Package 'PolyPatEx'

October 12, 2022

Type Package

Title Paternity Exclusion in Autopolyploid Species

Version 0.9.2

Date 2016-04-11

Author Alexander Zwart

Maintainer Alexander Zwart <Alec.Zwart@csiro.au>

Description Functions to perform paternity exclusion via allele matching, in autopolyploid species having ploidy 4, 6, or 8. The marker data used can be genotype data (copy numbers known) or 'allelic phenotype data' (copy numbers not known).

License GPL-3 + file LICENSE

LazyLoad yes

LazyData yes

Imports gtools, utils

RoxygenNote 5.0.1.9000

NeedsCompilation no

Repository CRAN

Date/Publication 2016-04-11 09:06:29

R topics documented:

`PolyPatEx-package` *Polyploid paternity exclusion by allele matching.*

Description

PolyPatEx provides functions to perform paternity exclusion analysis in autopolyploid species having ploidy 4, 6, or 8. The package requires codominant marker data from two or more loci in a monoecious or dioecious species. The marker data can be 'genotypic' (copy numbers known) or 'phenotypic' (copy numbers not known). PolyPatEx can also perform exclusion on diploid (ploidy 2) *genotype* data.

Details

Routines are provided to compare each progeny with its mother, and then with the candidate fathers, to determine which candidates are indeed capable of being fathers, on the basis of the allele sets they display at each locus.

PolyPatEx addresses the question - at a given locus, can the candidate father provide a viable gamete given its allele set, and given the possible paternal gametes indicated by the progeny's and mother's allele sets?

Note that PolyPatEx does not implement a probabilistic solution to the exclusion problem, merely a simple comparative analysis based on available alleles and their multiplicities. Also note that PolyPatEx is not optimised for very large marker datasets such as SNP datasets, instead is suited to low density, high information markers such as SSRs.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

`convertToPhenot` *Convert a genotype allele dataset to a phenotype dataset*

Description

Convert a 'genotypic' dataset (marker dosages known) to a 'phenotypic' dataset (marker dosages not known, unique alleles appear only once in each allele set).

Usage

```
convertToPhenot(adata)
```

Arguments

adata	data frame: a genotypic allele dataset.

Details

In the terminology used by the PolyPatEx package, a 'genotypic' allele dataset is one where marker dosages are known, hence any locus at which fewer than p (the ploidy) alleles are detected is incomplete (and subsequently ignored by the genotype-specific routines in this package). A 'phenotypic' allele dataset is one where marker dosages are not known, hence individual alleles appear only once in an allele set, and a complete allele set can contain between 1 and p alleles.

convertToPhenot converts a genotypic dataset to a phenotypic dataset, simply by removing any allele duplicates from each allele set. This is probably not something many will want to do, since one loses considerable information in the process...

Value

A data frame, containing the phenotypic form of the original genotypic dataset.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Using the example dataset 'FR_Genotype':
data(FR_Genotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
gData <- preprocessData(FR_Genotype,
                        numLoci=7,
                        ploidy=4,
                        dataType="genotype",
                        dioecious=TRUE,
                        mothersOnly=TRUE)

head(gData)  ## Checked and Cleaned version of FR_Genotype

pData <- convertToPhenot(gData)

head(pData)
```

`fixCSV` *Tidy a comma separated value (CSV) file*

Description

Tidies up a Comma Separated Value (CSV) file, ensuring that each row of the table in the file contains the same number of commas, and no empty rows are left below the table.

Usage

```
fixCSV(file, skip = 0, overwrite = FALSE)
```

Arguments

`file`	character: the name of the CSV file to be 'fixed'.
`skip`	integer: the number of lines in the CSV file to skip before the header row of the table. The skipped lines are copied directly to the output file unchanged. The default is `skip=0`, implying that the header row is the first row of the CSV file.
`overwrite`	logical: Write output to a separate, 'FIXED' file (`overwrite=FALSE`, the default), or overwrite the original file (`overwrite=TRUE`)? If `overwrite=TRUE`, the original file is copied to a `.BAK` file before being overwritten.

Details

`fixCSV` tidies up a Comma Separated Value (CSV) file to ensure that the CSV file contains a strictly rectangular block of data for input into R (ignoring any preliminary comment rows via the `skip=` argument).

CSV formatted files are a plain text file format for tabular data, in which cell entries in the same row of a table are separated by commas. When such files are exported from other applications such as spreadsheet software, the software has to decide whether any empty cells to the right-hand side of, or below, the table or spreadsheet should be represented by trailing commas in the CSV file. Such decisions can result in a 'ragged' table in the CSV file, in which some rows contain fewer commas ('short rows') or more commas ('long rows') than others, or where empty rows below the table are included as comma-only rows in the CSV file.

While R's `read.table` and related functions can sensibly extend short rows as needed, ragged tables in a CSV file can still result in errors, unwanted empty rows (below the table) or unwanted columns (to the right of the table) when the data is loaded into R.

`fixCSV` reads in a specified CSV file and removes or adds commas to rows, to ensure that each row in the body of the table contains the same number of cells as the header row of the table. Any empty rows below the table are also removed. The resulting table is then written back to file, either to a new file with 'FIXED' added to the filename (argument `overwrite=FALSE`, the default) or overwriting the original file (`overwrite=TRUE` - the original file is copied to a `.BAK` file before being overwritten).

Note that:

- The table of data in the CSV file *must* contain a header row of the correct length, since this row is used to determine the correct number of columns for the table. Note: if this header row is too short, then subsequent rows will be truncated to match the length of the header, so beware. Misspecification of the `skip=` argument (see below) can similarly lead to such corruption of the 'fixed' file.
- In the header row, any trailing commas representing empty cells to the right of the (non-empty) header entries are first removed before determining the correct number of columns for the table. Thus the length of the header row (and hence the assumed width of the entire table) is determined by the *right-most non-empty cell* in the header row.
- `fixCSV` does not remove empty cells, rows or columns within the interior (or on the left side) of the table - it is concerned only with the right and bottom boundaries of the table.
- A `skip=` argument is included to tell `fixCSV` to ignore the specified number of comment rows preceding the header row. Such rows are simply copied over into the output file unchanged. The default for this parameter is `skip=0`, so that the first row in the data file is assumed to be the header row. As noted above, misspecification of this argument can seriously corrupt the output.
- `fixCSV` can overwrite your data file(s) (via `overwrite=TRUE`), and althought it makes a backup of your original file, you should still make sure that you have a separate backup of your data file in a safe place before using this function! The author of this code takes no responsibility for any data loss or corruption as a result of the use of this routine...

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Not run:

## Assuming CSV file 'alleleDataFile.csv' exists in the current
## directory.  The following overwrites the CSV file - make sure
## you have a backup!

fixCSV("alleleDataFile.csv",overwrite=TRUE)


## End(Not run)
```

`foreignAlleles` *Identify foreign alleles*

Description

Identify 'foreign' alleles (alleles present in progeny, but not in parents or other non-parental adults). Note that `foreignAlleles` does *not* distinguish between populations as indicated by the `popn` column of the allele data frame.

Usage

```
foreignAlleles(adata)
```

Arguments

adata	data frame: the preprocessed allele data set returned by either inputData or preprocessData.

Value

A list, containing two data frames. Data frame byLocus summarises the foreign alleles found at each locus. Data frame byProgenyLocus summarises the same alleles by progeny and locus. In this latter data frame, the code P.missing indicates no alleles were present in the progeny at this locus.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Using the example dataset 'FR_Genotype':
data(FR_Genotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
gData <- preprocessData(FR_Genotype,
                        numLoci=7,
                        ploidy=4,
                        dataType="genotype",
                        dioecious=TRUE,
                        mothersOnly=TRUE)

head(gData)  ## Checked and Cleaned version of FR_Genotype

fAlleles <- foreignAlleles(gData)

## View foreign alleles detected at each locus:
fAlleles$byLocus

## View foreign alleles detected in each progeny, at each locus:
fAlleles$byProgenyLocus

## Both of these objects are data frames, hence can be written to file
## via, e.g., write.csv().
```

FR_Genotype *Example genotype allele dataset*

Description

Example genotype allele dataset - a dioecious tetraploid species, seven loci observed.

Details

The dataset is available in two forms - as a compressed data file which can be loaded easily into R via the R `data` function, i.e., `data(FR_Genotype)`, and as a CSV (Comma-Separated-Value, a plain text format) file, to provide an example of the required CSV format.

Note that a technicality of R's package building process requires the use of `data` to load the data in the reference help examples, whereas the user would generally invoke the `inputData` function to load their own data from file. An example of the latter is demonstrated in the example section on this page, but is not run.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Not run:
## To locate this dataset in your filesystem, use:

gDataFile <- system.file("extdata/FR_Genotype.csv",
                         package="PolyPatEx")
print(gDataFile)

## To load this file using PolyPatEx's 'inputData' function, use:

gData <- inputData(gDataFile,
                   numLoci=7,
                   ploidy=4,
                   dataType="genotype",
                   dioecious=TRUE,
                   mothersOnly=TRUE)

## ...or use 'mothersOnly=FALSE' if you wish to retain
## non-maternal females in the dataset.

## gData now contains the checked and preprocessed allele dataset,
## ready to be passed to other PolyPatEx analysis functions.

## End(Not run)
```

genotPPE *Simple paternity exclusion for genotype allele data*

Description

Conduct a paternity exclusion analysis on a genotype dataset.

Usage

```
genotPPE(adata)
```

Arguments

adata	data frame: the preprocessed allele data set returned by either inputData or preprocessData.

Details

genotPPE conducts a basic paternity exclusion analysis on a genotype dataset.

For the purposes of the PolyPatEx package, the term 'genotype' refers to forms of marker data where the allele copy numbers (or multiplicities) are known - hence for a polyploid species of ploidy p, there should be exactly p alleles detected at each locus, some of which may be repeats of the same allele state. In PolyPatEx, no allowance is made for undetected alleles in genotype data - allele sets having fewer than p alleles present should have been reset to contain no alleles by preprocessData.

For the above and other reasons, genotPPE should **NOT** be applied to a dataset that has not been preprocessed by preprocessData (either by calling directly preprocessData on the data frame directly, or by using inputData to load the data from file).

The genotype-based paternity analysis is based on simple comparison of genotype allele sets between mother, progeny, and candidate father. The mother-progeny relationship is assumed to be known. For genotype data, should a progeny contain only alleles also present in its mother, then a potential father is any candidate that can provide a gamete compatible with the progeny's genotype, given the mother's genotype.

Value

A list whose components are described below. The components that are probably of primary interest to the user are adultTables$FLCount and adultTables$VLTotal. These are likely to be large tables, so note that the functions potentialFatherCounts and potentialFatherIDs are available to usefully summarise their content.

The list returned by genotPPE contains two elements, progenyTables and adultTables, each of which are themselves lists.

Element adultTables contains the following components:

FLCount Father Loci Count - a matrix, showing for each progeny-candidate combination, the number of loci at which the candidate matches (i.e., could have fathered) the progeny

`VLTotal` Valid Loci Total - a matrix, showing for each progeny-candidate combination, the total number of loci at which a valid comparison between progeny and candidate could be made. (Missing allele sets, whether in the original data, or due to progeny-mother mismatches found by `preprocessData`, can result in fewer loci at which progeny-candidate (father) comparisons are possible.)

`fatherSummaryTable` A matrix, combining the results of `FLCount` and `VLTotal` (see above) for each progeny-candidate combination in one table. This is purely for ease of viewing purposes, but note also the functions `potentialFatherCounts` and `potentialFatherIDs` which may provide more useful summary output.

`CPNotM.alleleArray` A 3D array containing the alleles present in both candidate (father) and progeny, but not in the progeny's mother (for each progeny/candidate/locus combination)

`CMP.alleleArray` A 3D array containing the alleles present in candidate, progeny and progeny's mother (for each progeny/candidate/locus combination)

`simpleFatherArray` A 3D array indicating whether each candidate is compatible with each progeny, for each locus

`progenyTables` contains the following components:

`progenyStatusTable` A data frame, indicating the status of the progeny / mother allele set comparison (for each progeny, at each locus).

`MP.alleleTable` A data frame containing the alleles that are found in both mother's and progeny's allele sets (for each progeny, at each locus)

`PNotM.alleleTable` A data frame, containing the alleles in the progeny's allele set, that are *not* present in the mother's allele set(for each progeny, at each locus)

The status codes in `progenyTables$progenyStatusTable` are:

`"MAO"` Mother Alleles Only - the progeny contains only alleles found also in the mother

`"NMA"` Non-Mother Alleles - the progeny contains alleles that are not found in the mother

`"P.missing"` No comparison was possible at this locus because the progeny's allele set was missing

`"P.missing"` No comparison was possible at this locus because the mother's allele set was missing

`"PM.missing"` No comparison was possible at this locus because both progeny's and mother's allele sets were missing

Note that some of the `"P.missing"` or `"PM.missing"` codes may have arisen due to progeny / mother mismatches found (and corresponding progeny allele sets removed) by `preprocessData`.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

See Also

`phenotPPE`

Examples

```
## Using the example dataset 'FR_Genotype':
data(FR_Genotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
gData <- preprocessData(FR_Genotype,
                        numLoci=7,
                        ploidy=4,
                        dataType="genotype",
                        dioecious=TRUE,
                        mothersOnly=TRUE)

head(gData)  ## Checked and Cleaned version of FR_Genotype

gPPE <- genotPPE(gData)  ## Perform the exclusion analyses

## gPPE is a large (and rather ugly!) data structure - see
## functions potentialFatherCounts() and potentialFatherIDs() for
## more useful output from the gPPE object.
```

GF_Phenotype *Example phenotype allele dataset*

Description

Example 'phenotype' allele dataset - a monoecious non-selfing hexaploid species, seven loci observed.

Details

The dataset is available in two forms - as a compressed data file which can be loaded easily into R via the R `data` function, i.e., `data(FR_Genotype)`, and as a CSV (Comma-Separated-Value, a plain text format) file, to provide an example of the required CSV format.

Note that a technicality of R's package building process requires the use of `data` to load the data in the reference help examples, whereas the user would generally invoke the `inputData` function to load their own data from file. An example of the latter is demonstrated in the example section on this page, but is not run.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Not run:
## To locate this dataset in your filesystem, use:

pDataFile <- system.file("extdata/GF_Phenotype.csv",
                         package="PolyPatEx")
print(pDataFile)

## To load this file using PolyPatEx's 'inputData' function use:

pData <- inputData(pDataFile,
                   numLoci=7,
                   ploidy=6,
                   dataType="phenotype",
                   dioecious=FALSE,
                   selfCompatible=FALSE)

## pData now contains the checked and preprocessed allele dataset,
## ready to be passed to other PolyPatEx analysis functions.


## End(Not run)
```

`inputData` *Read in, check and preprocess the allele dataset*

Description

Read in an allele dataset from file, and return a checked and preprocessed data frame.

Usage

```
inputData(file, numLoci, ploidy, dataType, dioecious, selfCompatible = NULL,
  mothersOnly = NULL, lociMin = 1, matMismatches = 0, skip = 0)
```

Arguments

`file`	character: the name of the allele data file.
`numLoci`	integer: the number of loci in the allele dataset.
`ploidy`	integer: the species' ploidy, one of `2`, `4`, `6`, or `8`.
`dataType`	character: either `"genotype"` or `"phenotype"`.
`dioecious`	logical: is the species dioecious (`TRUE`) or monoecious (`FALSE`)?
`selfCompatible`	logical: In monoecious species (`dioecious=FALSE`), can individuals self-fertilise? When `dioecious=FALSE`, this argument may be left at its default value of NULL - it will be set to `FALSE` by `preprocessData`.

mothersOnly	logical: in dioecious species, should females without progeny present be removed from the dataset? If dioecious=TRUE, then mothersOnly must be set to either TRUE or FALSE. If dioecious=FALSE, argument mothersOnly should be left at its default value of NULL.
lociMin	integer: the minimum number of loci in an individual that must have alleles present for the individual (and its progeny, if the individual is a mother) to be retained in the dataset. See the help for preprocessData for more on this parameter.
matMismatches	an integer between 0 and numLoci-1, being the maximum number of mismatching loci between mother and offspring that are allowed before the offspring is removed from the dataset. The default value is 0. If an offspring has fewer than matMismatches loci that mismatch with its mother, the offending loci are set to contain no alleles.
skip	integer: the number of lines in the CSV to skip before the header row of the table.

Details

inputData reads in an allele dataset from the specified file, then calls preprocessData to perform a series of data format checks and preprocessing steps before returning the checked and preprocessed dataset as an R data frame. The reference information for preprocessData contains further information on the checks and preprocessing - it is strongly recommended you read that information in addition to the information below.

The use of inputData is optional, if you wish to create or load the allele dataset into R by other means. However, it is then necessary to call preprocessData on the data frame prior to using any other analysis functions in this package. Similarly, if you decide to change or manipulate the data frame contents within R, you should call preprocessData again on the data frame prior to using any of the **PolyPatEx** analysis functions. See the help for preprocessData for further details.

Note that inputData strips leading or trailing spaces (whitespace) from each entry in the allele dataset as it is read in. If you load your data by a means other than inputData, you should ensure that you perform this step yourself, as preprocessData will not carry out this necessary step.

Note also that you should not use spaces in any of your allele codes - PolyPatEx functions use spaces to separate allele codes as they process the data - if allele codes already contains spaces, errors will occur in this processing. If you need a separator, I recommend using either 'code.' (a period) or 'code_' (an underscore) rather than a space.

Neither inputData (nor preprocessData) will alter the CSV file from which the data is loaded - they merely return a checked and preprocessed version of your allele dataset (in the form of an R data frame) within the R environment, ready for use by other **PolyPatEx** functions.

To load the allele dataset into R, inputData calls R's read.csv function with certain arguments specified. These arguments make read.csv more stringent about the precise format of the input datafile, requiring in particular that each row of the CSV-formatted data file contain the correct number of commas. This is not always guaranteed when the CSV file has been exported from spreadsheet software. Should you get 'Error in scan' messages complaining about the number of elements in a line of the input file, consider calling fixCSV on the data file, before calling inputData again. fixCSV attempts to find and correct such errors in a CSV file - see the help for this function. Note that if you specify the skip parameter in a call to fixCSV, you should use the same value for this parameter in inputData to avoid an error.

The various **PolyPatEx** functions need to know the characteristics of the dataset being analysed - these are specified in the `inputData` or `preprocessData` calls and are invisibly attached to the allele data frame that is returned, for use by other **PolyPatEx** functions. The required characteristics are:

- `numLoci`: the number of loci in the dataset.
- `ploidy`: the ploidy p of the species (currently allowed to be 4, 6, or 8. `ploidy` can also be 2, provided `dataType="genotype"`).
- `dataType`: whether the data is genotypic (all p alleles at each locus are observed) or phenotypic (only the distinct allele states at a locus are observed - alleles that appear more than once in the genotype of a locus only appear once in the phenotype).
- `dioecious`: whether the species is dioecious or monoecious.
- `selfCompatible` whether a monoecious species is self compatible (i.e., whether an individual can fertilise itself).
- `mothersOnly`: whether a dioecious dataset should retain only adult females that are mothers of progeny in the dataset. If `dioecious=TRUE`, then `mothersOnly` must be set to either `TRUE` or `FALSE`.

Value

A data frame, containing the checked and pre-processed allele data, ready for further analysis by other **PolyPatEx** functions. All columns in the output data frame will be of mode `character`.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Not run:

## Obtain path to the example genotype data file
## 'FR_Genotype.csv'
gDataFile <- system.file("extdata/FR_Genotype.csv",
                         package="PolyPatEx")
print(gDataFile)

gData <- inputData(gDataFile,
                   numLoci=7,
                   ploidy=4,
                   dataType="genotype",
                   dioecious=TRUE,
                   mothersOnly=TRUE)

## ...or use 'mothersOnly=FALSE' if you wish to retain
## non-maternal females in the dataset.

## gData now contains the checked and preprocessed allele dataset,
## ready to be passed to other PolyPatEx analysis functions.
```

```
## In your own workflow, you would typically specify the path to
## your allele dataset directly - e.g. if the dataset
## myAlleleData.csv is on the Data subdirectory of the current R
## working directory (see R function setwd()), then:
##
## gData <- inputData("Data/myAlleleData.csv",
##                    numLoci= etc etc etc...,


## End(Not run)
```

`multilocusTypes` *Genotype summaries*

Description

Return summaries of individual- and multi- locus genotypes for adults and progeny.

Usage

```
multilocusTypes(adata)
```

Arguments

`adata`	data frame: the checked and preprocessed dataset returned by `inputData`.

Details

Function `multilocusTypes` summarises the different genotypes present at each locus in the dataset (separately for progeny and adults), and across the loci (again, separately for progeny and adults). `multilocusTypes` returns a list structure with several elements.

Value

A list structure, with the following components:

`uniqueProgenyTypes` A data frame containing, for each locus, the distinct genotypes that are present in the progeny in the dataset, and the numbers of progeny containing each genotype at that locus.

`numUniqueProgenyTypes` The number of unique genotypes at each locus in the progeny in the dataset.

`uniqueAdultTypes` A data frame containing, for each locus, the distinct genotypes that are present in the adults in the dataset, and the numbers of adults containing each genotype at that locus.

`numUniqueAdultTypes` The number of unique genotypes at each locus in the adult set.

`uniqueProgenyMLTypes` A data frame containing the distinct genotypes *across all loci* that are present in the progeny in the dataset, and the numbers of progeny containing each multilocus genotype.

numUniqueProgenyMLTypes The total number of progeny multilocus genotypes.

uniqueAdultMLTypes A data frame containing, the distinct genotypes *across all loci* that are present in the adults in the dataset, and the numbers of adults containing each multilocus genotype.

numUniqueAdultMLTypes The total number of adult multilocus genotypes.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Using the example dataset 'FR_Genotype':
data(FR_Genotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
gData <- preprocessData(FR_Genotype,
                        numLoci=7,
                        ploidy=4,
                        dataType="genotype",
                        dioecious=TRUE,
                        mothersOnly=TRUE)

head(gData)  ## Checked and Cleaned version of FR_Genotype

mTypes <- multilocusTypes(gData)

## mTypes is a list structure - individual components can be
## printed to the screen, or saved to file via, e.g. read.csv().

mTypes$numUniqueProgenyTypes

## Components of mTypes
names(mTypes)
```

---

phenotPPE *Simple paternity exclusion for phenotype allele data*

Description

Conduct a paternity exclusion analysis on a phenotype dataset.

Usage

```
phenotPPE(adata)
```

Arguments

adata	data frame: the preprocessed allele data set returned by either inputData or preprocessData.

Details

phenotPPE conducts a basic paternity exclusion analysis on a 'phenotype' dataset.

For the purposes of the PolyPatEx package, the term 'phenotype' refers to forms of marker data where the allele dosages (or multiplicities) are not known - hence for a polyploid species of ploidy p, a valid allele set may contain between one and p alleles, which should all be distinct. Any cases of allele sets having duplicated alleles will have previously been caught by preprocessData (automatically called by inputData) and will have produced an error message, requiring the user to remove any duplicated alleles (within allele sets) in the original data file.

For the above and other reasons, phenotPPE should **NOT** be applied to a dataset that has not been preprocessed by preprocessData (either by calling preprocessData on the allele data frame directly, or by loading the allele data into R using inputData).

Value

A list whose components are described below. The components that are probably of primary interest to the user are adultTables$FLCount and adultTables$VLTotal. These are likely to be large tables, so note that the functions potentialFatherCounts and potentialFatherIDs are available to usefully summarise their content.

The list returned by phenotPPE contains two elements, progenyTables and adultTables, each of which are themselves lists.

adultTables contains the following components:

FLCount Father Loci Count - a matrix, showing for each progeny-candidate combination, the number of loci at which the candidate matches (i.e., could have fathered) the progeny

VLTotal Valid Loci Total - a matrix, showing for each progeny-candidate combination, the total number of loci at which a valid comparison between progeny and candidate could be made. (Missing allele sets, whether in the original data, or due to progeny-mother mismatches found by preprocessData can result in fewer loci at which progeny-candidate (father) comparisons are possible.)

fatherSummaryTable A matrix, combining the results of FLCount and VLTotal (see above) for each progeny-candidate combination in one table. This is purely for ease of viewing purposes, but note also the functions potentialFatherCounts and potentialFatherIDs which may provide more useful summary output.

CPNotM.alleleArray A 3D array containing the alleles present in both candidate (father) and progeny, but not in the progeny's mother (for each progeny/candidate/locus combination)

CMP.alleleArray A 3D array containing the alleles present in candidate, progeny and progeny's mother (for each progeny/candidate/locus combination)

simpleFatherArray A 3D array indicating whether each candidate is compatible with each progeny, for each locus

progenyTables contains the following components:

`progenyStatusTable` A data frame, indicating the status of the progeny / mother allele set comparison (for each progeny, at each locus).

`MP.alleleTable` A data frame containing the alleles that are found in both mother's and progeny's allele sets (for each progeny, at each locus)

`PNotM.alleleTable` A data frame, containing the alleles in the progeny's allele set, that are *not* present in the mother's allele set(for each progeny, at each locus)

The status codes in `progenyTables$progenyStatusTable` are:

`"MAO"` Mother Alleles Only - the progeny contains only alleles found also in the mother

`"NMA"` Non-Mother Alleles - the progeny contains alleles that are not found in the mother

`"P.missing"` No comparison was possible at this locus because the progeny's allele set was missing

`"P.missing"` No comparison was possible at this locus because the mother's allele set was missing

`"PM.missing"` No comparison was possible at this locus because both progeny's and mother's allele sets were missing

Note that some of the `"P.missing"` or `"PM.missing"` codes may have arisen due to progeny / mother mismatches found (and corresponding progeny allele sets removed) by `preprocessData`.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

See Also

`genotPPE`

Examples

```
## Using the example dataset 'GF_Phenotype':
data(GF_Phenotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
pData <- preprocessData(GF_Phenotype,
                        numLoci=7,
                        ploidy=6,
                        dataType="phenotype",
                        dioecious=FALSE,
                        selfCompatible=FALSE)

pPPE <- phenotPPE(pData)

## pPPE is a large (and rather ugly!) data structure - see
## functions potentialFatherCounts() and potentialFatherIDs() for
## more useful output from the gPPE object.
```

`potentialFatherCounts` *Count potential fathers*

Description

Count the number of potential fathers detected for each progeny.

Usage

```
potentialFatherCounts(dataset, mismatches = 0, VLTMin = 1)
```

Arguments

`dataset`	list: a list structure previously output from `genotPPE` or `phenotPPE`.
`mismatches`	integer: the maximum allowed number of mismatching loci between candidate and progeny, before the candidate is rejected as a potential father.
`VLTMin`	integer: the minimum number of 'valid' loci (loci at which a valid progeny-candidate comparison was possible) required for a candidate to be considered as a potential father.

Details

Given the output from `genotPPE` or `phenotPPE`, `potentialFatherCounts` returns, for each progeny, the number of candidates that are identified as potential fathers.

To decide whether a given candidate is a potential father to a given progeny, `potentialFatherCounts` uses the quantities FLCount (the number of loci at which a candidate can provide a gamete compatible with the progeny) and VLTotal (the number of loci at which a valid comparison was possible - 'valid' loci) that are returned by `genotPPE` or `phenotPPE`.

For a candidate to be identified as a potential father of a progeny, there are two criteria to be met:

1. `VLTotal >= max(VLTMin,mismatches+1)`,
2. `FLCount >= VLTotal-mismatches`.

Here, `VLTmin` and `mismatches` are user-specified parameters. `VLTmin` allows the user to ensure that a candidate is only considered for potential fatherhood if a sufficient number of valid loci were available for comparison. `mismatches` allows the user to specify a maximum number of allowed mismatching loci between progeny and candidate, before the candidate is rejected as a potential father. Hence the user may wish to relax the condition that ALL valid loci must match for a candidate to be regarded as a potential father to a progeny.

Value

A data frame, containing columns `Progeny` (progeny id), `Mother` (id of the progeny's mother) and `potentialFatherCount` (the number of potential fathers found for the given progeny, given the criteria described above).

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Using the example dataset 'FR_Genotype':
data(FR_Genotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
gData <- preprocessData(FR_Genotype,
                        numLoci=7,
                        ploidy=4,
                        dataType="genotype",
                        dioecious=TRUE,
                        mothersOnly=TRUE)

head(gData)  ## Checked and Cleaned version of FR_Genotype

gPPE <- genotPPE(gData)  ## Perform the exclusion analyses

## Obtain counts of potential fathers of each seedling, allowing a
## single allele mismatch:
pFC <- potentialFatherCounts(gPPE,mismatches=1,VLTMin=2)

## pFC can be viewed or written to file via, e.g. write.csv()
```

---

potentialFatherIDs *Identify potential fathers*

Description

Identify the potential fathers for each progeny.

Usage

```
potentialFatherIDs(dataset, mismatches = 0, VLTMin = 1)
```

Arguments

dataset	list: a list structure previously output from genotPPE or phenotPPE.
mismatches	integer: the maximum allowed number of mismatching loci between candidate and progeny, before the candidate is rejected as a potential father. Default value is 0 - i.e., no mismatches allowed.
VLTMin	integer: the minimum number of 'valid' loci (loci at which a valid progeny-candidate comparison was possible) required for a candidate to be considered as a potential father. Default value is 1.

Details

Given the output from genotPPE or phenotPPE, potentialFatherIDs returns, for each progeny, the IDs of candidates that are identified as potential fathers.

To decide whether a given candidate is a potential father to a given progeny, potentialFatherIDs uses the quantities FLCount (the number of loci at which a candidate can provide a gamete compatible with the progeny) and VLTotal (the number of loci at which a valid comparison was possible - 'valid' loci) that are returned by genotPPE or phenotPPE.

For a candidate to be identified as a potential father of a progeny, there are two criteria to be met:

1. VLTotal >= max(VLTMin,mismatches+1),
2. FLCount >= VLTotal-mismatches.

Here, VLTmin and mismatches are user-specified parameters. VLTmin allows the user to ensure that a candidate is only considered for potential fatherhood if a sufficient number of valid loci were available for comparison. mismatches allows the user to specify a maximum number of allowed mismatching loci between progeny and candidate, before the candidate is rejected as a potential father. Hence the user may wish to relax the condition that ALL valid loci must match for a candidate to be regarded as a potential father to a progeny.

Value

A data frame, containing the columns Progeny (ID) Mother (ID), potentialFather (ID or None) FLCount and VLTotal (the FLCount and VLTotal values for the given potential father).

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## Using the example dataset 'FR_Genotype':
data(FR_Genotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
gData <- preprocessData(FR_Genotype,
                        numLoci=7,
                        ploidy=4,
                        dataType="genotype",
                        dioecious=TRUE,
                        mothersOnly=TRUE)

head(gData)  ## Checked and Cleaned version of FR_Genotype

gPPE <- genotPPE(gData)  ## Perform the exclusion analyses

## Obtain IDs of potential fathers of each seedling, allowing a
## single allele mismatch:
pFI <- potentialFatherIDs(gPPE,mismatches=1,VLTMin=2)
```

```
## pFC can be viewed or written to file via, e.g. write.csv()
```

`preprocessData` *Check and preprocess an allele dataset*

Description

Check and preprocess the input allele data frame prior to subsequent analysis.

Usage

```
preprocessData(adata, numLoci, ploidy, dataType, dioecious,
  selfCompatible = NULL, mothersOnly = NULL, lociMin = 1,
  matMismatches = 0)
```

Arguments

`adata`	data frame: an allele dataset.
`numLoci`	integer: the number of loci in the allele dataset.
`ploidy`	integer: the species' ploidy, one of `2`, `4`, `6`, or `8`.
`dataType`	character: either `"genotype"` or `"phenotype"`.
`dioecious`	logical: is the species dioecious or monoecious?
`selfCompatible`	logical: In monoecious species (`dioecious=FALSE`), can individuals self-fertilise? When `dioecious=FALSE`, this argument may be left at its default value of NULL - it will be set to `FALSE` by `preprocessData`.
`mothersOnly`	logical: in dioecious species, should females without progeny present be removed from the dataset? If `dioecious=TRUE`, then `mothersOnly` must be set to either `TRUE` or `FALSE`. If `dioecious=FALSE`, argument `mothersOnly` should be left at its default value of `NULL`.
`lociMin`	integer: the minimum number of loci in a individual that must have alleles present for the individual (and its progeny, if any) to be retained in the dataset (default 1).
`matMismatches`	an integer between 0 and `numLoci`-1, being the maximum number of mismatching alleles between mother and offspring that are allowed before the offspring is removed from the dataset. The default value is 0. If an offspring has fewer than `matMismatches` loci that mismatch with its mother, the offending loci are set to contain no alleles.

Details

If inputData is used to load the allele data set into R, then preprocessData will be called automatically on the data frame before it is returned by inputData. However, if the user loads their data into R by some means other than inputData, then preprocessData MUST be called on the data frame prior to using any other PolyPatEx functions to analyse the allele data—preprocessData performs a series of critical checks and preprocessing steps on the data frame, without which other analysis functions in PolyPatEx will fail.

Note that inputData strips leading or trailing spaces (whitespace) from each entry in the allele dataset as it is read in. If you load your data by a means other than inputData, you should ensure that you perform this step yourself, as preprocessData will not carry out this necessary step.

Note also that you should not use spaces in any of your allele codes - PolyPatEx functions use spaces to separate allele codes as they process the data - if allele codes already contains spaces, errors will occur in this processing. If you need a separator, I recommend using either 'code.' (a period) or 'code_' (an underscore) rather than a space.

preprocessData first performs a number of simple checks on the format and validity of the data set. These checks look for the presence of certain required columns and correct naming and content of these columns. preprocessData will usually stop with an error message should the data fail these basic checks. Correct the indicated problem in the CSV file or R allele data frame, then call inputData or preprocessData again as appropriate. If you use a spreadsheet to edit the CSV file, don't forget that you may also need to call fixCSV on the CSV file, prior to calling inputData again.

If the data is 'genotypic' data PolyPatEx requires that all p alleles must be present in each allele set, where p is the species' ploidy. If an allele set contains fewer than p alleles, then it is reset to contain no alleles and is subsequently ignored by other PolyPatEx functions. ID and locus information is printed to the R terminal, to help the user locate these cases in their original dataset.

Further checks look for mismatches between progeny and their mothers' allele sets at each locus—these are situations where a progeny's allele set could not have arisen from any gamete that the mother can provide. When no more than matMismatches mismatching loci occurs in a mother-progeny pair, the offending allele sets in the progeny are reset to contain no alleles (we term these 'missing' allele sets). When mismatches occur in more than matMismatches loci, the progeny is removed entirely from the dataset. Information is printed to the R terminal to assist the user in identifying the affected individuals and loci—in particular, note that removal of several (or all) of a single mother's progeny may indicate an error in the mother's allele data, rather than in her progeny.

After the mother/progeny mismatch check above, a subsequent check removes individuals from the dataset that have fewer than lociMin non-missing allele sets remaining. The default value for lociMin is 1—an individual must have at least one non-missing locus to remain in the dataset. If any mothers are removed from the dataset at this stage, all of her progeny are removed also. Again, information about these removals is printed to the R terminal.

Note that in the data frame that is returned by preprocessData, the alleles in each allele set (i.e, corresponding to each locus) will be sorted into alphanumeric order—this sort order is necessary for the correct operation of other PolyPatEx routines, and should not be altered.

PolyPatEx needs to know the characteristics of the dataset being analysed. These are specified in the inputData or preprocessData calls and are invisibly attached to the allele data frame that is returned, for use by other PolyPatEx functions. The required characteristics are:

- numLoci: the number of loci in the dataset

- `ploidy`: the ploidy (p) of the species (currently allowed to be 4, 6, or 8. `ploidy` can also be 2, provided `dataType="genotype"`)
- `dataType`: whether the data is genotypic (all p alleles at each locus are observed) or phenotypic (only the distinct allele states at a locus are observed - alleles that appear more than once in the genotype of a locus only appear once in the phenotype)
- `dioecious`: whether the species is dioecious or monoecious
- `selfCompatible`: whether a monoecious species is self compatible (i.e., whether an individual can fertilise itself)
- `mothersOnly`: whether a dioecious dataset should retain only adult females that are mothers of progeny in the dataset.

Value

A data frame, containing the checked and pre-processed allele data, ready for further analysis by other **PolyPatEx** functions. All columns in the returned data frame will be of mode `character`.

Author(s)

Alexander Zwart (alec.zwart at csiro.au)

Examples

```
## If using inputData to input the allele dataset from CSV file,
## preprocessData() is applied automatically before the dataset is
## returned by inputData().

## Otherwise, if the allele dataset is created or loaded into R
## by other means, such preprocessData() must be applied before
## other PolyPatEx analysis routines are applied:

## Using the example dataset 'GF_Phenotype':
data(GF_Phenotype)

## Since we did not load this dataset using inputData(), we must
## first process it with preprocessData() before doing anything
## else:
pData <- preprocessData(GF_Phenotype,
                        numLoci=7,
                        ploidy=6,
                        dataType="phenotype",
                        dioecious=FALSE,
                        selfCompatible=FALSE)

head(pData)  ## Checked and Cleaned version of GF_Phenotype

## pData is now ready to be passed to other PolyPatEx analysis
## functions...
```

Index