RADdata` object, notably with columns named by locus number rather than locus name.

`genotypes` A numeric matrix, formatted as the output of `GetProbableGenotypes` or `SimGenotypes`, indicating genotypes as allele copy number.

Details

To prevent highly inflated values in the output, ExpectedHindHe filters loci with minor allele frequencies below five times the sequencing error rate.

Value

ExpectedHindHe and ExpectedHindHeMapping invisibly return a matrix, with loci in rows and reps in columns, containing H_{ind}/H_E from the simulated loci.

SimGenotypes and SimGenotypesMapping return a numeric matrix of allele copy number, with samples in rows and alleles in columns, similar to that produced by GetProbableGenotypes.

SimAlleleDepth returns an integer matrix of allelic read depth, with samples in rows and alleles in columns, similar to the $alleleDepth slot of a RADdata object.

Author(s)

Lindsay V. Clark

References

Clark, L. V., Mays, W., Lipka, A. E. and Sacks, E. J. (2022) A population-level statistic for assessing Mendelian behavior of genotyping-by-sequencing data from highly duplicated genomes. *BMC Bioinformatics* **23**, 101, doi:10.1186/s12859-022-04635-9.

Examples

```
# Load dataset for the example
data(exampleRAD)
exampleRAD <- AddAlleleFreqHWE(exampleRAD)

# Simulate genotypes
simgeno <- SimGenotypes(exampleRAD$alleleFreq, exampleRAD$alleles2loc, 10, 0.2, 2)

# Simulate reads
simreads <- SimAlleleDepth(exampleRAD$locDepth[1:10,], simgeno, exampleRAD$alleles2loc)

# Get expected Hind/He distribution if all loci in exampleRAD were well-behaved
ExpectedHindHe(exampleRAD, reps = 10)

# Mapping population example
data(exampleRAD_mapping)
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")
exampleRAD_mapping <- AddAlleleFreqMapping(exampleRAD_mapping,
                                           expectedFreqs = c(0.25, 0.75),
                                           allowedDeviation = 0.08)
exampleRAD_mapping <- AddGenotypeLikelihood(exampleRAD_mapping)
exampleRAD_mapping <- EstimateParentalGenotypes(exampleRAD_mapping,
                                                n.gen.backcrossing = 1)

simgenomap <- SimGenotypesMapping(exampleRAD_mapping$likelyGeno_donor[1,],
                                  exampleRAD_mapping$likelyGeno_recurrent[1,],
                                  exampleRAD_mapping$alleles2loc,
                                  nsam = 10, ploidy.don = 2, ploidy.rec = 2,
                                  n.gen.backcrossing = 1,
                                  n.gen.selfing = 0)

ExpectedHindHeMapping(exampleRAD_mapping, n.gen.backcrossing = 1, reps = 10)
```

ExportGAPIT *Export RADdata Object for Use by Other R Packages*

Description

After a "RADdata" object has been run through a pipeline such as IteratePopStruct, these functions can be used to export the genotypes to R packages and other software that can perform genome-wide association and genomic prediction. ExportGAPIT, Export_rrBLUP_Amat, Export_rrBLUP_GWAS, Export_GWASpoly, and Export_TASSEL_Numeric all export continuous numerical genotypes generated by GetWeightedMeanGenotypes. Export_polymapR, Export_Structure, and Export_adegenet_genind use GetProbableGenotypes to export discrete genotypes. Export_MAPpoly and Export_polymapR_probs export genotype posterior probabilities.

Usage

```
ExportGAPIT(object, onePloidyPerAllele = FALSE)

Export_rrBLUP_Amat(object, naIfZeroReads = FALSE,
                   onePloidyPerAllele = FALSE)

Export_rrBLUP_GWAS(object, naIfZeroReads = FALSE,
                   onePloidyPerAllele = FALSE)

Export_TASSEL_Numeric(object, file, naIfZeroReads = FALSE,
                      onePloidyPerAllele = FALSE)

Export_polymapR(object, naIfZeroReads = TRUE,
                progeny = GetTaxa(object)[!GetTaxa(object) %in%
                  c(GetDonorParent(object), GetRecurrentParent(object),
                    GetBlankTaxa(object))])

Export_polymapR_probs(object, maxPcutoff = 0.9,
                      correctParentalGenos = TRUE,
                      multiallelic = "correct")

Export_MAPpoly(object, file, pheno = NULL, ploidyIndex = 1,
               progeny = GetTaxa(object)[!GetTaxa(object) %in%
                 c(GetDonorParent(object), GetRecurrentParent(object),
                   GetBlankTaxa(object))],
                digits = 3)

Export_GWASpoly(object, file, naIfZeroReads = TRUE, postmean = TRUE, digits = 3,
                splitByPloidy = TRUE)

Export_Structure(object, file, includeDistances = FALSE, extraCols = NULL,
                 missingIfZeroReads = TRUE)

Export_adegenet_genind(object, ploidyIndex = 1)
```

Arguments

`object` A `"RADdata"` object with posterior genotype probabilities already estimated.

`onePloidyPerAllele` Logical. If `TRUE`, for each allele the inheritance mode with the lowest χ^2 value is selected and is assumed to be the true inheritance mode. If `FALSE`, inheritance modes are weighted by inverse χ^2 values for each allele, and mean genotypes that have been weighted across inheritance modes are returned.

`naIfZeroReads` A logical indicating whether `NA` should be inserted into the output matrix for any taxa and loci where the total read depth for the locus is zero. If `FALSE`, the output for these genotypes is essentially the mode (for `Export_polymapR` and `Export_GWASpoly`) or mean (for others) across prior genotype probabilities, since prior and posterior genotype probabilities are equal when there are no reads.

`file` A character string indicating a file path to which to write.

`pheno` A data frame or matrix of phenotypic values, with progeny in rows and traits in columns. Columns should be named.

`ploidyIndex` Index, within `object$possiblePloidies`, of the ploidy to be exported.

`progeny` A character vector indicating which individuals to export as progeny of the cross.

`maxPcutoff` A cutoff for posterior probabilities, below which genotypes will be reported as 'NA' in the 'geno' column.

`correctParentalGenos` Passed to `GetProbableGenotypes`. If `TRUE`, parental genotypes are corrected based on progeny allele frequencies.

`multiallelic` Passed to `GetProbableGenotypes`. Under the default, genotypes at multiallelic loci will be corrected to sum to the ploidy.

`digits` Number of decimal places to which to round genotype probabilities or posterior mean genotypes in the output file.

`postmean` Logical. If `TRUE`, posterior mean genotypes will be output. If `FALSE`, discrete genotypes will be output.

`splitByPloidy` Logical. If `TRUE` and there are multiple `taxaPloidy` values in the dataset, multiple files are written, one per ploidy.

`includeDistances` Logical. If `TRUE`, the second row of the Structure file will contain distances between markers, which can be used by the linkage model in Structure.

extraCols	An optional data frame, with one row per taxon, containing columns of data to output to the left of the genotypes in the Structure file.
missingIfZeroReads	See naIfZeroReads. If TRUE, a value of -9 will be output for any genotypes with zero reads, indicating that those genotypes are missing.

Details

GAPIT, **FarmCPU**, **rrBLUP**, **TASSEL**, and **GWASpoly** allow genotypes to be a continuous numeric variable. **MAPpoly** and **polymapR** allow for import of genotype probabilities. **GAPIT** does not allow missing data, hence there is no naIfZeroReads argument for ExportGAPIT. Genotypes are exported on a scale of -1 to 1 for **rrBLUP**, on a scale of 0 to 2 for **GAPIT** and **FarmCPU**, and on a scale of 0 to 1 for **TASSEL**.

For all functions except Export_Structure and Export_adegenet_genind, one allele per marker is dropped. Export_MAPpoly also drops alleles where one or both parental genotypes could not be determined, and where both parents are homozygotes.

For ExportGAPIT and Export_rrBLUP_GWAS, chromosome and position are filled with dummy data if they do not exist in object$locTable. For Export_TASSEL_Numeric, allele names are exported, but no chromosome or position information per se.

If the chromosomes in object$locTable are in character format, ExportGAPIT, Export_MAPpoly, and Export_GWASpoly will attempt to extract chromosome numbers.

For **polymapR** there must only be one possible inheritance mode across loci (one value in object$possiblePloidies) in the RADdata object, although triploid F1 populations derived from diploid and tetraploid parents are allowed. See SubsetByPloidy for help reducing a RADdata object to a single inheritance mode.

MAPpoly only allows one ploidy, but Export_MAPpoly allows the user to select which inheritance mode from the RADdata object to use. (This is due to how internal **polyRAD** functions are coded.)

Value

For ExportGAPIT, a list:

GD	A data frame with taxa in the first column and alleles (markers) in subsequent columns, containing the genotypes. To be passed to the GD argument for GAPIT or FarmCPU.
GM	A data frame with the name, chromosome number, and position of every allele (marker). To be passed to the GM argument for GAPIT or FarmCPU.

For Export_rrBLUP_Amat, a matrix with taxa in rows and alleles (markers) in columns, containing genotype data. This can be passed to A.mat in **rrBLUP**.

For Export_rrBLUP_GWAS, a data frame with alleles (markers) in rows. The first three columns contain the marker names, chromosomes, and positions, and the remaining columns each represent one taxon and contain the genotype data. This can be passed to the GWAS function in **rrBLUP**.

Export_TASSEL_Numeric and Export_MAPpoly write a file but does not return an object.

For Export_polymapR, a matrix of integers indicating the most probable allele copy number, with markers in rows and individuals in columns. The parents are listed first, followed by all progeny.

For Export_polymapR_probs, a data frame suitable to pass to the probgeno_df argument of checkF1. Note that under default parameters, in some cases the maxP, maxgeno, and geno columns may not actually reflect the maximum posterior probability if genotype correction was performed.

For Export_adegenet_genind, a "genind" object.

Export_MAPpoly, Export_GWASpoly, and Export_Structure write files but do not return an object. Files output by Export_GWASpoly are comma delimited and in numeric format. Sample and locus names are included in the file output by Export_Structure, and the number of rows for each sample is equal to the highest ploidy as determined by the taxaPloidy slot and the output of GetProbableGenotypes.

Note

rrBLUP and **polymapR** are available through CRAN, and **GAPIT** and **FarmCPU** must be downloaded from the Zhang lab website. **MAPpoly** is available on GitHub but not yet on CRAN. **GWASpoly** is available from GitHub.

In my experience with **TASSEL 5**, numerical genotype files that are too large do not load/display properly. If you run into this problem I recommend using SplitByChromosome to split your RADdata object into multiple smaller objects, which can then be exported to separate files using Export_TASSEL_Numeric. If performing GWAS, you may also need to compute a kinship matrix using separate software such as **rrBLUP**.

Author(s)

Lindsay V. Clark

References

GAPIT and FarmCPU:

https://zzlab.net/GAPIT/

Lipka, A. E., Tian, F., Wang, Q., Peiffer, J., Li, M., Bradbury, P. J., Gore, M. A., Buckler, E. S. and Zhang, Z. (2012) GAPIT: genome association and prediction integrated tool. *Bioinformatics* **28**, 2397–2399.

https://zzlab.net/FarmCPU/

Liu, X., Huang, M., Fan, B., Buckler, E. S., Zhang, Z. (2016) Iterative usage of fixed and random effects models for powerful and efficient genome-wide association studies. *PLoS Genetics* **12**, e1005767.

rrBLUP:

Endelman, J.B. (2011) Ridge Regression and Other Kernels for Genomic Selection with R Package rrBLUP. *The Plant Genome* **4**, 250–255.

TASSEL:

https://www.maizegenetics.net/tassel

Bradbury, P. J., Zhang, Z., Kroon, D. E., Casstevens, T. M., Ramdoss, Y. and Buckler, E. S. (2007) TASSEL: Software for association mapping of complex traits in diverse samples. *Bioinformatics* **23**, 2633–2635.

polymapR:

Bourke, P., van Geest, G., Voorrips, R. E., Jansen, J., Kranenberg, T., Shahin, A., Visser, R. G. F., Arens, P., Smulders, M. J. M. and Maliepaard, C. (2018) polymapR: linkage analysis and genetic map construction from F1 populations of outcrossing polyploids. *Bioinformatics* **34**, 3496–3502.

MAPpoly:

https://github.com/mmollina/MAPpoly

Mollinari, M. and Garcia, A. A. F. (2018) Linkage analysis and haplotype phasing in experimental autopolyploid populations with high ploidy level using hidden Markov models. *bioRxiv* doi: https://doi.org/10.1101/415232.

GWASpoly:

https://github.com/jendelman/GWASpoly

Rosyara, U. R., De Jong, W. S., Douches, D. S., and Endelman, J. B. (2016) Software for Genome-Wide Association Studies in Autopolyploids and Its Application to Potato. *Plant Genome* **9**.

Structure:

https://web.stanford.edu/group/pritchardlab/structure.html

Hubisz, M. J., Falush, D., Stephens, M. and Pritchard, J. K. (2009) Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources* **9**, 1322–1332.

Falush, D., Stephens, M. and Pritchard, J. K. (2007) Inferences of population structure using multilocus genotype data: dominant markers and null alleles. *Molecular Ecology Notes* **7**, 574–578

Falush, D., Stephens, M. and Pritchard, J. K. (2003) Inferences of population structure using multilocus genotype data: linked loci and correlated allele frequencies. *Genetics* **164**, 1567–1587.

Pritchard, J. K., Stephens, M. and Donnelly, P. (2000) Inference of population structure using multilocus genotype data. *Genetics* **155**, 945–959.

See Also

GetWeightedMeanGenotypes, RADdata2VCF

Examples

```
# load example dataset
data(exampleRAD)
# get genotype posterior probabilities
exampleRAD <- IterateHWE(exampleRAD)

# export to GAPIT
exampleGAPIT <- ExportGAPIT(exampleRAD)

# export to rrBLUP
example_rrBLUP_A <- Export_rrBLUP_Amat(exampleRAD)
example_rrBLUP_GWAS <- Export_rrBLUP_GWAS(exampleRAD)

# export to TASSEL
outfile <- tempfile() # temporary file for example
Export_TASSEL_Numeric(exampleRAD, outfile)
```

```
# for mapping populations
data(exampleRAD_mapping)

# specify donor and recurrent parents
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")

# run the pipeline
exampleRAD_mapping <- PipelineMapping2Parents(exampleRAD_mapping)

# convert to polymapR format
examplePMR <- Export_polymapR(exampleRAD_mapping)

examplePMR2 <- Export_polymapR_probs(exampleRAD_mapping)

# export to MAPpoly
outfile2 <- tempfile() # temporary file for example
# generate a dummy phenotype matrix containing random numbers
mypheno <- matrix(rnorm(200), nrow = 100, ncol = 2,
                  dimnames = list(GetTaxa(exampleRAD_mapping)[-(1:2)],
                                  c("Height", "Yield")))
Export_MAPpoly(exampleRAD_mapping, file = outfile2, pheno = mypheno)

# load data into MAPpoly
# require(mappoly)
# mydata <- read_geno_prob(outfile2)

# export to GWASpoly
outfile3 <- tempfile() # temporary file for example
Export_GWASpoly(SubsetByPloidy(exampleRAD, list(2)), outfile3)

# export to Structure
outfile4 <- tempfile() # temporary file for example
Export_Structure(exampleRAD, outfile4)

# export to adegenet
if(requireNamespace("adegenet", quietly = TRUE)){
  mygenind <- Export_adegenet_genind(exampleRAD)
}
```

GetLikelyGen *Output the Most Likely Genotype*

Description

For a single taxon in a "RADdata" object, GetLikelyGen returns the most likely genotype (expressed in allele copy number) for each allele and each possible ploidy. The likelihoods used for determining genotypes are those stored in object$genotypeLikelihood.

Usage

```
GetLikelyGen(object, taxon, minLikelihoodRatio = 10)
```

Arguments

`object` A "RADdata" object.

`taxon` A character string indicating the taxon for which genotypes should be returned.

`minLikelihoodRatio` A number indicating the minimum ratio of the likelihood of the most likely genotype to the likelihood of the second-most likely genotype for any genotype to be output for a given allele. If this number is one or less, all of the most likely genotypes will be output regardless of likelihood ratio. Where filtering is required so that only high confidence genotypes are retained, this number should be increased.

Value

A matrix with ploidies in rows (named with ploidies converted to character format) and alleles in columns. Each value indicates the most likely number of copies of that allele that the taxon has, assuming that ploidy.

Author(s)

Lindsay V. Clark

See Also

AddGenotypeLikelihood

Examples

```
# load dataset for this example
data(exampleRAD)
# add allele frequencies and genotype likelihoods
exampleRAD <- AddAlleleFreqHWE(exampleRAD)
exampleRAD <- AddGenotypeLikelihood(exampleRAD)

# get most likely genotypes
GetLikelyGen(exampleRAD, "sample001")
GetLikelyGen(exampleRAD, "sample082")

# try different filtering
GetLikelyGen(exampleRAD, "sample001", minLikelihoodRatio = 1)
GetLikelyGen(exampleRAD, "sample001", minLikelihoodRatio = 100)
```

GetWeightedMeanGenotypes

Export Numeric Genotype Values from Posterior Probabilities

Description

These functions calculate numerical genotype values using posterior probabilities in a "RADdata" object, and output those values as a matrix of taxa by alleles. GetWeightedMeanGenotypes returns continuous genotype values, weighted by posterior genotype probabilities (*i.e.* posterior mean genotypes). GetProbableGenotypes returns discrete genotype values indicating the most probable genotype. If the "RADdata" object includes more than one possible inheritance mode, the $ploidyChiSq slot is used for selecting or weighting inheritance modes for each allele.

Usage

```
GetWeightedMeanGenotypes(object, ...)
## S3 method for class 'RADdata'
GetWeightedMeanGenotypes(object, minval = 0, maxval = 1,
                         omit1allelePerLocus = TRUE,
                         omitCommonAllele = TRUE,
                         naIfZeroReads = FALSE,
                         onePloidyPerAllele = FALSE, ...)

GetProbableGenotypes(object, ...)
## S3 method for class 'RADdata'
GetProbableGenotypes(object, omit1allelePerLocus = TRUE,
                     omitCommonAllele = TRUE,
                     naIfZeroReads = FALSE,
                     correctParentalGenos = TRUE,
                     multiallelic = "correct", ...)
```

Arguments

object
: A "RADdata" object. Posterior genotype probabilities should have been added with AddGenotypePosteriorProb, and if there is more than one possible ploidy, ploidy chi-squared values should have been added with AddPloidyChiSq.

...
: Additional arguments, listed below, to be passed to the method for "RADdata".

minval
: The number that should be used for indicating that a taxon has zero copies of an allele.

maxval
: The number that should be used for indicating that a taxon has the maximum copies of an allele (equal to the ploidy of the locus).

omit1allelePerLocus
: A logical indicating whether one allele per locus should be omitted from the output, in order to reduce the number of variables and prevent singularities for genome-wide association and genomic prediction. The value for one allele can be predicted from the values from all other alleles at its locus.

`omitCommonAllele`
A logical, passed to the `commonAllele` argument of `OneAllelePerMarker`, indicating whether the most common allele for each locus should be omitted (as opposed to simply the first allele for each locus). Ignored if `omit1allelePerLocus = FALSE`.

`naIfZeroReads` A logical indicating whether `NA` should be inserted into the output matrix for any taxa and loci where the total read depth for the locus is zero. If `FALSE`, the output for these genotypes is essentially calculated using prior genotype probabilities, since prior and posterior genotype probabilities are equal when there are no reads.

`onePloidyPerAllele`
Logical. If `TRUE`, for each allele the inheritance mode with the lowest χ^2 value is selected and is assumed to be the true inheritance mode. If `FALSE`, inheritance modes are weighted by inverse χ^2 values for each allele, and mean genotypes that have been weighted across inheritance modes are returned.

`correctParentalGenos`
Logical. If `TRUE` and if the dataset was processed with `PipelineMapping2Parents`, the parental genotypes that are output are corrected according to the progeny allele frequencies, using the `likelyGeno_donor` and `likelyGeno_recurrent` slots in `object`. For the ploidy of the marker, the appropriate ploidy for the parents is selected using the `donorPloidies` and `recurrentPloidies` slots.

`multiallelic` A string indicating how to handle cases where allele copy number across all alleles at a locus does not sum to the ploidy. To retain the most probable copy number for each allele, even if they don't sum to the ploidy across all alleles, use `"ignore"`. To be conservative and convert these allele copy numbers to `NA`, use `"na"`. To adjust allele copy numbers to match the ploidy (adding or subtracting allele copies while maximizing the product of posterior probabilities across alleles), use `"correct"`.

Details

For each inheritance mode m, taxon t, allele a, allele copy number i, total ploidy k, and posterior genotype probability $p_{i,t,a,m}$, posterior mean genotype $g_{t,a,m}$ is estimated by `GetWeightedMeanGenotypes` as:

$$g_{t,a,m} = \sum_{i=0}^{k} p_{i,t,a,m} * \frac{i}{k}$$

For `GetProbableGenotypes`, the genotype is the one with the maximum posterior probability:

$$g_{t,a,m} = i | \max_{i=0}^{k} p_{i,t,a,m}$$

When there are multiple inheritance modes and `onePloidyPerAllele = FALSE`, the weighted genotype is estimated by `GetWeightedMeanGenotypes` as:

$$g_{t,a} = \sum_{m} [g_{t,a,m} * \frac{1}{\chi^2_{m,a}} / \sum_{m} \frac{1}{\chi^2_{m,a}}]$$

In GetProbableGenotypes, or GetWeightedMeanGenotypes when there are multiple inheritance modes and onePloidyPerAllele = TRUE, the genotype is simply the one corresponding to the inheritance mode with the minimum χ^2 value:

$$g_{t,a} = g_{t,a,m} | \min_m \chi^2_{m,a}$$

Value

For GetWeightedMeanGenotypes, a named matrix, with taxa in rows and alleles in columns, and values ranging from minval to maxval. These values can be treated as continuous genotypes.

For GetProbableGenotypes, a list:

genotypes — A named integer matrix, with taxa in rows and alleles in columns, and values ranging from zero to the maximum ploidy for each allele. These values can be treated as discrete genotypes.

ploidy_index — A vector with one value per allele. It contains the index of the most likely inheritance mode of that allele in object$priorProbPloidies.

Author(s)

Lindsay V. Clark

Examples

```
# load dataset
data(exampleRAD_mapping)

# run a genotype calling pipeline;
# substitute with any pipeline and parameters
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")
exampleRAD_mapping <- PipelineMapping2Parents(exampleRAD_mapping,
                                 n.gen.backcrossing = 1, useLinkage = FALSE)


# get weighted mean genotypes
wmg <- GetWeightedMeanGenotypes(exampleRAD_mapping)
# examine the results
wmg[1:10,]

# get most probable genotypes
pg <- GetProbableGenotypes(exampleRAD_mapping, naIfZeroReads = TRUE)
# examine the results
pg$genotypes[1:10,]
```

HindHe

Identify Non-Mendelian Loci and Taxa that Deviate from Ploidy Expectations

Description

HindHe and HindHeMapping both generate a matrix of values, with taxa in rows and loci in columns. The mean value of the matrix is expected to be a certain value depending on the ploidy and, in the case of natural populations and diversity panels, the inbreeding coefficient. colMeans of the matrix can be used to filter non-Mendelian loci from the dataset. rowMeans of the matrix can be used to identify taxa that are not the expected ploidy, are interspecific hybrids, or are a mix of multiple samples.

Usage

```
HindHe(object, ...)

## S3 method for class 'RADdata'
HindHe(object, omitTaxa = GetBlankTaxa(object), ...)

HindHeMapping(object, ...)

## S3 method for class 'RADdata'
HindHeMapping(object, n.gen.backcrossing = 0, n.gen.intermating = 0,
              n.gen.selfing = 0, ploidy = object$possiblePloidies[[1]],
              minLikelihoodRatio = 10,
              omitTaxa = c(GetDonorParent(object), GetRecurrentParent(object),
                           GetBlankTaxa(object)), ...)
```

Arguments

object
: A RADdata object. Genotype calling does not need to have been performed yet. If the population is a mapping population, SetDonorParent and SetRecurrentParent should have been run already.

omitTaxa
: A character vector indicating names of taxa not to be included in the output. For HindHe, these taxa will also be omitted from allele frequency estimations.

n.gen.backcrossing
: The number of generations of backcrossing performed in a mapping population.

n.gen.intermating
: The number of generations of intermating performed in a mapping population. Included for consistency with PipelineMapping2Parents, but currently will give an error if set to any value other than zero. If the most recent generation in your mapping population was random mating among all progeny, use HindHe instead of HindHeMapping.

n.gen.selfing
: The number of generations of self-fertilization performed in a mapping population.

`ploidy` A single value indicating the assumed ploidy to test. Currently, only autopolyploid and diploid inheritance modes are supported.

`minLikelihoodRatio`
Used internally by `EstimateParentalGenotypes` as a threshold for certainty of parental genotypes. Decrease this value if too many markers are being discarded from the calculation.

`...` Additional arguments (none implemented).

Details

These functions are especially useful for highly duplicated genomes, in which RAD tag alignments may have been incorrect, resulting in groups of alleles that do not represent true Mendelian loci. The statistic that is calculated is based on the principle that observed heterozygosity will be higher than expected heterozygosity if a "locus" actually represents two or more collapsed paralogs. However, the statistic uses read depth in place of genotypes, eliminating the need to perform genotype calling before filtering.

For a given taxon * locus, H_{ind} is the probability that two sequencing reads, sampled without replacement, are different alleles (RAD tags).

In `HindHe`, H_E is the expected heterozygosity, estimated from allele frequencies by taking the column means of `object$depthRatios`. This is also the estimated probability that if two alleles were sampled at random from the population at a given locus, they would be different alleles.

In `HindHeMapping`, H_E is the average probability that in a random progeny, two alleles sampled without replacement would be different. The number of generations of backcrossing and self-fertilization, along with the ploidy and estimated parental genotypes, are needed to make this calculation. The function essentially simulates the mapping population based on parental genotypes to determine H_E.

The expectation is that

$$H_{ind}/H_E = \frac{ploidy - 1}{ploidy} * (1 - F)$$

in a diversity panel, where F is the inbreeding coefficient, and

$$H_{ind}/H_E = \frac{ploidy - 1}{ploidy}$$

in a mapping population. Loci that have much higher average values likely represent collapsed paralogs that should be removed from the dataset. Taxa with much higher average values may be higher ploidy than expected, interspecific hybrids, or multiple samples mixed together.

Value

A named matrix, with taxa in rows and loci in columns. For `HindHeMapping`, loci are omitted if consistent parental genotypes could not be determined across alleles.

Author(s)

Lindsay V. Clark

References

Clark, L. V., Mays, W., Lipka, A. E. and Sacks, E. J. (2022) A population-level statistic for assessing Mendelian behavior of genotyping-by-sequencing data from highly duplicated genomes. *BMC Bioinformatics* **23**, 101, doi:10.1186/s12859-022-04635-9.

A seminar describing H_{ind}/H_E is available at https://youtu.be/Z2xwLQYc8OA?t=1678.

See Also

InbreedingFromHindHe, ExpectedHindHe

Examples

```
data(exampleRAD)

hhmat <- HindHe(exampleRAD)
colMeans(hhmat, na.rm = TRUE) # near 0.5 for diploid loci, 0.75 for tetraploid loci

data(exampleRAD_mapping)
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")

hhmat2 <- HindHeMapping(exampleRAD_mapping, n.gen.backcrossing = 1)
colMeans(hhmat2, na.rm = TRUE) # near 0.5; all loci diploid
```

InbreedingFromHindHe *Estimate Inbreeding from Hind/He for a Given Ploidy*

Description

After running HindHe and examining the distribution of values across taxa and loci, InbreedingFromHindHe can be used to estimate the inbreeding statistic F from the median or mode value of H_{ind}/H_E. The statistic estimated encompasses inbreeding from all sources, including population structure, self-fertilization, and preferential mating among relatives. It is intended to be used as input to the process_isoloci.py script.

Usage

```
InbreedingFromHindHe(hindhe, ploidy)
```

Arguments

hindhe	A value for H_{ind}/H_E. It should generally range from zero to one.
ploidy	A single integer indicating the ploidy of the population.

Value

A number indicating the inbreeding statistic F. This is calculated as:

$$1 - hindhe * ploidy/(ploidy - 1)$$

Author(s)

Lindsay V. Clark

See Also

HindHe, ExpectedHindHe, readProcessSamMulti, readProcessIsoloci

Examples

```
InbreedingFromHindHe(0.5, 2)
InbreedingFromHindHe(0.4, 2)
InbreedingFromHindHe(0.5, 4)
```

IterateHWE *Iteratively Estimate Population Parameters and Genotypes In a Diversity Panel*

Description

These are wrapper function that iteratively run other **polyRAD** functions until allele frequencies stabilize to within a user-defined threshold. Genotype posterior probabilities can then be exported for downstream analysis.

Usage

```
IterateHWE(object, selfing.rate = 0, tol = 1e-05,
           excludeTaxa = GetBlankTaxa(object),
           overdispersion = 9)

IterateHWE_LD(object, selfing.rate = 0, tol = 1e-05,
              excludeTaxa = GetBlankTaxa(object),
              LDdist = 1e4, minLDcorr = 0.2,
              overdispersion = 9)

IteratePopStruct(object, selfing.rate = 0, tol = 1e-03,
                 excludeTaxa = GetBlankTaxa(object),
                 nPcsInit = 10, minfreq = 0.0001,
                 overdispersion = 9, maxR2changeratio = 0.05)

IteratePopStructLD(object, selfing.rate = 0, tol = 1e-03,
                   excludeTaxa = GetBlankTaxa(object),
                   nPcsInit = 10, minfreq = 0.0001, LDdist = 1e4,
                   minLDcorr = 0.2,
                   overdispersion = 9, maxR2changeratio = 0.05)
```

Arguments

`object` A "RADdata" object.

`selfing.rate` A number ranging from zero to one indicating the frequency of self-fertilization in the species. For individuals with odd ploidy (e.g. triploids), the selfing rate is always treated as zero and a warning is printed if a value above zero is provided.

`tol` A number indicating when the iteration should end. It indicates the maximum mean difference in allele frequencies between iterations that is tolerated. Larger numbers will lead to fewer iterations.

`excludeTaxa` A character vector indicating names of taxa that should be excluded from allele frequency estimates and chi-squared estimates.

`nPcsInit` An integer indicating the number of principal component axes to initially estimate from `object$depthRatio`. Passed to `AddPCA`.

`minfreq` A number indicating the minimum allele frequency allowed. Passed to `AddAlleleFreqByTaxa`.

`LDdist` The distance, in basepairs, within which to search for alleles that may be in linkage disequilibrium with a given allele.

`minLDcorr` The minimum correlation coefficient between two alleles for linkage disequilibrium between those alleles to be used by the pipeline for genotype estimation; see `AddAlleleLinkages`.

`overdispersion` Overdispersion parameter; see `AddGenotypeLikelihood`.

`maxR2changeratio`

This number determines how many principal component axes are retained. The difference in R^2 values between the first and second axes is multiplied by `maxR2changeratio`. The last axis retained is the first axis after which the R^2 value changes by less than this value. Lower values of `maxR2changeratio` will result in more axes being retained.

Details

For `IterateHWE`, the following functions are run iteratively, assuming no population structure: `AddAlleleFreqHWE`, `AddGenotypePriorProb_HWE`, `AddGenotypeLikelihood`, `AddPloidyChiSq`, and `AddGenotypePosteriorProb`.

`IterateHWE_LD` runs each of the functions listed for `IterateHWE` once, then runs `AddAlleleLinkages`. It then runs `AddAlleleFreqHWE`, `AddGenotypePriorProb_HWE`, `AddGenotypePriorProb_LD`, `AddGenotypeLikelihood`, `AddPloidyChiSq`, and `AddGenotypePosteriorProb` iteratively until allele frequencies converge.

For `IteratePopStruct`, the following functions are run iteratively, modeling population structure: `AddPCA`, `AddAlleleFreqByTaxa`, `AddAlleleFreqHWE`, `AddGenotypePriorProb_ByTaxa`, `AddGenotypeLikelihood`, `AddPloidyChiSq`, and `AddGenotypePosteriorProb`. After the first PCA analysis, the number of principal component axes is not allowed to decrease, and can only increase by one from one round to the next, in order to help the algorithm converge.

`IteratePopStructLD` runs each of the functions listed for `IteratePopStruct` once, then runs `AddAlleleLinkages`. It then runs `AddAlleleFreqHWE`, `AddGenotypePriorProb_ByTaxa`, `AddGenotypePriorProb_LD`, `AddGenotypeLikelihood`, `AddPloidyChiSq`, `AddGenotypePosteriorProb`, `AddPCA`, and `AddAlleleFreqByTaxa` iteratively until convergence of allele frequencies.

Value

A "RADdata" object identical to that passed to the function, but with $alleleFreq, $priorProb, $depthSamplingPermutations, $genotypeLikelihood, $ploidyChiSq, and $posteriorProb slots added. For IteratePopStruct and IteratePopStructLD, $alleleFreqByTaxa and $PCA are also added. For IteratePopStructLD and IterateHWE_LD, $alleleLinkages and $priorProbLD are also added.

Note

If you see the error

Error in if (rel_ch < threshold & count > 5) { : missing value where TRUE/FALSE needed

try lowering nPcsInit.

Author(s)

Lindsay V. Clark

See Also

GetWeightedMeanGenotypes for outputting genotypes in a useful format after iteration is completed.

StripDown to remove memory-hogging slots that are no longer needed after the pipeline has been run.

PipelineMapping2Parents for mapping populations.

Examples

```
# load dataset
data(exampleRAD)

# iteratively estimate parameters
exampleRAD <- IterateHWE(exampleRAD)

# export results
GetWeightedMeanGenotypes(exampleRAD)

# re-load to run pipeline assuming population structure
data(exampleRAD)

# run pipeline
exampleRAD <- IteratePopStruct(exampleRAD, nPcsInit = 3)

# export results
GetWeightedMeanGenotypes(exampleRAD)

# dataset for LD pipeline
data(Msi01genes)

# run HWE + LD pipeline
```

```
mydata1 <- IterateHWE_LD(Msi01genes)

# run pop. struct + LD pipeline
# (tolerance raised to make example run faster)
mydata2 <- IteratePopStructLD(Msi01genes, tol = 0.01)
```

LocusInfo *Get Information about a Single Locus*

Description

This function returns, and optionally prints, information about a single locus with a RADdata object, including alignment position, allele sequences, and genes overlapping the site.

Usage

```
LocusInfo(object, ...)
## S3 method for class 'RADdata'
LocusInfo(object, locus, genome = NULL,
          annotation = NULL, verbose = TRUE, ...)
```

Arguments

object	A RADdata object.
locus	A character string indicating the name of the locus to display. Alternatively, a character string indicating the name of an allele, for which the corresponding locus will be identified.
genome	An optional FaFile or BSgenome object containing the reference genome sequence.
annotation	An optional TxDb object containing the genome annotation.
verbose	If TRUE, results will be printed to the console.
...	Additional arguments (none implemented).

Details

The locus name, allele names, and allele sequences are always returned (although allele names are not printed with verbose). If the chromosome and position are known, those are also returned and printed. If annotation is provided, the function will return and print genes that overlap the locus. If annotation and genome are provided, the function will attempt to identify any amino acid changes caused by the alleles, using predictCoding internally. Identification of amino acid changes will work if the RADdata object was created with VCF2RADdata using the refgenome argument to fill in non-variable sites, and/or if the alleles are only one nucleotide long.

Value

A list containing:

`Locus`	The name of the locus.
`Chromosome`	The chromosome name, if present.
`Position`	The position in base pairs on the chromosome, if present.
`Alleles`	Allele names for the locus.
`Haplotypes`	Allele sequences for the locus, in the same order.
`Frequencies`	Allele frequencies, if present, in the same order.
`Transcripts`	Transcripts overlapping the locus, if an annotation was provided but it wasn't possible to predict amino acid changes.
`PredictCoding`	The output of `predictCoding`, if it was run.

Author(s)

Lindsay V. Clark

See Also

`makeTxDbFromGFF`, `GetLoci`

Examples

```
data(exampleRAD)
exampleRAD <- AddAlleleFreqHWE(exampleRAD)
loc2info <- LocusInfo(exampleRAD, "loc2")
```

`MakeTasselVcfFilter` *Filter Lines of a VCF File By Call Rate and Allele Frequency*

Description

This function creates another function that can be used as a prefilter by the function `filterVcf` in the package **VariantAnnotation**. The user can set a minimum number of indiviuals with reads and a minimum number of individuals with the minor allele (either the alternative or reference allele). The filter can be used to generate a smaller VCF file before reading with `VCF2RADdata`.

Usage

```
MakeTasselVcfFilter(min.ind.with.reads = 200, min.ind.with.minor.allele = 10)
```

Arguments

`min.ind.with.reads`
: An integer indicating the minimum number of individuals that must have reads in order for a marker to be retained.

`min.ind.with.minor.allele`
: An integer indicating the minimum number of individuals that must have the minor allele in order for a marker to be retained.

Details

This function assumes the VCF file was output by the TASSEL GBSv2 pipeline. This means that each genotype field begins with two digits ranging from zero to three separated by a forward slash to indicate the called genotype, followed by a colon.

Value

A function is returned. The function takes as its only argument a character vector representing a set of lines from a VCF file, with each line representing one SNP. The function returns a logical vector the same length as the character vector, with `TRUE` if the SNP meets the threshold for call rate and minor allele frequency, and `FALSE` if it does not.

Author(s)

Lindsay V. Clark

References

https://bitbucket.org/tasseladmin/tassel-5-source/wiki/Tassel5GBSv2Pipeline

Examples

```
# make the filtering function
filterfun <- MakeTasselVcfFilter(300, 15)


# Executable code excluded from CRAN testing for taking >10 s:

require(VariantAnnotation)
# get the example VCF installed with polyRAD
exampleVCF <- system.file("extdata", "Msi01genes.vcf", package = "polyRAD")
exampleBGZ <- paste(exampleVCF, "bgz", sep = ".")

# zip and index the file using Tabix (if not done already)
if(!file.exists(exampleBGZ)){
  exampleBGZ <- bgzip(exampleVCF)
  indexTabix(exampleBGZ, format = "vcf")
}

# make a temporary file
# (for package checks; you don't need to do this in your own code)
outfile <- tempfile(fileext = ".vcf")
```

```
# filter to a new file
filterVcf(exampleBGZ, destination = outfile,
          prefilters = FilterRules(list(filterfun)))
```

MergeIdenticalHaplotypes *Merge Alleles with Identical DNA Sequences*

Description

If any alleles within a locus have identical `alleleNucleotides` values (including those identical based on IUPAC ambiguity codes), this function merges those alleles, summing their read depths. This function is primarily intended to be used internally in cases where tags vary in length within a locus, resulting in truncated `alleleNucleotides`.

Usage

```
MergeIdenticalHaplotypes(object, ...)
```

Arguments

`object`	A `RADdata` object.
`...`	Additional arguments (none implemented).

Value

A `RADdata` object identical to `object`, but with alleles merged.

Author(s)

Lindsay V. Clark

See Also

`MergeRareHaplotypes`, `readProcessIsoloci`

Examples

```
data(exampleRAD)
# change a haplotype for this example
exampleRAD$alleleNucleotides[5] <- "GY"

nAlleles(exampleRAD)
exampleRAD <- MergeIdenticalHaplotypes(exampleRAD)
nAlleles(exampleRAD)
```

MergeRareHaplotypes *Consolidate Reads from Rare Alleles*

Description

MergeRareHaplotypes searches for rare alleles in a "RADdata" object, and merges them into the most similar allele at the same locus based on nucleotide sequence (or the most common allele if multiple are equally similar). Read depth is summed across merged alleles, and the alleleNucleotides slot of the "RADdata" object contains IUPAC ambiguity codes to indicate nucleotide differences across merged alleles. This function is designed to be used immediately after data import.

Usage

```
MergeRareHaplotypes(object, ...)
## S3 method for class 'RADdata'
MergeRareHaplotypes(object, min.ind.with.haplotype = 10, ...)
```

Arguments

object	A "RADdata" object.
min.ind.with.haplotype	The minimum number of taxa having reads from a given allele for that allele to not be merged.
...	Additional arguments; none implemented.

Details

Alleles with zero reads across the entire dataset are removed by MergeRareHaplotypes without merging nucleotide sequences. After merging, at least one allele is left, even if it has fewer than min.ind.with.haplotype taxa with reads, as long as it has more than zero taxa with reads.

Value

A "RADdata" object identical to object, but with its $alleleDepth, $antiAlleleDepth, $depthRatio, $depthSamplingPermutations, $alleleNucleotides, and $alleles2loc arguments adjusted after merging alleles.

Author(s)

Lindsay V. Clark

See Also

SubsetByLocus, VCF2RADdata, readStacks

Examples

```
data(exampleRAD)
exampleRAD2 <- MergeRareHaplotypes(exampleRAD,
                                   min.ind.with.haplotype = 20)
exampleRAD$alleleDepth[21:30,6:7]
exampleRAD2$alleleDepth[21:30,6,drop=FALSE]
exampleRAD$alleleNucleotides
exampleRAD2$alleleNucleotides
```

MergeTaxaDepth *Combine Read Depths from Multiple Taxa into One Taxon*

Description

This function should be used in situations where data that were imported as separate taxa should be merged into a single taxon. The function should be used before any of the pipeline functions for genotype calling. Read depths are summed across duplicate taxa and output as a single taxon.

Usage

```
MergeTaxaDepth(object, ...)

## S3 method for class 'RADdata'
MergeTaxaDepth(object, taxa, ...)
```

Arguments

`object`	A `RADdata` object.
`taxa`	A character vector indicating taxa to be merged. The first taxon in the vector will be used to name the combined taxon in the output.
`...`	Additional arguments (none implemented).

Details

Examples of reasons to use this function:

- Duplicate samples across different libraries were given different names so that preliminary analysis could confirm that they were truly the same (*i.e.* no mix-ups) before combining them.
- Typos in the key file for the SNP mining software (TASSEL, Stacks, etc.) caused duplicate samples to have different names when they really should have had the same name.

To merge multiple sets of taxa into multiple combined taxa, this function can be run multiple times or in a loop.

Value

A `RADdata` object derived from `object`. The `alleleDepth`, `antiAlleleDepth`, `locDepth`, `depthRatio`, and `depthSamplingPermutation` slots, and `"taxa"` and `"nTaxa"` attributes, have been changed accordingly to reflect the merge.

Author(s)

Lindsay V. Clark

See Also

SubsetByTaxon

Examples

```
# dataset for this example
data(exampleRAD)

# merge the first three taxa into one
exampleRADm <- MergeTaxaDepth(exampleRAD, c("sample001", "sample002", "sample003"))

# inspect read depth
exampleRAD$alleleDepth[1:3,]
exampleRADm$alleleDepth[1:3,]
```

OneAllelePerMarker *Return the Index of One Allele for Each Locus*

Description

This function exists primarily to be called by functions such as AddPCA and GetWeightedMeanGenotypes that may need to exclude one allele per locus to avoid mathematical singularities. For a "RADdata" object, it returns the indices of one allele per locus.

Usage

```
OneAllelePerMarker(object, ...)
## S3 method for class 'RADdata'
OneAllelePerMarker(object, commonAllele = FALSE, ...)
```

Arguments

object	A "RADdata" object.
commonAllele	If TRUE, the index of the most common allele for each locus is returned, according to object$alleleFreq. If FALSE, the index of the first allele for each locus is returned.
...	Additional arguments (none implemented).

Value

An integer vector indicating the index of one allele for each locus in object.

Author(s)

Lindsay V. Clark

See Also

GetTaxa for a list of accessors.

Examples

```
data(exampleRAD)

OneAllelePerMarker(exampleRAD)

OneAllelePerMarker(exampleRAD, commonAllele = TRUE)
```

PipelineMapping2Parents
Run polyRAD Pipeline on a Mapping Population

Description

This function is a wrapper for AddAlleleFreqMapping, AddGenotypeLikelihood, AddGenotypePriorProb_Mapping2Pare AddPloidyChiSq, and AddGenotypePosteriorProb. It covers the full pipeline for estimating genotype posterior probabilities from read depth in a "RADdata" object containing data from a mapping population.

Usage

```
PipelineMapping2Parents(object, n.gen.backcrossing = 0,
                        n.gen.intermating = 0, n.gen.selfing = 0,
                        minLikelihoodRatio = 10, freqAllowedDeviation = 0.05,
                        freqExcludeTaxa = c(GetDonorParent(object),
                                            GetRecurrentParent(object),
                                            GetBlankTaxa(object)),
                        useLinkage = TRUE, linkageDist = 1e7,
                        minLinkageCorr = 0.5, overdispersion = 9)
```

Arguments

`object` A "RADdata" object.

`n.gen.backcrossing`
An integer, zero or greater, indicating how many generations of backcrossing to the recurrent parent were performed.

`n.gen.intermating`
An integer, zero or greater, indicating how many generations of intermating within the population were performed.

n.gen.selfing An integer, zero or greater, indicating how many generations of selfing were performed.

minLikelihoodRatio The minimum likelihood ratio for determining parental genotypes with confidence, to be passed to GetLikelyGen for both parental taxa.

freqAllowedDeviation For AddAlleleFreqMapping, the amount by which an allele frequency can deviate from an expected allele frequency in order to be counted as that allele frequency.

freqExcludeTaxa A character vector indicating taxa to exclude from allele frequency estimates and ploidy χ^2 estimates.

useLinkage Boolean. Should genotypes at nearby loci (according to genomic alignment data) be used for updating genotype priors?

linkageDist A number, in basepairs, indicating the maximum distance for linked loci. Ignored if useLinkage = FALSE.

minLinkageCorr A number ranging from zero to one. Indicates the minimum correlation coeffienct between weighted mean genotypes at two alleles in order for linkage data to be used for updating genotype priors. Ignored if useLinkage = FALSE.

overdispersion Overdispersion parameter; see AddGenotypeLikelihood.

Details

Unlike IterateHWE and IteratePopStruct, PipelineMapping2Parents only runs through each function once, rather than iteratively until convergence.

Value

A "RADdata" object identical to that passed to the function, with the following slots added: $alleleFreq, depthSamplingPermutations, $genotypeLikelihood, likelyGeno_donor, likelyGeno_recurrent, $priorProb, $ploidyChiSq, $posteriorProb, and if useLinkage = TRUE, $alleleLinkages and $priorProbLD. See the documentation for the functions listed in the description for more details on the data contained in these slots.

Author(s)

Lindsay V. Clark

See Also

SetDonorParent and SetRecurrentParent to indicate which individuals are the parents before running the function.

AddGenotypePriorProb_Mapping2Parents for how ploidy of parents and progeny is interpreted.

GetWeightedMeanGenotypes or Export_polymapR for exporting genotypes from the resulting object.

StripDown to remove memory-hogging slots that are no longer needed after the pipeline has been run.

Examples

```
# load data for the example
data(exampleRAD_mapping)

# specify donor and recurrent parents
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")

# run the pipeline
exampleRAD_mapping <- PipelineMapping2Parents(exampleRAD_mapping,
                                 n.gen.backcrossing = 1)

# export results
wmgeno <- GetWeightedMeanGenotypes(exampleRAD_mapping)[-(1:2),]
wmgeno
```

RADdata *RADdata object constructor*

Description

RADdata is used internally to generate objects of the S3 class “RADdata” by **polyRAD** functions for importing read depth data. It is also available at the user level for cases where the data for import are not already in a format supported by **polyRAD**.

Usage

```
RADdata(alleleDepth, alleles2loc, locTable, possiblePloidies, contamRate,
        alleleNucleotides, taxaPloidy)

## S3 method for class 'RADdata'
plot(x, ...)
```

Arguments

alleleDepth	An integer matrix, with taxa in rows and alleles in columns. Taxa names should be included as row names. Each value indicates the number of reads for a given allele in a given taxon. There should be no NA values; use zero to indicate no reads.
alleles2loc	An integer vector with one value for each column of alleleDepth. The number indicates the identity of the locus to which the allele belongs. A locus can have any number of alleles assigned to it (including zero).
locTable	A data frame, where locus names are row names. There must be at least as many rows as the highest value of alleles2loc; each number in alleles2loc corresponds to a row index in locTable. No columns are required, although if provided a column named “Chr” will be used for indicating chromosome identities, a column named “Pos” will be used for indicating physical position, and a column named “Ref” will be used to indicate the reference sequence.

`possiblePloidies`
: A list, where each item in the list is an integer vector (or a numeric vector that can be converted to integer). Each vector indicates an inheritance pattern that markers in the dataset might obey. `2` indicates diploid, `4` indicates autotetraploid, `c(2, 2)` indicates allotetraploid, *etc.*

`contamRate`
: A number ranging from zero to one (although in practice probably less than 0.01) indicating the expected sample cross-contamination rate.

`alleleNucleotides`
: A character vector with one value for each column of `alleleDepth`, indicating the DNA sequence for that allele. Typically only the sequence at variable sites is provided, although intervening non-variable sequence can also be provided.

`taxaPloidy`
: An integer vector indicating ploidies of taxa. If a single value is provided, it will be assumed that all taxa are the same ploidy. Otherwise, one value must be provided for each taxon. If unnamed, it is assumed that taxa are in the same order as the rows of `alleleDepth`. If named, names must match the row names of `alleleDepth` but do not need to be in the same order. This value is used as a multiplier with `possiblePloidies`; see Details.

`x`
: A “RADdata” object.

`...`
: Additional arguments to pass to `plot`, for example `col` or `pch`.

Details

For a single locus, ideally the string provided in `locTable$Ref` and all strings in `alleleNucleotides` are the same length, so that SNPs and indels may be matched by position. The character “-” indicates a deletion with respect to the reference, and can be used within `alleleNucleotides`. The character “.” is a placeholder where other alleles have an insertion with respect to the reference, and may be used in `locTable$Ref` and `alleleNucleotides`. Note that it is possible for the sequence in `locTable$Ref` to be absent from `alleleNucleotides` if the reference haplotype is absent from the dataset, as may occur if the reference genome is that of a related species and not the actual study species. For the `alleleNucleotides` vector, the attribute `"Variable_sites_only"` indicates whether non-variable sequence in between variants is included; this needs to be `FALSE` for other functions to determine the position of each variant within the set of tags.

Inheritance mode is determined by multiplying the values in `possiblePloidies` by the values in `taxaPloidy` and dividing by two. For example, if you wanted to assume autotetraploid inheritance across the entire dataset, you could set `possiblePloidies = list(4)` and `taxaPloidy = 2`, or alternatively `possiblePloidies = list(2)` and `taxaPloidy = 4`. To indicate a mix of diploid and allotetraploid inheritance across loci, set `possiblePloidies = list(2, c(2, 2))` and `taxaPloidy = 2`. If taxa themselves vary in ploidy, provide one value of `taxaPloidy` for each taxon. All inheritance modes listed in `possiblePloidies` apply equally to all taxa, even when ploidy varies by taxon.

Value

An object of the S3 class “RADdata”. The following slots are available using the `$` operator:

`alleleDepth`
: Identical to the argument provided to the function.

`alleles2loc`
: Identical to the argument provided to the function.

locTable
: Identical to the argument provided to the function.

possiblePloidies
: The possiblePloidies argument, converted to integer.

locDepth
: A matrix with taxa in rows and loci in columns, with read depth summed across all alleles for each locus. Column names are locus numbers rather than locus names. See GetLocDepth for retrieving the same matrix but with locus names as column names.

depthSamplingPermutations
: A numeric matrix with taxa in rows and alleles in columns. It is calculated as $log(locDepth choose alleleDepth)$. This is used as a coefficient for likelihood estimations done by other **polyRAD** functions (*i.e.* AddGenotypeLikelihood).

depthRatio
: A numeric matrix with taxa in rows and alleles in columns. Calculated as $alleleDepth/locDepth$. Used by other **polyRAD** functions for rough estimation of genotypes and allele frequency.

antiAlleleDepth
: An integer matrix with taxa in rows and alleles in columns. For each allele, the number of reads from the locus that do NOT belong to that allele. Calculated as $locDepth - alleleDepth$. Used for likelihood estimations by other **polyRAD** functions.

alleleNucleotides
: Identical to the argument provided to the function.

taxaPloidy
: A named integer vector with one value per taxon, indicating the ploidy of taxa.

The object additionally has several attributes (see attr):

taxa
: A character vector listing all taxa names, in the same order as the rows of alleleDepth.

nTaxa
: An integer indicating the number of taxa.

nLoc
: An integer indicating the number of loci in locTable.

contamRate
: Identical to the argument provided to the function.

The plot method performs a principal components analysis with AddPCA if not already done, then plots the first two axes. Points represent individuals (taxa). If mapping population parents have been noted in the object (see SetDonorParent), they are indicated in the plot.

Author(s)

Lindsay V. Clark

See Also

Data import functions that internally call RADdata:

readHMC, readTagDigger, VCF2RADdata, readStacks, readTASSELGBSv2, readProcessSamMulti, readProcessIsoloci

Examples

```
# create the dataset
mydepth <- matrix(sample(100, 16), nrow = 4, ncol = 4,
                  dimnames = list(paste("taxon", 1:4, sep = ""),
                  paste("loc", c(1,1,2,2), "_", c(0,1,0,1), sep = "")))
mydata <- RADdata(mydepth, c(1L,1L,2L,2L),
                  data.frame(row.names = c("loc1", "loc2"), Chr = c(1,1),
                             Pos = c(2000456, 5479880)),
                  list(2, c(2,2)), 0.001, c("A", "G", "G", "T"), 6)

# inspect the dataset
mydata
mydata$alleleDepth
mydata$locDepth
mydata$depthRatio
mydata$taxaPloidy

# the S3 class structure is flexible; other data can be added
mydata$GPS <- data.frame(row.names = attr(mydata, "taxa"),
                         Lat = c(43.12, 43.40, 43.05, 43.27),
                         Long = -c(70.85, 70.77, 70.91, 70.95))
mydata$GPS

# If you have NA in your alleleDepth matrix to indicate zero reads,
# perform the following before running the RADdata constructor:
mydepth[is.na(mydepth)] <- 0L

# plotting a RADdata object
plot(mydata)
```

RADdata2VCF *Export RADdata Genotypes to VCF*

Description

Converts genotype calls from **polyRAD** into VCF format. The user may send the results directly to a file, or to a CollapsedVCF for further manipulation.

Usage

```
RADdata2VCF(object, file = NULL, asSNPs = TRUE, hindhe = TRUE,
            sampleinfo = data.frame(row.names = GetTaxa(object)),
            contigs = data.frame(row.names = unique(object$locTable$Chr)))
```

Arguments

object	A RADdata object in which genotype calling has been performed. It is also important for the data to have been imported in a way that preserves variant positions (i.e. readProcessIsoloci, readTASSELGBSv2, VCF2RADdata using the refgenome argument).

`file` An optional character string or connection indicating where to write the file. Append mode may be used with connections if multiple RADdata objects need to be written to one VCF.

`asSNPs` Boolean indicating whether to convert haplotypes to individual SNPs and indels.

`hindhe` Boolean indicating whether to export a mean value of Hind/He (see `HindHe`) for every sample and locus.

`sampleinfo` A data frame with optional columns indicating any sample metadata to export to "SAMPLE" header lines.

`contigs` A data frame with optional columns providing information about contigs to export to "contig" header lines.

Details

Currently, the FORMAT fields exported are GT (genotype), AD (allelic read depth), and DP (read depth). Genotype posterior probabilities are not exported due to the mathematical intractability of converting pseudo-biallelic probabilities to multiallelic probabilities.

Genotypes exported to the GT field are obtained internally using `GetProbableGenotypes`.

INFO fields exported include the standard fields NS (number of samples with more than zero reads) and DP (total depth across samples) as well as the custom fields LU (index of the marker in the original RADdata object) and HH (Hind/He statistic for the marker).

This function requires the BioConductor package **VariantAnnotation**. See `https://bioconductor.org/packages/release/bioc/html/VariantAnnotation.html` for installation instructions.

Value

A `CollapsedVCF` object.

Author(s)

Lindsay V. Clark

References

`https://samtools.github.io/hts-specs/VCFv4.3.pdf`

See Also

`VCF2RADdata`, `ExportGAPIT`

Examples

```
# Set up example dataset for export.
# You DO NOT need to adjust attr or locTable in your own dataset.
data(exampleRAD)
attr(exampleRAD$alleleNucleotides, "Variable_sites_only") <- FALSE
exampleRAD$locTable$Ref <-
  exampleRAD$alleleNucleotides[match(1:nLoci(exampleRAD), exampleRAD$alleles2loc)]
exampleRAD <- IterateHWE(exampleRAD)
```

```
# An optional table of sample data
sampleinfo <- data.frame(row.names = GetTaxa(exampleRAD),
                         Population = rep(c("North", "South"), each = 50))

# Add contig information (fill in with actual data rather than random)
mycontigs <- data.frame(row.names = c("1", "4", "6", "9"), length = sample(1e8, 4),
                        URL = rep("ftp://mygenome.com/mygenome.fa", 4))

# Set up a file destination for this example
# (It is not necessary to use tempfile with your own data)
outfile <- tempfile(fileext = ".vcf")


# Export VCF
testvcf <- RADdata2VCF(exampleRAD, file = outfile, sampleinfo = sampleinfo,
                       contigs = mycontigs)
```

readDArTag *Import Data from DArT Sequencing*

Description

Diversity Array Technologies (DArT) provides a tag-based genotyping-by-sequencing service. Together with Breeding Insight, a format was developed indicting haplotype sequence and read depth, and that format is imported by this function to make a RADdata object. The target SNP and all off-target SNPs within the amplicon are imported as haplotypes. Because the file format does not indicate strandedness of the tag, BLAST results are used so that sequence and position are accurately stored in the RADdata object. See the "extdata" folder of the **polyRAD** installation for example files.

Usage

```
readDArTag(file, botloci = NULL, blastfile = NULL, excludeHaps = NULL,
           includeHaps = NULL, n.header.rows = 0, sample.name.row = 1,
           trim.sample.names = "_[^_]+_[ABCDEFGH][[:digit:]][012]?$",
           sep.counts = ",", sep.blast = "\t", possiblePloidies = list(2),
           taxaPloidy = 2L, contamRate = 0.001)
```

Arguments

`file` The file name of a spreadsheet from DArT indicating haplotype sequence and read depth.

`botloci` A character vector indicating the names of loci for which the sequence is on the bottom strand with respect to the reference genome. All other loci are assumed to be on the top strand. Only one of `blastfile` and `botloci` should be provided.

`blastfile` File name for BLAST results for haplotypes. The file should be in tabular format with `qseqid`, `sseqid`, `sstart`, `send`, and `pident` columns, indicated with column headers. Only one of `blastfile` and `botloci` should be provided.

`excludeHaps` Optional. Character vector with names of haplotypes (from the "AlleleID" column) that should not be imported. Should not be used if `includeHaps` is provided.

`includeHaps` Optional. Character vector with names of haplotypes (from the "AlleleID" column) that should be imported. Should not be used if `excludeHaps` is provided.

`n.header.rows` Integer. The number of header rows in `file`, not including the full row of column headers.

`sample.name.row`
Integer. The row within `file` from which sample names should be derived.

`trim.sample.names`
A regular expression indicating text to trim off of sample names. Use `""` if no trimming should be performed.

`sep.counts` The field separator character for `file`. The default assumes CSV.

`sep.blast` The field separator character for the BLAST results. The default assumes tab-delimited.

`possiblePloidies`
A list indicating possible inheritance modes. See `RADdata`.

`taxaPloidy` A single integer, or an integer vector with one value per taxon, indicating ploidy. See `RADdata`.

`contamRate` Expected sample cross-contamination rate. See `RADdata`.

Details

The "CloneID" column is used for locus names, and is assumed to contain the chromosome (or scaffold) name and position, separated by an underscore. The position is assumed to refer to the target SNP, which is identified by comparing the "Ref_001" and "Alt_002" sequences. The position is then converted to refer to the beginning of the tag (which may have been reverse complemented depending on BLAST results), since additional SNPs may be present. This facilitates accurate export to VCF using `RADdata2VCF`.

Column names for the BLAST file can be "Query", "Subject", "S_start", "S_end", and "%Identity", for compatibility with Breeding Insight formats.

Value

A `RADdata` object ready for QC and genotype calling. Assuming the "Ref_001" and "Alt_002" alleles were not excluded, the `locTable` slot will include columns for chromosome, position, strand, and reference sequence.

Author(s)

Lindsay V. Clark

References

https://www.diversityarrays.com/

https://breedinginsight.org/

See Also

reverseComplement

readTagDigger, VCF2RADdata, readStacks, readTASSELGBSv2, readHMC

RADdata2VCF

Examples

```
## Older Excellence in Breeding version
# Example files installed with polyRAD
dartfile <- system.file("extdata", "DArTag_example.csv", package = "polyRAD")
blastfile <- system.file("extdata", "DArTag_BLAST_example.txt",
                         package = "polyRAD")

# One haplotype doesn't seem to have correct alignment (see BLAST results)
exclude_hap <- c("Chr1_30668472|RefMatch_004")

# Import data
mydata <- readDArTag(dartfile, blastfile = blastfile,
                      excludeHaps = exclude_hap,
                      possiblePloidies = list(4),
                      n.header.rows = 7, sample.name.row = 7)

## Newer Excellence in Breeding version (2022)
# Example files installed with polyRAD
dartfile <- system.file("extdata", "DArTag_example2.csv", package = "polyRAD")
blastfile <- system.file("extdata", "DArTag_BLAST_example2.txt",
                         package = "polyRAD")

# One haplotype doesn't seem to have correct alignment (see BLAST results)
exclude_hap <- c("Chr1_30668472|RefMatch_0004")

# Import data
mydata <- readDArTag(dartfile, blastfile = blastfile,
                      excludeHaps = exclude_hap,
                      possiblePloidies = list(4),
                      n.header.rows = 0, sample.name.row = 1)
```

readHMC *Import read depth from UNEAK*

Description

This function reads the “HapMap.hmc.txt” and “HapMap.fas.txt” files output by the UNEAK pipeline and uses the data to generate a “RADdata” object.

Usage

```
readHMC(file, includeLoci = NULL, shortIndNames = TRUE,
        possiblePloidies = list(2), taxaPloidy = 2L, contamRate = 0.001,
        fastafile = sub("hmc.txt", "fas.txt", file, fixed = TRUE))
```

Arguments

`file`	Name of the file containing read depth (typically "HapMap.hmc.txt").
`includeLoci`	An optional character vector of loci to be included in the output.
`shortIndNames`	Boolean. If TRUE, taxa names will be shortened with respect to those in the file, eliminating all text after and including the first underscore.
`possiblePloidies`	A list of numeric vectors indicating potential inheritance modes of SNPs in the dataset. See `RADdata`.
`taxaPloidy`	A single integer, or an integer vector with one value per taxon, indicating ploidy. See `RADdata`.
`contamRate`	A number ranging from zero to one (typically small) indicating the expected rate of sample cross-contamination.
`fastafile`	Name of the file containing tag sequences (typically "HapMap.fas.txt").

Value

A `RADdata` object containing read depth, taxa and locus names, and nucleotides at variable sites.

Note

UNEAK is not able to report read depths greater than 127, which may be problematic for high depth data on polyploid organisms. The UNEAK pipeline is no longer being updated and is currently only available with archived versions of TASSEL.

Author(s)

Lindsay V. Clark

References

Lu, F., Lipka, A. E., Glaubitz, J., Elshire, R., Cherney, J. H., Casler, M. D., Buckler, E. S. and Costich, D. E. (2013) Switchgrass genomic diversity, ploidy, and evolution: novel insights from a network-based SNP discovery protocol. *PLoS Genetics* **9**, e1003215.

https://www.maizegenetics.net/tassel

https://tassel.bitbucket.io/TasselArchived.html

See Also

`readTagDigger`, `VCF2RADdata`, `readStacks`, `readTASSELGBSv2`, `readDArTag`

Examples

```
# for this example we'll create dummy files rather than using real ones
hmc <- tempfile()
write.table(data.frame(rs = c("TP1", "TP2", "TP3"),
                       ind1_merged_X3 = c("15|0", "4|6", "13|0"),
                       ind2_merged_X3 = c("0|0", "0|1", "0|5"),
                       HetCount_allele1 = c(0, 1, 0),
                       HetCount_allele2 = c(0, 1, 0),
                       Count_allele1 = c(15, 4, 13),
                       Count_allele2 = c(0, 7, 5),
                       Frequency = c(0, 0.75, 0.5)), row.names = FALSE,
            quote = FALSE, col.names = TRUE, sep = "\t", file = hmc)
fas <- tempfile()
writeLines(c(">TP1_query_64",
             "TGCAGAAAAAAAACGCTCGATGCCCCCTAATCCGTTTTCCCCATTCCGCTCGCCCCATCGGAGT",
             ">TP1_hit_64",
             "TGCAGAAAAAAAACGCTCGATGCCCCCTAATCCGTTTTCCCCATTCCGCTCGCCCCATTGGAGT",
             ">TP2_query_64",
             "TGCAGAAAAACAACACCCTAGGTAACAACCATATCTTATATTGCCGAATAAAAAACAACACCCC",
             ">TP2_hit_64",
             "TGCAGAAAAACAACACCCTAGGTAACAACCATATCTTATATTGCCGAATAAAAAATAACACCCC",
             ">TP3_query_64",
             "TGCAGAAAACATGGAGAGGGAGATGGCACGGCAGCACCACCGCTGGTCCGCTGCCCGTTTGCGG",
             ">TP3_hit_64",
             "TGCAGAAAACATGGAGATGGAGATGGCACGGCAGCACCACCGCTGGTCCGCTGCCCGTTTGCGG"),
             fas)

# now read the data
mydata <- readHMC(hmc, fastafile = fas)

# inspect the results
mydata
mydata$alleleDepth
mydata$alleleNucleotides
row.names(mydata$locTable)
```

---

readProcessIsoloci *Import Read Depth from Output of process_isoloci.py*

Description

After process_isoloci.py is used to assign RAD tags to alignment locations within a highly duplicated genome, `readProcessIsoloci` imports the resulting CSV to a "`RADdata`" object.

Usage

```
readProcessIsoloci(sortedfile, min.ind.with.reads = 200,
                   min.ind.with.minor.allele = 10,
                   min.median.read.depth = 10,
                   possiblePloidies = list(2), taxaPloidy = 2L,
                   contamRate = 0.001,
                   nameFromTagStart = TRUE, mergeRareHap = TRUE)
```

Arguments

`sortedfile` File path to a CSV output by process_isoloci.py.

`min.ind.with.reads`
Minimum number of individuals with reads needed to retain a locus.

`min.ind.with.minor.allele`
Minimum number of individuals with reads in a minor allele needed to retain a locus.

`min.median.read.depth`
Minimum median read depth across individuals (including individuals with depth 0) needed to retain a locus.

`possiblePloidies`
A list indicating possible inheritance modes of loci. See `RADdata`.

`taxaPloidy` A single integer, or an integer vector with one value per taxon, indicating ploidy. See `RADdata`.

`contamRate` Approximate rate of cross-contamination among samples.

`nameFromTagStart`
If `TRUE` loci will be named based on the alignment position and strand of the RAD tag itself. If `FALSE`, loci will be named based on the leftmost position of the variable region of the RAD tag. In either case, `locTable$Pos` within the output will indicate the position of the variable region of the tag.

`mergeRareHap` Boolean indicating whether to run `MergeRareHaplotypes` after building the `"RADdata"` object.

Details

`MergeIdenticalHaplotypes` is used internally by this function to merge alleles with identical sequence for the region shared by all tags, in cases where tags vary in length within a locus.

Value

A `"RADdata"` object containing read depth and alignment positions from `sortedfile`.

Author(s)

Lindsay V. Clark

See Also

`readProcessSamMulti`

readProcessSamMulti *Import Preliminary Data to Determine Parameters for Isolocus Sorting*

Description

This function imports the files output by process_sam_multi.py to a "RADdata" object so that HindHe can be run to filter samples and determine optimal parameters for process_isoloci.py.

Usage

```
readProcessSamMulti(alignfile,
                    depthfile = sub("align", "depth", alignfile),
                    expectedLoci = 1000,
                    min.ind.with.reads = 200,
                    min.ind.with.minor.allele = 10,
                    possiblePloidies = list(2), taxaPloidy = 2L,
                    contamRate = 0.001,
                    expectedAlleles = expectedLoci * 15,
                    maxLoci = expectedLoci)
```

Arguments

alignfile A file output by process_sam_multi.py, generally in the format prefix_1_align.csv.

depthfile A file output by process_sam_multi.py, generally in the format prefix_1_depth.csv.

expectedLoci The number of loci expected in the final object. The default, 1000, is fairly small because this function is intended to be used for preliminary analysis only.

min.ind.with.reads
The minimum number of taxa with reads needed in order for a locus to be retained in the output.

min.ind.with.minor.allele
The minimum number of taxa with the same minor allele needed in order for a locus to be retained in the output.

possiblePloidies
A list indicating expected inheritance modes for markers. See RADdata.

taxaPloidy A single integer, or an integer vector with one value per taxon, indicating ploidy. See RADdata.

contamRate A number ranging from zero to one (although in practice probably less than 0.01) indicating the expected sample cross-contamination rate.

expectedAlleles
The expected number of alleles in the dataset.

maxLoci The maximum number of loci to import before ceasing to read the file. Set to Inf if you want to read the entire file.

Value

A "RADdata" object.

Author(s)

Lindsay V. Clark

See Also

readProcessIsoloci

Examples

```
## Not run:
myRAD <- readProcessSamMulti("mydata_2_align.csv")

## End(Not run)
```

---

readStacks *Import Read Depth from Stacks*

Description

Using the catalog files output by cstacks and matches file output by sstacks, this function imports read depth into a RADdata object. If genomic alignments were used, alignment data can optionally be imported.

Usage

```
readStacks(allelesFile, matchesFolder, version = 2,
           min.ind.with.reads = 200,
           min.ind.with.minor.allele = 10, readAlignmentData = FALSE,
           sumstatsFile = "populations.sumstats.tsv",
           possiblePloidies = list(2), taxaPloidy = 2L, contamRate = 0.001)
```

Arguments

allelesFile	Path to the "alleles" file from the Stacks catalog.
matchesFolder	Path to the folder containing "matches" files to import.
version	Either the number 1 or 2, indicating the version of Stacks.
min.ind.with.reads	For filtering loci. A locus must have at least this many samples with reads in order to be retained.

min.ind.with.minor.allele
: For filtering loci. A locus must have at least this many samples with reads for the minor allele in order to be retained. For loci with more than two alleles, at least two alleles must be present in at least this many individuals. This argument is also passed internally to the min.ind.with.haplotype argument of MergeRareHaplotypes to consolidate reads from rare alleles.

readAlignmentData
: If TRUE and version = 1, the "tags" file from the Stacks catalog will be read, and chromosome, position, and strand will be imported to the locTable slot of the output. It is assumed that the "tags" file is in the same directory as the "alleles" file. If TRUE and version = 2, sumstatsFile will be used for import of chromosome and position data.

sumstatsFile
: The name of the file containing summary statistics for loci. Ignored unless version = 2 and readAlignmentData = TRUE.

possiblePloidies
: A list indicating possible inheritance modes in the dataset. See RADdata.

taxaPloidy
: A single integer, or an integer vector with one value per taxon, indicating ploidy. See RADdata.

contamRate
: A number from 0 to 1 (generally very small) indicating the expected rate of cross contamination between samples.

Value

A RADdata object.

Note

This function has been tested with output from Stacks 1.47.

Author(s)

Lindsay V. Clark

References

Stacks website: http://catchenlab.life.illinois.edu/stacks/

Rochette, N. and Catchen, J. (2017) Deriving genotypes from RAD-seq short-read data using Stacks. *Nature Protocols* **12**, 2640–2659.

Catchen, J., Hohenlohe, P. A., Bassham, S., Amores, A., and Cresko., W. A. (2013) Stacks: an analysis tool set for population genomics. *Molecular Ecology* **22**, 3124–3140.

Catchen, J. M., Amores, A., Hohenlohe, P., Cresko, W., and Postlethwait, J. H. (2011) Stacks: building and genotyping loci de novo from short-read sequences. *G3: Genes, Genomes, Genetics* **1**, 171–182.

See Also

VCF2RADdata, readTagDigger, readHMC, readTASSELGBSv2, readDArTag

Examples

```
## Not run:

# Assuming the working directory contains the catalog and all matches files:

myStacks <- readStacks("batch_1.catalog.alleles.tsv", ".",
                       version = 1,
                       readAlignmentData = TRUE)

## End(Not run)
```

readTagDigger *Import Read Counts from TagDigger*

Description

readTagDigger reads the CSV output containing read counts from TagDigger and generates a "RADdata" object. Optionally, it can also import a tag database generated by the Tag Manager program within TagDigger, containing information such as alignment position, to be stored in the $locTable slot of the "RADdata" object.

Usage

```
readTagDigger(countfile, includeLoci = NULL,
              possiblePloidies = list(2), taxaPloidy = 2L,
              contamRate = 0.001,
              dbfile = NULL, dbColumnsToKeep = NULL,
              dbChrCol = "Chr", dbPosCol = "Pos",
              dbNameCol = "Marker name")
```

Arguments

countfile
Name of the file containing read counts.

includeLoci
An optional character vector containing names of loci to retain in the output.

possiblePloidies
A list of numeric vectors indicating potential inheritance modes of SNPs in the dataset. See RADdata.

taxaPloidy
A single integer, or an integer vector with one value per taxon, indicating ploidy. See RADdata.

contamRate
A number ranging from zero to one (typically small) indicating the expected rate of sample cross-contamination.

dbfile
Optionally, name of the Tag Manager database file.

dbColumnsToKeep
Optionally, a character vector indicating the names of columns to keep from the database file.

dbChrCol	The name of the column containing the chromosome number in the database file.
dbPosCol	The name of the column indicating alignment position in the database file.
dbNameCol	The name of the column containing marker names in the database file.

Details

Nucleotides associated with the alleles, to be stored in the $alleleNucleotides slot, are extracted from the allele names in the read counts file. It is assumed that the allele names first contain the marker name, followed by an underscore, followed by the nucleotide(s) at any variable positions.

Value

A "RADdata" object.

Author(s)

Lindsay V. Clark

References

https://github.com/lvclark/tagdigger

Clark, L. V. and Sacks, E. J. (2016) TagDigger: User-friendly extraction of read counts from GBS and RAD-seq data. *Source Code for Biology and Medicine* **11**, 11.

See Also

readHMC, readStacks, VCF2RADdata, readTASSELGBSv2, readDArTag

Examples

```
# for this example we'll create dummy files
countfile <- tempfile()
write.csv(data.frame(row.names = c("Sample1", "Sample2", "Sample3"),
                     Mrkr1_A_0 = c(0, 20, 4),
                     Mrkr1_G_1 = c(7, 0, 12)),
          file = countfile, quote = FALSE)
dbfile <- tempfile()
write.csv(data.frame(Marker.name = "Mrkr1", Chr = 5, Pos = 66739827),
          file = dbfile, row.names = FALSE, quote = FALSE)

# read the data
myrad <- readTagDigger(countfile, dbfile = dbfile)
```

readTASSELGBSv2 *Import Read Depth and Alignment from TASSEL GBS v2*

Description

This function reads TagTaxaDist and SAM files output by the TASSEL 5 GBS v2 pipeline, and generates a `RADdata` object suitable for downstream processing for genotype estimation. It elimintes the need to run the DiscoverySNPCallerPluginV2 or the ProductionSNPCallerPluginV2, since **polyRAD** operates on haplotypes rather than SNPs.

Usage

```
readTASSELGBSv2(tagtaxadistFile, samFile, min.ind.with.reads = 200,
                min.ind.with.minor.allele = 10, possiblePloidies = list(2),
                taxaPloidy = 2L, contamRate = 0.001, chromosomes = NULL)
```

Arguments

`tagtaxadistFile`
: File name or path to a tab-delimited text file of read depth generated by the GetTagTaxaDistFromDBPlugin in TASSEL.

`samFile`
: File name or path to the corresponding SAM file containing alignment information for the same set of tags. This file is obtained by running the TagExportToFastqPlugin in TASSEL, followed by alignment using Bowtie2 or BWA.

`min.ind.with.reads`
: Integer used for marker filtering. The minimum number of individuals that must have read depth above zero for a locus to be retained in the output.

`min.ind.with.minor.allele`
: Integer used for marker filtering. The minimum number of individuals possessing reads for the minor allele for a locus to be retained in the output. This value is also passed to the `min.ind.with.haplotype` argument of `MergeRareHaplotypes`.

`possiblePloidies`
: A list indicating inheritance modes that might be encountered in the dataset. See `RADdata`.

`taxaPloidy`
: A single integer, or an integer vector with one value per taxon, indicating ploidy. See `RADdata`.

`contamRate`
: A number indicating the expected sample cross-contamination rate. See `RADdata`.

`chromosomes`
: A character vector of chromosome names, indicating chromosomes to be retained in the output. If `NULL`, all chromosomes to be retained. This argument is intended to be used for reading data in a chromosome-wise fashion in order to conserve computer memory.

Value

A `RADdata` object containing read depth and alignment infomation from the two input files.

Note

Sequence tags must be identical in length to be assigned to the same locus by this function. This is to prevent errors with `MergeRareHaplotypes`.

Author(s)

Lindsay V. Clark

References

TASSEL GBSv2 pipeline: `https://bitbucket.org/tasseladmin/tassel-5-source/wiki/Tassel5GBSv2Pipeline`

Bowtie2: `https://bowtie-bio.sourceforge.net/bowtie2/index.shtml`

BWA: `https://bio-bwa.sourceforge.net/`

See Also

Other data import functions: `readStacks`, `readHMC`, `readTagDigger`, `VCF2RADdata`, `readDArTag`

Examples

```
# get files for this example
samfile <- system.file("extdata", "exampleTASSEL_SAM.txt",
                       package = "polyRAD")
ttdfile <- system.file("extdata", "example_TagTaxaDist.txt",
                       package = "polyRAD")

# import data
myrad <- readTASSELGBSv2(ttdfile, samfile, min.ind.with.reads = 8,
                         min.ind.with.minor.allele = 2)
```

`reverseComplement` *Reverse Complement of DNA Sequence Stored as Character String*

Description

Whereas the `reverseComplement` function available in **Biostrings** only functions on `XString` and `XStringSet` objects, the version in **polyRAD** also works on character strings. It is written as an S4 method in order to avoid conflict with **Biostrings**. It is primarily included for internal use by **polyRAD**, but may be helpful at the user level as well.

Usage

```
reverseComplement(x, ...)
```

Arguments

`x`	A vector of character strings indicating DNA sequence using IUPAC codes.
`...`	Additional arguments (none implemented)

Value

A character vector.

Author(s)

Lindsay V. Clark

See Also

`readDArTag` uses this function internally.

Examples

```
reverseComplement(c("AAGT", "CCA"))
```

`SetBlankTaxa` *Functions to Assign Taxa to Specific Roles*

Description

These functions are used for assigning and retrieving taxa from a `"RADdata"` object that serve particular roles in the dataset. Blank taxa can be used for estimating the contamination rate (see `EstimateContaminationRate`), and the donor and recurrent parents are used for determining expected genotype distributions in mapping populations. Many functions in **polyRAD** will automatically exclude taxa from analysis if they have been assigned to one of these roles.

Usage

```
SetBlankTaxa(object, value)
GetBlankTaxa(object, ...)
SetDonorParent(object, value)
GetDonorParent(object, ...)
SetRecurrentParent(object, value)
GetRecurrentParent(object, ...)
```

Arguments

`object`	A `"RADdata"` object.
`value`	A character string (or a character vector for `SetBlankTaxa`) indicating the taxon or taxa to be assigned to the role.
`...`	Other arguments (none currently supported).

Value

For the “Get” functions, a character vector indicating the taxon or taxa that have been assigned to that role. For the “Set” functions, a `"RADdata"` object identical to the one passed to the function, but with new taxa assigned to that role.

Author(s)

Lindsay V. Clark

See Also

`AddGenotypePriorProb_Mapping2Parents`

Examples

```
# assign parents in a mapping population
data(exampleRAD_mapping)
exampleRAD_mapping <- SetDonorParent(exampleRAD_mapping, "parent1")
exampleRAD_mapping <- SetRecurrentParent(exampleRAD_mapping, "parent2")
GetDonorParent(exampleRAD_mapping)
GetRecurrentParent(exampleRAD_mapping)

# assign blanks
exampleRAD_mapping <- SetBlankTaxa(exampleRAD_mapping,
                                   c("progeny019", "progeny035"))
GetBlankTaxa(exampleRAD_mapping)
```

StripDown *Remove Unneeded Slots to Conserve Memory*

Description

This function is designed to be used after a `RADdata` object has been processed by one of the pipeline functions. Slots that are no longer needed are removed in order to conserve memory.

Usage

```
StripDown(object, ...)
## S3 method for class 'RADdata'
StripDown(object,
          remove.slots = c("depthSamplingPermutations",
                           "depthRatio", "antiAlleleDepth",
                           "genotypeLikelihood", "priorProb",
                           "priorProbLD"),
          ...)
```

Arguments

`object`	A `RADdata` object.
`remove.slots`	A character vector listing slots that will be removed.
`...`	Additional arguments (none implemented).

Details

The default slots that are removed take up a lot of memory but are not used by the export functions. Other slots to consider removing are alleleFreq, alleleFreqByTaxa, PCA, locDepth, alleleDepth, and alleleLinkages. Of course, if you have custom uses for some of the slots that are removed by default, you can change the remove.slots vector to not include them.

The function will throw an error if the user attempts to remove key slots that are needed for export and downstream analysis, including:

- alleles2loc
- alleleNucleotides
- locTable
- possiblePloidies
- ploidyChiSq
- posteriorProb

Value

A RADdata object

Author(s)

Lindsay V. Clark

See Also

SubsetByTaxon, SubsetByLocus

Examples

```
# load a dataset for this example
data(exampleRAD)

# run a pipeline
exampleRAD <- IterateHWE(exampleRAD)

# check the size of the resulting object
object.size(exampleRAD)

# remove unneeded slots
exampleRAD <- StripDown(exampleRAD)

# check object size again
object.size(exampleRAD)
```

SubsetByLocus *Create RADdata Objects with a Subset of Loci*

Description

These functions take a RADdata object as input and generate smaller RADdata objects containing only the specified loci. SubsetByLocus allows the user to specify which loci are kept, whereas SplitByChromosome creates multiple RADdata objects representing chromosomes or sets of chromosomes. RemoveMonomorphicLoci eliminates any loci with fewer than two alleles. RemoveHighDepthLoci eliminates loci that have especially high read depth in order to eliminate false loci originating from repetitive sequence. RemoveUngenotypedLoci is intended for datasets that have been run through PipelineMapping2Parents and may have some genotypes that are missing or non-variable due to how priors were determined.

Usage

```
SubsetByLocus(object, ...)
## S3 method for class 'RADdata'
SubsetByLocus(object, loci, ...)

SplitByChromosome(object, ...)
## S3 method for class 'RADdata'
SplitByChromosome(object, chromlist = NULL, chromlist.use.regex = FALSE,
                  fileprefix = "splitRADdata", ...)

RemoveMonomorphicLoci(object, ...)
## S3 method for class 'RADdata'
RemoveMonomorphicLoci(object, verbose = TRUE, ...)

RemoveHighDepthLoci(object, ...)
## S3 method for class 'RADdata'
RemoveHighDepthLoci(object, max.SD.above.mean = 2, verbose = TRUE, ...)

RemoveUngenotypedLoci(object, ...)
## S3 method for class 'RADdata'
RemoveUngenotypedLoci(object, removeNonvariant = TRUE, ...)
```

Arguments

object	A RADdata object.
loci	A character or numeric vector indicating which loci to include in the output RADdata object. If numeric, it refers to row numbers in object$locTable. If character, it refers to row names in object$locTable.
chromlist	An optional list indicating how chromosomes should be split into separate RADdata objects. Each item in the list is a vector of the same class as object$locTable$Chr (character or numeric) containing the names of chromosomes that should go into

one group. If not provided, each chromosome will be sent to a separate `RADdata` object.

`chromlist.use.regex`
If `TRUE`, the character strings in `chromlist` will be treated as regular expressions for searching chromosome names. For example, if one wanted all chromosomes beginning with the string "scaffold" to go into one `RADdata` object, one could include the string `"^scaffold"` as an item in `chromlist` and set `chromlist.use.regex = TRUE`. If `FALSE`, exact matches to chromosome names will be used.

`fileprefix` A character string indicating the prefix of .RData files to export.

`max.SD.above.mean`
The maximum number of standard deviations above the mean read depth that a locus can be in order to be retained.

`verbose` If `TRUE`, print out information about the original number of loci and the number of loci that were retained. For `RemoveHighDepthLoci`, a histogram is also plotted showing mean depth per locus, and the cutoff for removing loci.

`removeNonvariant`
If `TRUE`, in addition to removing loci where posterior probabilities are missing, loci will be removed where posterior probabilities are uniform across the population.

`...` Additional arguments (none implemented).

Details

`SubsetByLocus` may be useful if the user has used their own filtering criteria to determine a set of loci to retain, and wants to create a new dataset with only those loci. It can be used at any point in the analysis process.

`SplitByChromosome` is intended to make large datasets more manageable by breaking them into smaller datasets that can be processed independently, either in parallel computing jobs on a cluster, or one after another on a computer with limited RAM. Generally it should be used immediately after data import. Rather than returning new `RADdata` objects, it saves them individually to separate workspace image files, which can than be loaded one at a time to run analysis pipelines such as `IteratePopStruct`. `GetWeightedMeanGenotypes` or one of the export functions can be run on each resulting `RADdata` object, and the resulting matrices concatenated with `cbind`.

`SplitByChromosome`, `RemoveMonomorphicLoci`, and `RemoveHighDepthLoci` use `SubsetByLocus` internally.

Value

`SubsetByLocus`, `RemoveMonomorphicLoci`, `RemoveHighDepthLoci`, and `RemoveUngenotypedLoci` return a `RADdata` object with all the slots and attributes of `object`, but only containing the loci listed in `loci`, only loci with two or more alleles, only loci without abnormally high depth, or only loci where posterior probabilities are non-missing and variable, respectively.

`SplitByChromosome` returns a character vector containing file names where .RData files have been saved. Each .RData file contains one `RADdata` object named `splitRADdata`.

Author(s)

Lindsay V. Clark

See Also

VCF2RADdata, SubsetByTaxon

Examples

```
# load a dataset for this example
data(exampleRAD)
exampleRAD

# just keep the first and fourth locus
subsetRAD <- SubsetByLocus(exampleRAD, c(1, 4))
subsetRAD

# split by groups of chromosomes
exampleRAD$locTable
tf <- tempfile()
splitfiles <- SplitByChromosome(exampleRAD, list(c(1, 4), c(6, 9)),
                                fileprefix = tf)
load(splitfiles[1])
splitRADdata

# filter out monomorphic loci (none removed in example)
filterRAD <- RemoveMonomorphicLoci(exampleRAD)

# filter out high depth loci (none removed in this example)
filterRAD2 <- RemoveHighDepthLoci(filterRAD)

# filter out loci with missing or non-variable genotypes
# (none removed in this example)
filterRAD3 <- IterateHWE(filterRAD2)
filterRAD3 <- RemoveUngenotypedLoci(filterRAD3)
```

SubsetByPloidy *Create a RADdata object with a Subset of Possible Ploidies*

Description

This function is used for removing some of the ploidies (i.e. inheritance modes possible across loci) stored in a RADdata object. If genotype calling has already been performed, all of the relevant slots will be subsetted to only keep the ploidies that the user indicates.

Usage

```
SubsetByPloidy(object, ...)
## S3 method for class 'RADdata'
SubsetByPloidy(object, ploidies, ...)
```

Arguments

`object` A `RADdata` object.

`ploidies` A list, formatted like `object$possiblePloidies`, indicating ploidies to retain. Each item in the list is a vector, where `2` indicates diploid, `c(2, 2)` allotetraploid, `4` autotetraploid, etc.

`...` Other arguments (none implemented).

Details

Note that slots of `object` are subsetted but not recalculated. For example, `GetWeightedMeanGenotypes` takes a weighted mean across ploidies, which is in turn used for estimating allele frequencies and performing PCA. If the values in `object$ploidyChiSq` are considerably higher for the ploidies being removed than for the ploidies being retained, this difference is likely to be small and not substantially impact genotype calling. Otherwise, it may be advisable to re-run genotype calling after running `SubsetByPloidy`.

Value

A `RADdata` object identical to `object`, but only containing data relevant to the inheritance modes listed in `ploidies`.

Note

If you only wish to retain taxa of a certain ploidy, instead do

`object <- SubsetByTaxon(object, GetTaxaByPloidy(object, 4))`

to, for example, only retain tetraploid taxa.

Author(s)

Lindsay V. Clark

See Also

`SubsetByTaxon`, `SubsetByLocus`

Examples

```
# Example dataset assuming diploidy or autotetraploidy
data(exampleRAD)
exampleRAD <- IterateHWE(exampleRAD)
# Subset to only keep tetraploid results
exampleRAD <- SubsetByPloidy(exampleRAD, ploidies = list(4))
```

SubsetByTaxon *Create RADdata Object with a Subset of Taxa*

Description

This function is used for removing some of the taxa from a dataset stored in a RADdata object.

Usage

```
SubsetByTaxon(object, ...)
## S3 method for class 'RADdata'
SubsetByTaxon(object, taxa, ...)
```

Arguments

object	A RADdata object.
taxa	A character or numeric vector indicating which taxa to retain in the output.
...	Additional arguments (none implemented).

Details

This function may be used for subsetting a RADdata object either immediately after data import, or after additional analysis has been performed. Note however that estimation of allele frequencies, genotype prior probabilities, PCA, *etc.* are very dependent on what samples are included in the dataset. If those calculations have already been performed, the results will be transferred to the new object but not recalculated.

Value

A RADdata object containing only the taxa listed in taxa.

Author(s)

Lindsay V. Clark

See Also

SubsetByLocus

Examples

```
# load data for this example
data(exampleRAD)
exampleRAD

# just keep the first fifty taxa
subsetRAD <- SubsetByTaxon(exampleRAD, 1:50)
subsetRAD
```

TestOverdispersion *Test the Fit of Read Depth to Beta-Binomial Distribution*

Description

This function is intended to help the user select a value to pass to the `overdispersion` argument of `AddGenotypeLikelihood`, generally via pipeline functions such as `IterateHWE` or `PipelineMapping2Parents`.

Usage

```
TestOverdispersion(object, ...)

## S3 method for class 'RADdata'
TestOverdispersion(object, to_test = seq(6, 20, by = 2), ...)
```

Arguments

`object`	A `RADdata` object. Genotype calling does not need to have been performed, although for mapping populations it might be helpful to have done a preliminary run of `PipelineMapping2Parents` without linkage.
`to_test`	A vector containing values to test. These are values that will potentially be used for the `overdispersion` argument of a pipeline function. They should all be positive numbers.
`...`	Additional arguments (none implemented).

Details

If no genotype calling has been performed, a single iteration under HWE using default parameters will be done. `object$ploidyChiSq` is then examined to determine the most common/most likely inheritance mode for the whole dataset. The alleles that are examined are only those where this inheritance mode has the lowest chi-squared value.

Within this inheritance mode and allele set, genotypes are selected where the posterior probability of having a single copy of the allele is at least 0.95. Read depth for these genotypes is then analyzed. For each genotype, a two-tailed probability is calculated for the read depth ratio to deviate from the expected ratio by at least that much under the beta-binomial distribution. This test is performed for each overdispersion value provided in `to_test`.

Value

A list of the same length as `to_test` plus one. The names of the list are `to_test` converted to a character vector. Each item in the list is a vector of p-values, one per examined genotype, of the read depth ratio for that genotype to deviate that much from the expected ratio. The last item, named "optimal", is a single number indicating the optimal value for the overdispersion parameter based on the p-value distributions. If the optimal value was the minimum or maximum tested, `NA` is returned in the `"optimal"` slot to encourage the user to test other values.

Author(s)

Lindsay V. Clark

Examples

```
# dataset with overdispersion
data(Msi01genes)

# test several values for the overdispersion parameter
myP <- TestOverdispersion(Msi01genes, to_test = 8:10)

# view results as quantiles
sapply(myP[names(myP) != "optimal"],
       quantile, probs = c(0.01, 0.25, 0.5, 0.75, 0.99))
```

VCF2RADdata *Create a RADdata Object from a VCF File*

Description

This function reads a Variant Call Format (VCF) file containing allelic read depth and SNP alignment positions, such as can be produced by TASSEL or GATK, and generates a RADdata dataset to be used for genotype calling in **polyRAD**.

Usage

```
VCF2RADdata(file, phaseSNPs = TRUE, tagsize = 80, refgenome = NULL,
            tol = 0.01, al.depth.field = "AD", min.ind.with.reads = 200,
            min.ind.with.minor.allele = 10, possiblePloidies = list(2),
            taxaPloidy = 2L, contamRate = 0.001,
        samples = VariantAnnotation::samples(VariantAnnotation::scanVcfHeader(file)),
            svparam = VariantAnnotation::ScanVcfParam(fixed = "ALT", info = NA,
                                                      geno = al.depth.field,
                                                      samples = samples),
            yieldSize = 5000, expectedAlleles = 5e+05, expectedLoci = 1e+05,
            maxLoci = NA)
```

Arguments

file The path to a VCF file to be read. This can be uncompressed, bgzipped using Samtools or Bioconductor, or a TabixFile object from Bioconductor.

phaseSNPs If TRUE, markers that appear to have come from the same set of reads will be phased and grouped into haplotypes. Otherwise, each row of the file will be kept as a distinct marker.

tagsize The read length, minus any barcode sequence, that was used for genotyping. In TASSEL, this is the same as the kmerLength option. This argument is used for grouping SNPs into haplotypes and is ignored if phaseSNPs = FALSE.

`refgenome` Optional. The name of a FASTA file, or an `FaFile` object, containing the reference genome. When grouping SNPs into haplotypes, if provided this reference genome is used to insert non-variable nucleotides between the variable nucleotides in the `alleleNucleotides` slot of the `RADdata` output. Ignored if `phaseSNPs = FALSE`. Useful if exact SNP positions need to be retained for downstream analysis after genotype calling in **polyRAD**. In particular this argument is necessary if you plan to export genotype calls back to VCF.

`tol` The proportion by which two SNPs can differ in read depth and still be merged into one group for phasing. Ignored if `phaseSNPs = FALSE`.

`al.depth.field` The name of the genotype field in the VCF file that contains read depth at each allele. This should be "AD" unless your format is very unusual.

`min.ind.with.reads`
Integer used for filtering SNPs. To be retained, a SNP must have at least this many samples with reads.

`min.ind.with.minor.allele`
Integer used for filtering SNPs. To be retained, a SNP must have at least this many samples with the minor allele. When there are more than two alleles, at least two alleles must have at least this many samples with reads for the SNP to be retained.

`possiblePloidies`
A list indicating inheritance modes that might be encountered in the dataset. See `RADdata`.

`taxaPloidy` A single integer, or an integer vector with one value per taxon, indicating ploidy. See `RADdata`.

`contamRate` A number indicating the expected sample cross-contamination rate. See `RADdata`.

`samples` A character vector containing the names of samples from the file to export to the `RADdata` object. The default is all samples. If a subset is provided, filtering with `min.ind.with.reads` and `min.ind.with.minor.allele` is performed within that subset. Ignored if a different `samples` argument is provided within `svparam`.

`svparam` A `ScanVcfParam` object to be used with `readVcf`. The primary reasons to change this from the default would be 1) if you want additional FIXED or INFO fields from the file to be exported to the `locTable` slot of the `RADdata` object, and/or 2) if you only want to import particular regions of the genome, as specified with the `which` argument of `ScanVcfParam`.

`yieldSize` An integer indicating the number of lines of the file to read at once. Increasing this number will make the function faster but consume more RAM.

`expectedAlleles`
An integer indicating the approximate number of alleles that are expected to be imported after filtering and phasing. If this number is too low, the function may slow down considerably. Increasing this number increases the amount of RAM used by the function.

`expectedLoci` An integer indicating the approximate number of loci that are expected to be imported after filtering and phasing. If this number is too low, the function may slow down considerably. Increasing this number increases the amount of RAM used by the function.

maxLoci An integer indicating the approximate maximum number of loci to return. If provided, the function will stop reading the file once it has found at least this many loci that pass filtering and phasing. This argument is intended to be used for generating small RADdata objects for testing purposes, and should be left NA under normal circumstances.

Details

This function requires the BioConductor package **VariantAnnotation**. See https://bioconductor.org/packages/release/bioc/html/VariantAnnotation.html for installation instructions.

If you anticipate running VCF2RADdata on the same file more than once, it is recommended to run bgzip and indexTabix from the package **Rsamtools** once before running VCF2RADdata. See examples. If the reference genome is large enough to require a .csi index rather than a .tbi index, after bgzipping the file you can generate the index from the bash terminal using tabix --csi from Samtools.

min.ind.with.minor.allele is used for filtering SNPs as the VCF file is read. Additionally, because phasing SNPs into haplotypes can cause some haplotypes to fail to pass this threshold, VCF2RADdata internally runs MergeRareHaplotypes with min.ind.with.haplotype = min.ind.with.minor.allele, then RemoveMonomorphicLoci, before returning the final RADdata object.

Value

A RADdata object.

Note

In the python directory of the **polyRAD** installation, there is a script called tassel_vcf_tags.py that can identify the full tag sequence(s) for every allele imported by VCF2RADdata.

Author(s)

Lindsay V. Clark

References

Variant Call Format specification: http://samtools.github.io/hts-specs/

TASSEL GBSv2 pipeline: https://bitbucket.org/tasseladmin/tassel-5-source/wiki/Tassel5GBSv2Pipeline

GATK: https://gatk.broadinstitute.org/hc/en-us

Tassel4-Poly: https://github.com/guilherme-pereira/tassel4-poly

See Also

MakeTasselVcfFilter for filtering to a smaller VCF file before reading with VCF2RADdata.

To export to VCF: RADdata2VCF

Other data import functions: readStacks, readHMC, readTagDigger, readTASSELGBSv2, readProcessIsoloci, readDArTag

Examples

```
# get the example VCF installed with polyRAD
exampleVCF <- system.file(”extdata”, ”Msi01genes.vcf”, package = ”polyRAD”)


# loading VariantAnnotation namespace takes >10s,
# so is excluded from CRAN checks

require(VariantAnnotation)

# Compress and index the VCF before reading, if not already done
if(!file.exists(paste(exampleVCF, ”bgz”, sep = ”.”))){
  vcfBG <- bgzip(exampleVCF)
  indexTabix(vcfBG, ”vcf”)
}

# Read into RADdata object
myRAD <- VCF2RADdata(exampleVCF, expectedLoci = 100, expectedAlleles = 500)

# Example of subsetting by genomic region (first 200 kb on Chr01)
mysv <- ScanVcfParam(fixed = ”ALT”, info = NA, geno = ”AD”,
                     samples = samples(scanVcfHeader(exampleVCF)),
                     which = GRanges(”01”, IRanges(1, 200000)))
myRAD2 <- VCF2RADdata(exampleVCF, expectedLoci = 100, expectedAlleles = 500,
                      svparam = mysv, yieldSize = NA_integer_)
```

Index