Package 'beadarrayMSV'

February 19, 2015

Type Package

Title Analysis of Illumina BeadArray SNP data including MSV markers

Version 1.1.0

Date 2011-02-20

Author Lars Gidskehaug

Maintainer Lars Gidskehaug <lg@camo.no>

Description Imports bead-summary data from Illumina scanner. Pre-processes using a suite of optional normalizations and transformations. Clusters and automatically calls genotypes, critically able to handle markers in duplicated regions of the genome (multisite variants; MSVs). Interactive clustering if needed. MSVs with variation in both paralogs may be resolved and mapped to their respective chromosomes. Quality control including pedigree checking and visual assessment of clusters. Too large data-sets are handled by working on smaller subsets of the data in sequence.

License GPL (>= 2)

LazyLoad yes

LazyData no

Depends R (>= 2.10.0), Biobase (>= 2.5.5), methods, geneplotter

Imports rggobi, limma

SystemRequirements GGobi

Repository CRAN

Date/Publication 2011-02-21 06:44:19

NeedsCompilation no

R topics documented:

beadarrayMSV-package 2
AlleleSetIllumina-class 3
assignParalogues 5

BeadSetIllumina-class . . . 8
BSRed.193 . . . 10
callGenotypes . . . 12
cart2pol . . . 17
countFailedSNP . . . 18
createAlleleSet . . . 19
findClusters . . . 22
findPolyploidClusters . . . 23
findSeTheta . . . 25
generatePolyCenters . . . 26
getCenters . . . 27
getNoiseDistributions . . . 29
getNormInd . . . 31
getSingleCalls . . . 32
locateParalogues . . . 33
makeDiploidCalls . . . 36
makeFilenames . . . 37
manualCall . . . 38
normalizeShearedChannels . . . 40
plotGenotypes . . . 42
preprocessBeadSet . . . 44
readBeadSummaryOutput . . . 47
resolveInheritanceSNP . . . 49
scatterArrays . . . 50
shearRawSignal . . . 51
testHardyWeinberg . . . 54
transformChannels . . . 55
translateTheta . . . 57
unmixParalogues . . . 59
validateCallsPedigree . . . 61
writeAlleleSet . . . 62

Index **64**

`beadarrayMSV-package` *beadarrayMSV: Package for analysis of high-throughput Illumina BeadArrays*

Description

Functions for reading, pre-processing and analyzing Illumina BeadArray data, in particular from the Infinium platforms. Methods for clustering and genotype calling of multisite variancs (MSVs) from polyploid genomes are provided. Paralogs may be resolved and mapped to individual chromosomes

Details

Package:	**beadarrayMSV**
Type:	Package
Version:	1.1
Date:	2011-02-20
License:	GPL (>=2)
LazyLoad:	yes
LazyData:	no
Depends:	R (>= 2.10.0), Biobase (>= 2.5.5), methods, geneplotter
Imports:	rggobi, limma
SystemRequirements:	GGobi

Two classes, "BeadSetIllumina" and "AlleleSetIllumina", are defined for holding and working with genetic data.

Illumina BeadArray summary-data are loaded using the function readBeadSummaryOutput, and pre-processed using preprocessBeadSet.

Automatic clustering and genotype calling is performed using callGenotypes, and validated using validateCallsPedigree and plotGenotypes.

Manual clustering with callGenotypes.interactive, which benefits from the interactive graphics package **rggobi**. Often used in combination with assignToAlleleSet.

Reading and writing data from and to files with the functions createAlleleSetFromFiles and writeAlleleSet.

For splitting MSV-5 markers into individual paralogs and assigning them to chromosomes, use assignParalogues.

Author(s)

Lars Gidskehaug

Maintainer: Lars Gidskehaug <lg@camo.no>

References

L. Gidskehaug, M. Kent, B. J. Hayes, and S. Lien. (2011) Genotype calling and mapping of multisite variants using an Atlantic salmon iSelect SNP-array. *Bioinformatics*, 27(3):303-310

AlleleSetIllumina-class

Class to Contain Objects Describing High-Throughput Illumina BeadArrays

---

Description

Container for high-throughput assays and experimental metadata. "AlleleSetIllumina" class is derived from "eSet" and requires matrices "intensity", "theta", and "SE" as assay data members

Objects from the Class

`new("AlleleSetIllumina", ...)`

`new("AlleleSetIllumina", phenoData = [AnnotatedDataFrame], featureData = [AnnotatedDataFrame], exp`

Most arguments are optional, however usually the arguments to `new` include at least "intensity", "theta", and "SE". Note that several methods requires additional `assayData`, `phenoData`, or `featureData` entries

`"AlleleSetIllumina"` instances are usually created from `"BeadSetIllumina"` instances using `createAlleleSet` or from text-files using `createAlleleSetFromFiles`

Slots

Inherited from `"eSet"`:

`assayData`**:** Object of class `"AssayData"`, with storage mode "list". Contains data-matrices with rows corresponding to markers and columns to samples/arrays. Required members are "intensity", "theta", and "SE". May also include intensities "A" and "B" (see `createAlleleSet`), "call" (see `callGenotypes`), "ped.check" (see `validateCallsPedigree`), or any other suitable matrix

`phenoData`**:** Object of class `"AnnotatedDataFrame"` with additional information about the samples

`featureData`**:** Object of class `"AnnotatedDataFrame"` with additional information about the markers

`experimentData`**:** Object of class `"MIAME"`

`annotation`**:** Object of class `"character"`

`.__classVersion__`**:** Object of class `"Versions"`

Extends

Class `"eSet"`, directly. Class `"VersionedBiobase"`, by class `"eSet"`, distance 2. Class `"Versioned"`, by class `"eSet"`, distance 3.

Methods

Class-specific methods:

`initialize`**:** `signature(.Object = "AlleleSetIllumina")`: Object instantiation, can be called by derived classes but not usually by the user

`validObject(object)`**:** Validity-checking method, ensuring (1) all `assayData` components have the same number and names of features and samples; (2) the number and names of `phenoData` and `featureData` rows match the number and names of `assayData` columns and rows, respectively; (3) `storageMode(object)` is set to "list"

`assignToAlleleSet(object,object1)`**:** Used to assign data in `object1` to corresponding features in `object`, given the `phenoData` are identical between the two

A selection of methods inherited from `"eSet"`:

`sampleNames(object)`, `sampleNames(object) <- value`**:** See `"eSet"`

`featureNames(object)`, `featureNames(object) <- value`: See "eSet"

`phenoData(object)`, `phenoData(object) <- value`: See "eSet"

`pData(object)`, `pData(object) <- value`: See "eSet"

`varMetadata(object)` See "eSet"

`varLabels(object)` See "eSet"

`featureData(object)`, `featureData(object) <- value`: See "eSet"

`fData(object)`, `fData(object) <- value`: See "eSet"

`fvarMetadata(object)` See "eSet"

`fvarLabels(object)` See "eSet"

`assayData(object)`, `assayData(object) <- value`: See "eSet"

`combine(object,object1)`: See "eSet"

`storageMode(object)`: See "eSet". Do not assign new values to `storageMode`, as this could render the object invalid

Author(s)

Lars Gidskehaug

See Also

`createAlleleSet`, `callGenotypes`, `preprocessBeadSet`, `BeadSetIllumina`, `MultiSet`

Examples

```
showClass("AlleleSetIllumina")
```

---

assignParalogues *Assign MSV-5 paralogs to chromosomes*

Description

Based on linkage information and a set of MSV-5 markers which have been split into individual paralogs within half-sib families, this function attempts to map the paralogs to their respective chromosomes and name them accordingly

Usage

```
setMergeOptions(minC = NULL, noiseQuantile = 0.75,
    offspringLim = 7, ratioLim = 0.9, rngLD = 5)

assignParalogues(BSSnp, BSRed,
    paraCalls = unmixParalogues(BSRed, singleCalls),
    inheritP = resolveInheritanceSNP(BSSnp),
    singleCalls = getSingleCalls(BSRed),
    cHits = locateParalogues(BSSnp, paraCalls, inheritP,
        mO$offspringLim, mO$ratioLim)$cPerMarker,
    mO = setMergeOptions())
```

Arguments

minC	A numeric value corresponding to the elements of cHits below which no chromosomes are detected
noiseQuantile	The quantile of the third largest chromosomes across markers from which minC may be estimated
offspringLim	In order for a match between a paralogue and a chromosome to be detected, the number of (informative) half-siblings must equal or exceed this numeric value
ratioLim	The patterns of paternal and maternal inherited alleles among half-sib family offspring are compared between MSV-5 paralogs (see unmixParalogues) and genetic map SNPs (see resolveInheritanceSNP). The ratio of matching allele patterns between the two must equal or exceed this numeric value in order for a chromosome match to be detected
rngLD	Numeric indicating how many map-units (e.g. cM) to include on each side of the genetic map marker to increase the number of informative meioses and the power of the associations with the paralogs.
BSSnp	"AlleleSetIllumina" (or "MultiSet") object containing SNPs of known location on the chromosomes, including an assayData entry "call". A phenoData column PedigreeID must be included on the form <p><mmm><fff><oo>, identifying the population, mother, father and individual offspring, respectively. The featureData variables "Chromosome", "Female", and "Male" give the numbered chromosome and the genetic distances on the female and male map, respectively
BSRed	"AlleleSetIllumina" (or "MultiSet") object containing MSV-5's to be mapped, with a required assayData-list entry "call". Must contain the same samples as BSSnp, and also a phenoData column "PedigreeID"
paraCalls	List containing two matrices, "mother" and "father", with the parental inherited alleles of individual paralogs assuming unknown alternate parent (see unmixParalogues)
inheritP	List containing two matrices, "mother" and "father", with the parental inherited alleles for the markers in BSSnp (see resolveInheritanceSNP)
singleCalls	Matrix containing MSV-5s for which both paralogs are either monomorphic or polymorphic (see getSingleCalls)
cHits	A three-dimensional array of size (markers x chromosomes x 2) containing an average number of matches of a paralogue to a chromsome for both the mothers and fathers (average across the number of markers in the map for that chromosome; see locateParalogues)
mO	List with options used in the mapping of paralogs (see setMergeOptions)

Details

While the function locateParalogues allows for matching of paralogs to any chromosome, assignParalogues uses the former output and limits the allowed choices to one or two chromosomes. The paralogs are given names reflecting these chromosomes, which allows for merging of the linkage information in paraCalls into a single, much more informative data-table.

Initially, the largest value of cHits between "mother" and "father" is chosen, and the resulting scores are sorted decreasingly among chromosomes one marker at the time. Up to two of the highest

scoring chromosomes are selected if their values exceed `mO$minC`. If this element is `NULL`, it will be estimated based on the `mO$noiseQuantile`'th quantile of the third highest ranking chromosomes across markers. Also, the second ranking chromosome will not be selected unless it scores twice as high as the third ranking chromosome. Using the maternal and paternal half-sib families in turn, each paralogue is mapped to either of the selected chromosomes if sufficient association is detected.

For each half-sib family and each (informative) paralogue, only genetic map markers for which the parent in question is heterozygous are useful. This reduces the number of genetic map markers to which the paralogs can be associated. Similarly, for only a subset of the half-siblings are the parental inherited alleles in each paralogue known. This tends to reduce the number of informative offspring in each family drastically. Missing parental alleles among the genetic map markers further reduce the numbers of informative offspring, however these may sometimes be imputed using neighbouring markers assumed to be in linkage disequilibrium (LD) with the marker in question. The option `rngLD` indirectly controls the number of helping markers to use.

The mapping itself proceeds by applying the filter defined in `mO` to the genetic map markers on a specific chromosome (see `locateParalogues` for specifics about the filter). A set of statistics are then calculated to find the marker that matches the chromosome most closely. If there are two candidate chromosomes, the one with the highest ranked marker is selected. If there is only one candidate, it is selected if it outranks all the other chromosomes in terms of the calculated statistics. If a succesfull match is found, the parental inherited alleles for that family are assigned to the paralogue whose name reflects the chromosome match.

Value

A list containing

`x`	a matrix holding the calls for those paralogs that are successfully mapped to a chromosome. The rownames reflect the chromosome as well as the marker-name
`chromPairs`	a matrix with 0, 1, or 2 chromosomes to which the MSV-5's have been successfully mapped
`positionFemale`	a matrix holding the mapped paralogue positions as estimated by the female parent half-sib families
`positionMale`	a matrix holding the mapped paralogue positions as estimated by the male parent half-sib families

Note

This function may be time consuming, and even more so if many of the input parameters need be re-calculated each time. If some of them are available in the workspace, save time by including them in the function call

Author(s)

Lars Gidskehaug

See Also

`plotCountsChrom`, `setMergeOptions`, `unmixParalogues`, `resolveInheritanceSNP`, `MultiSet`, `AlleleSetIllumina`, `locateParalogues`, `getSingleCalls`

Examples

```
## Not run:
#Read markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling and splitting of MSV-5 paralogs
BSRed <- callGenotypes(BSRed)
BSRed <- validateCallsPedigree(BSRed)
iMSV5 <- fData(BSRed)$Classification %in% 'MSV-5' &
    fData(BSRed)$Ped.Errors %in% 0
singleCalls <- getSingleCalls(BSRed[iMSV5,])
paraCalls <- unmixParalogues(BSRed[iMSV5,],singleCalls)

#Genetic map SNPs and inherited parental alleles
iSNP <- fData(BSRed)$Classification %in% 'SNP' &!
    is.na(fData(BSRed)$Chromosome)
inheritP <- resolveInheritanceSNP(BSRed[iSNP,])

#Match paralogs with map
mO <- setMergeOptions(minC=1)
chromHits <- locateParalogues(BSRed[iSNP,],paraCalls,
   inheritP,mO$offspringLim,mO$ratioLim)

#The example data and map are too small to detect most homeologies
plotCountsChrom(chromHits$cPerMarker,1:sum(iMSV5),at=1:15,
   labels=dimnames(chromHits$c)[[2]],las=2)

#Only a few, single paralogs are succesfully assigned to chromosomes
mergedCalls <- assignParalogues(BSRed[iSNP,],BSRed[iMSV5],paraCalls,
   inheritP,singleCalls,cHits=chromHits$cPerMarker,mO=mO)
print(mergedCalls$chromPairs)
print(mergedCalls$x[,1:4])

## End(Not run)
```

`BeadSetIllumina-class` *Class to Contain Objects Describing High-Throughput Illumina BeadArrays*

Description

Container for high-throughput assays and experimental metadata. "`BeadSetIllumina`" class is derived from "`eSet`" and requires matrices "R", "G", "se.R", "se.G", and "no.beads" as assay data members

Objects from the Class

`new("BeadSetIllumina", ...)`

`new("BeadSetIllumina", phenoData = [AnnotatedDataFrame], featureData = [AnnotatedDataFrame],`

`"BeadSetIllumina"` instances are usually created from bead type summary text-files using `readBeadSummaryOutput`

Slots

`assayData`: Object of class `"AssayData"`, with storage mode "list". Contains data-matrices with rows corresponding to markers and columns to samples/arrays. Required members are "R", "G", "se.R", "se.G", and "no.beads".

`phenoData`: Object of class `"AnnotatedDataFrame"` with additional information about the samples.

`featureData`: Object of class `"AnnotatedDataFrame"` with additional information about the markers.

`experimentData`: Object of class `"MIAME"`

`annotation`: Object of class `"character"`

`.__classVersion__`: Object of class `"Versions"`

Extends

Class `"eSet"`, directly. Class `"VersionedBiobase"`, by class "eSet", distance 2. Class `"Versioned"`, by class "eSet", distance 3.

Methods

Class-specific methods:

`initialize signature(.Object = "BeadSetIllumina")`: Object instantiation, can be called by derived classes but not usually by the user.

`validObject(object)`: Validity-checking method, ensuring (1) all `assayData` components have the same number and names of features and samples; (2) the number and names of `phenoData` and `featureData` rows match the number and names of `assayData` columns and rows, respectively; (3) `storageMode(object)` is set to "list"

A selection of methods inherited from `"eSet"`:

`sampleNames(object)`, `sampleNames(object) <- value`: See `"eSet"`

`featureNames(object)`, `featureNames(object) <- value`: See `"eSet"`

`phenoData(object)`, `phenoData(object) <- value`: See `"eSet"`

`pData(object)`, `pData(object) <- value`: See `"eSet"`

`varMetadata(object)` See `"eSet"`

`varLabels(object)` See `"eSet"`

`featureData(object)`, `featureData(object) <- value`: See `"eSet"`

`fData(object)`, `fData(object) <- value`: See `"eSet"`

`fvarMetadata(object)` See "eSet"

`fvarLabels(object)` See "eSet"

`assayData(object), assayData(object) <- value`: See "eSet"

`combine(object,object1)`: See "eSet"

`storageMode(object)`: See "eSet". Do not assign new values to `storageMode`, as this could render the object invalid

Author(s)

Lars Gidskehaug

See Also

`readBeadSummaryOutput`, `preprocessBeadSet`, `createAlleleSet`, `AlleleSetIllumina`, `eSet`

Examples

```
showClass("BeadSetIllumina")
```

`BSRed.193` *Atlantic salmon genotype data with mainly MSV-5 markers*

Description

This data set contains data for 193 markers genotyped for 3230 Atlantic salmon half-sib family individuals. A large part of the markers are multisite variants segregating in both paralogs (MSV-5's)

Usage

```
data(BSRed.193)
```

Format

The format is: Formal class 'AlleleSetIllumina' [package **beadarrayMSV**] with 7 slots
..@ assayData : List of 6
.. ..$ intensity : num [1:193, 1:3230] 7.05 6.17 2.94 4.28 4.23 ...
.. ..$ theta : num [1:193, 1:3230] 0.3469 0.4952 0.7389 -0.0202 0.3441 ...
.. ..$ SE : num [1:193, 1:3230] 0.00582 0.00402 0.01764 0.00941 0.01083 ...
.. ..$ call : num [1:193, 1:3230] 0.25 0.75 0.75 0 0.25 1 0.75 0.5 0 0.75 ...
.. ..$ ped.check : logi [1:193, 1:3230] TRUE NA TRUE TRUE TRUE TRUE ...
.. ..$ ped.check.parents: num [1:193, 1:3230] 0 0 0 0 0 0 0 0 0 0 ...
.. .. ..- attr(*, "dimnames")=List of 2
.. ..$: chr [1:193] "AY388579_a" "AY388580_a" "ESTNV_14208_491" ...
.. ..$: chr [1:3230] "X4469076125_A" "X4469076125_B" "X4469076309_A" ...
..@ phenoData : Formal class 'AnnotatedDataFrame' [package **Biobase**] with 4 slots
.. .. ..@ varMetadata :'data.frame': 14 obs. of 1 variable:

.. ..$ labelDescription: chr [1:14] "Array ID" "No. detected beads" "Chip ID" ...
.. .. ..@ data :'data.frame': 3230 obs. of 14 variables:
.. ..$ arrayNames : chr [1:3230] "4469076125_A" "4469076125_B" "4469076309_A" ...
.. ..$ no.beads : int [1:3230] 558739 542929 549125 526811 511354 ...
.. ..$ chip : num [1:3230] 4.47e+09 4.47e+09 4.47e+09 4.47e+09 4.47e+09 ...
.. ..$ row : chr [1:3230] "A" "B" "A" "B" ...
.. ..$ col : int [1:3230] 1 1 1 1 1 1 1 1 1 1 ...
.. ..$ noiseIntensity: num [1:3230] 0.995 1.063 0.859 0.89 1.019 ...
.. ..$ Sample.ID : chr [1:3230] "100050151" "100050152" "100050153" ...
.. ..$ Sample.Plate : chr [1:3230] "SMS_009" "SMS_009" "SMS_009" ...
.. ..$ Sample.Name : chr [1:3230] "017_532_0385" "017_532_0386" "017_532_0387" ...
.. ..$ AMP.Plate : chr [1:3230] "WG0011117-MSA2" "WG0011117-MSA2" ...
.. ..$ Sample.Well : chr [1:3230] "E01" "F01" "G01" ...
.. ..$ Parent1 : chr [1:3230] "100000000" "100000000" "100000000" ...
.. ..$ Parent2 : chr [1:3230] "100050100" "100050100" "100050100" ...
.. ..$ PedigreeID : chr [1:3230] "199950151" "199950152" "199950153" ...
..@ featureData : Formal class 'AnnotatedDataFrame' [package **Biobase**] with 4 slots
.. .. ..@ varMetadata :'data.frame': 16 obs. of 1 variable:
.. ..$ labelDescription: chr [1:16] "" "Marker ID" "Polymorphism" ...
.. .. ..@ data :'data.frame': 193 obs. of 16 variables:
.. ..$ Index : int [1:193] 2 3 256 426 640 ...
.. ..$ Name : chr [1:193] "AY388579_a" "AY388580_a" "ESTNV_14208_491" ...
.. ..$ SNP : chr [1:193] "[A/G]" "[T/G]" "[T/C]" ...
.. ..$ ILMN.Strand : chr [1:193] "TOP" "BOT" "BOT" ...
.. ..$ Address : int [1:193] 7560672 6370541 6660563 2190240 3140563 ...
.. ..$ Address2 : int [1:193] 0 0 0 0 0 0 3120064 0 ...
.. ..$ Norm.ID : int [1:193] 1 2 2 2 2 2 102 2 ...
.. ..$ Classification: chr [1:193] "MSV-5" "FAIL" "MSV-5" ...
.. ..$ Cent.Deviation: num [1:193] 0.2315 0.3952 0.0796 0.018 0.0863 ...
.. ..$ Within.SD : num [1:193] 0.0258 0.0176 0.0494 0.0188 0.0332 ...
.. ..$ HW.Chi2 : num [1:193] 0.524 8.113 9.541 NA 7.628 ...
.. ..$ HW.P : num [1:193] 0.9135 0.0437 0.0229 NA 0.0544 ...
.. ..$ BAF.Locus1 : num [1:193] 0.508 0.819 0.887 NA 0.285 ...
.. ..$ BAF.Locus2 : num [1:193] 0.508 0.729 0.359 NA 0.126 ...
.. ..$ Call.Rate : num [1:193] 0.981 0.912 0.95 0.98 0.978 ...
.. ..$ Manual.Calls.R: chr [1:193] "MSV-5" "MSV-5" "MSV-5" ...
..@ experimentData :Formal class 'MIAME' [package **Biobase**]
..@ annotation : chr(0) ..@ protocolData :Formal class 'AnnotatedDataFrame' [package **Biobase**]
..@ .__classVersion__:Formal class 'Versions' [package **Biobase**]

Source

L. Gidskehaug, M. Kent, B. J. Hayes, and S. Lien. (2011) Genotype calling and mapping of multi-site variants using an Atlantic salmon iSelect SNP-array. *Bioinformatics*, 27(3):303-310

Examples

```
data(BSRed.193)
```

```
print(BSRed.193)
varMetadata(BSRed.193)
fvarMetadata(BSRed.193)
assayData(BSRed.193)$call[1:10,1:4]

## Not run:
plotGenotypes(BSRed.193,markers=1:12)

## End(Not run)
```

`callGenotypes` *Clustering and calling of genotypes*

Description

Clusters each marker into the most likely combination from a sequence of allowed cluster combinations. These may include multisite variants from polyploid genomes

Usage

```
setGenoOptions(largeSample = FALSE, snpPerArrayLim = 0.8,
    arrayPerSnpLim = 0, ploidy = "tetra",
    filterLim = 0, detectLim = 0.8,
    wSpreadLim = suggestGeno(largeSample)$wSpreadLim,
    devCentLim = 0.35,
    hwAlpha = suggestGeno(largeSample)$hwAlpha,
    probsIndSE = suggestGeno(largeSample)$probsIndSE,
    afList = seq(0, 0.5, 0.05),
    clAlpha = suggestGeno(largeSample)$clAlpha,
    rPenalty = 2,
    rotationLim = suggestGeno(largeSample)$rotationLim,
    minClLim = 5, nSdOverlap = 2,
    minBin = suggestGeno(largeSample)$minBin,
    binWidth = suggestGeno(largeSample)$binWidth)

callGenotypes(BSRed,
    gO = setGenoOptions(largeSample = ncol(BSRed) > 250))

callGenotypes.interactive(BSRed,
    gO = setGenoOptions(largeSample = ncol(BSRed) > 250))

callGenotypes.verboseTest(BSRed, singleMarker = 1,
    gO = setGenoOptions(largeSample = ncol(BSRed) > 250))
```

Arguments

`largeSample`	Logical indicating whether or not a large sample is genotyped, which influences the values of some default genotyping options. More than 2-300 arays may be considered large

snpPerArrayLim	Minimum ratio of non-NA markers per array. Failing arrays are disregarded
arrayPerSnpLim	Minimum ratio of non-NA arrays per marker. Failing markers are disregarded
ploidy	Character string indicating which genotyping classes to allow (see generatePolyCenters)
filterLim	Markers with range of assayData "theta" below this value are classified as "MONO-filt" and are not subjected to clustering. This option will speed up the analysis, however the calls and the statistics of classification will be missing for these markers
detectLim	Ratio of called arrays below which markers are classified as "FAIL"
wSpreadLim	The maximum allowed within cluster standard deviation along assayData "theta"
devCentLim	The maximum allowed difference in "theta" between expected and estimated cluster centres
hwAlpha	Significance level used in Hardy-Weinberg testing. Used to detect where the clustering fails, and a low value (e.g. 1e-10 - 1e-40) may be used to allow natural deviation from HW. This criterion should be used with caution, especially when the sample contains animals from different populations
probsIndSE	Numeric quantile for which markers with higher standard errors are excluded from the clustering step (however not discarded)
afList	Numeric vector of values within [0, 0.5], denoting which B allele frequencies to test in the case of two segregating paralogs (see below)
clAlpha	Numeric significance level for Hotelling's T^2 test (see below)
rPenalty	Scaling for the ordinate axis spanned by assayData "intensity" (see below). A high value means the influence of the intensity in the clustering is decreased
rotationLim	Controls the maximum allowed tilt of clusters, as defined by Hotelling's ellipses (see below)
minClLim	Clusters below this minimum size are disregarded in the Hotelling's test (all animals included in cluster)
nSdOverlap	Numeric multiple of assayData "theta" standard deviations defining within cluster spread
minBin	When estimating initial center points for the clustering, histogram peaks below this numeric value are set to zero
binWidth	Starting value for histogram bin-width used to find initial center points. Smaller clusters are detected with smaller bin-width
BSRed	"AlleleSetIllumina" object, holding a required phenoData column "noiseIntensity" (see preprocessBeadSet)
gO	List of options and cut-off limits used in the clustering (see setGenoOptions)
singleMarker	Index to a specific marker to test

Details

Initially, arrays with a fraction of non-NA markers less than gO$snpPerArrayLim, and markers with a fraction of non-NA arrays less than gO$arrayPerSnpLim, are discarded from the analysis. Also, markers with range of assayData "theta" below gO$filterLim are called "MONO-filt" and are not subjected to clustering.

Genotype calling for each marker is based on k-means clustering in the two dimensions defined by `assayData` entries "intensity" and "theta", where "intensity" is penalized by `gO$rPenalty` times its median value. The value of `gO$ploidy` sets the allowed cluster combinations through the function `generatePolyCenters`. Data points below the average value of the `phenoData` column "noiseIntensity" across included arrays are set to missing, and arrays with standard errors exceeding the quantile given by `gO$probsIndSE` are left out from the clustering step. Two or three of the most likely cluster combinations are estimated with the function `getCenters`. This function uses histograms to find starting values for the clustering, with input parameters `gO$minBin` defining a peak-height filter and `gO$binWidth` setting an initial bin-width. The ranked cluster combinations are tested in turn until one is found which passes the criteria below.

First, `gO$devCentLim` controls the maximum distance a cluster is allowed to deviate from its expected or ideal position. Second, `gO$wSpreadLim` limits how large the within cluster standard deviation can be. Both these criteria are tested in the "theta"-dimension, and if either fails, the algorithm will attempt to cluster the next most likely configuration and start over.

The next test is for Hardy-Weinberg (HW) equilibrium. This is a very powerful test to detect if the clustering has failed, given HW can be assumed. Otherwise, set the significance value `gO$hwAlpha` to zero to allow any deviation from HW. At duplicated loci, the observed B allele-frequency (BAF) is in fact the mean BAF across both paralogues. Several candidate values of BAF for one paralogue are set in `gO$afList`, such that the BAF for the other paralogue is given. Several values of `gO$afList` within [0, 0.5] are tested for HW, and the most likely BAF at both paralogs are those resulting in the highest p-value. Another, somewhat less powerful, quality control test is to look for overlapping clusters. If a cluster centre on the "theta"-axis is denoted `Cent`, and its spread is given by `Within.SD`, clusters are tested for overlap using `Cent +/- gO$nSdOverlap * Within.SD`

Markers passing the quality control are subjected to a Hotelling's T^2-test (Hotelling, 1931; Gidskehaug et al., 2007), which effectively superimposes an ellipse on each cluster and discards all data points falling outside its boundaries or within overlapping ellipses. Due to inaccurate ellipses for small clusters, only clusters with more than `gO$minClLim` members are tested. The extents of the clusters are controlled by the significance level `gO$clAlpha` of the T^2-tests in such a way that a small value yield large ellipses. The orientation of an ellipse is defined by the ratio of variances in the variance-/covariance matrix from the Hotelling's test. If the ratio of the "theta"-variance to the "intensity"-variance is given by `Sratio`, the weighted sum of `Sratio` across clusters is not allowed to exceed `gO$rotationLim`. Finally, the call rate is required to exceed `gO$detectLim`, or the algorithm will move on to the next most likely cluster combination. If a marker passes all the tests for one of the candidate cluster combinations, the genotype is called. Otherwise the marker is failed (i.e. called "FAIL").

`callGenotypes` is the main genotype calling function. Before genotyping thousands of markers, it is important spend a little time tuning the options in `gO`. A few hundred markers may be called using the default settings, and plotted using `plotGenotypes`. Individual markers whith questionable calls can be investigated more closely using `callGenotypes.verboseTest`. This function performs all the tests described above, for all suggested cluster combinations, without failing any of the tests. Rather, all the test results are plotted and returned, revealing which steps can be taken to improve the clustering, if any. In these plots, green dots are the estimated centre points, and red and orange dots represent arrays outside cluster boundaries or within overlapping ellipses, respectively. Several markers representing each of the possible genotype categories should be investigated in this way before commencing with genotyping the full set of markers.

For some markers with highly overlapping or inaccurate clusters, interactive clustering might be the last option. This can be performed using the function `callGenotypes.interactive` (requires

package **rggobi** to be installed). This function will loop through a smaller set of markers, allowing the user to define clusters manually. Pedigree checking is performed inside the function, and a phenoData column "PedigreeID" is required (see validateCallsPedigree). Use the function assignToAlleleSet to incorporate the manual results into a larger "AlleleSetIllumina" object.

Value

Output from setGenoOptions is a list with all defined or suggested options needed for genotype calling

Output from callGenotypes is an "AlleleSetIllumina" object with an extra assayData entry:

call	Matrix with numeric values {0, 1/4, 1/2, 3/4, 1}, each indicating the ratio of B alleles for a marker in a sample

The "AlleleSetIllumina" output from callGenotypes.interactive in addition holds the assayData entries:

ped.check	Logical matrix with FALSE indicating pedigree violations among offspring. Parental values are all NA (see validateCallsPedigree)
ped.check.parents	Matrix of numeric values {0, 1, 2}, denoting no error, offspring error, and parent error, respectively (see validateSingleCall)

The following columns are added to featureData:

Classification	Genotype call from automatic clustering
Cent.Deviation	Largest distance from cluster-centre to its ideal position
Within.SD	Largest within-cluster spread
HW.Chi2	Chi-squared statistic from test of Hardy-Weinberg equilibrium
HW.P	Probability of Hardy-Weinberg equilibrium
BAF.Locus1	Estimated B-allele frequency
BAF.Locus2	Estimated BAF of second paralogue (if exists)
Call.Rate	Ratio of arrays assigned to clusters
Manual.Calls.R	Calls from interactive clustering (if applied)

Output from callGenotypes.verboseTest is a list containing

call	The calls from each attempted cluster-combination
fData	Statistics from each cluster-combination
fMetadata	Explanation to each statistic
test	Table of "PASS" or "FAIL" for each test

Note

When pedigree data are available, pedigree checking is a very strong type of validation which is performed *after* the genotype calling in `callGenotypes`

In `callGenotypes.interactive`, pedigree checking is performed *during* genotype calling. This means the pedigree cannot subsequently be used as validation to the same degree as if it had been kept secret during genotype calling

Author(s)

Lars Gidskehaug

References

L. Gidskehaug, E. Anderssen, A. Flatberg, and B. K. Alsberg (2007) A framework for significance analysis of gene expression data using dimension reduction methods. *BMC Bioinformatics* **8**:346

H. Hotelling (1931) The Generalization of Student's Ratio. *Ann. Math. Stat.* **2**(3):360-378

See Also

`generatePolyCenters`, `getCenters`, `findPolyploidClusters`, `testHardyWeinberg`, `ggobi`, `plotGenotypes`, `validateCallsPedigree`, `assignToAlleleSet`, `manualCall`

Examples

```
## Not run:
#Read 25 markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],markers=1:25,beadInfo=beadInfo)

#Genotype calling and validation
gO <- setGenoOptions(largeSample=TRUE)
BSRed <- callGenotypes(BSRed, gO=gO)
BSRed <- validateCallsPedigree(BSRed)
sumClass <- tapply(rep(1,nrow(BSRed)),fData(BSRed)$Classification,sum)
print(sumClass)

#Plot default setting calls
plotGenotypes(BSRed)

#Tune settings to call an initially failed marker
dev.new()
verboseRes <- callGenotypes.verboseTest(BSRed, gO=gO, singleMarker=23)
print(verboseRes$fData)
print(verboseRes$test)
gO <- setGenoOptions(largeSample=TRUE, wSpreadLim=.1, hwAlpha=1e-50)
```

```
verboseRes <- callGenotypes.verboseTest(BSRed, gO=gO, singleMarker=23)

#New settings give (likely incorrect) SNP-call
BSRed <- callGenotypes(BSRed, gO=gO)
BSRed <- validateCallsPedigree(BSRed)
dev.new()
plotGenotypes(BSRed)

## End(Not run)
```

`cart2pol` *Transformation from Cartesian to polar coordinates*

Description

Cartesian coordiantes are transformed to polar coordinates using a specified distance measure

Usage

```
cart2pol(x, y, dist = "euclidean", pNorm = NULL)
```

Arguments

`x`	Cartesian abscissa value
`y`	Cartesian ordinate value
`dist`	One of "manhattan", "euclidean" (default), or "minkowski", defining which distance measure to use
`pNorm`	The exponent "p" in the p-norm (Minkowski) distance

Details

The Manhattan distance is the special case of the Minkowski distance of norm 1, the Euclidean distance equals the Minkowski distance of norm 2. The unit circle of the Minkowski distance changes from a diamond shape (1-norm), through an Euclidean circle (2-norm), to a square (infinity-norm) as `pNorm` increases from 1 to infinity. Any norm between 2 and infinity results in a unit circle resembling a square with more or less rounded corners.

For non-transformed signal, the 1-norm is the most accurate representation, however a higher norm is called for after root- or log-transformations.

Value

A list holding the polar coordinates

`r`	Signal intensity
`th`	Polar angle

Author(s)

Lars Gidskehaug

See Also

preprocessBeadSet, setNormOptions

Examples

```
## Points defining a 4-norm unit-circle in the first quadrant
x <- seq(0,1,.01)^(1/4)
y <- (1-x^4)^(1/4)

## Polar coordinates using different distance-measures
eucl <- cart2pol(x,y)
mink <- cart2pol(x,y,'minkowski',4)

## Not run:
## Plot in cartesian coordinates
dev.new()
plot(x,y,type='b',main='4-norm unit circle')

## Plot polar coordinates on cartesian axes
dev.new()
plot(eucl$th,eucl$r,type='b',col='red',xlab='theta',ylab='r')
points(mink$th,mink$r,type='b',col='blue')
title(main='Euclidean (red) and 4-norm (blue) distance')
## End(Not run)
```

---

countFailedSNP *Calculate ratio of called markers for each array*

Description

Not counting markers classified as "FAIL" or "MONO-filt", calculate the ratio of markers that are called for each array

Usage

```
countFailedSNP(BSRed, inclPedErrors = TRUE)
```

Arguments

BSRed	"AlleleSetIllumina" object containing an assayData entry "call" (see callGenotypes)
inclPedErrors	If TRUE, calls violating pedigree count as missing

Details

In order to include pedigree errors, BSRed must have an assayData entry ped.check (see validateCallsPedigree)

Value

An "AlleleSetIllumina" object with an added phenoData column "passRatio". This is a numeric vector holding the ratio of non-missing calls for each array

Note

This function may be used to discard arrays with a low number of called markers. Note however that an array may be valuable even if the sample falls just outside the cluster borders for many markers (e.g. for a high intensity array). Should therefore be used with caution

Author(s)

Lars Gidskehaug

See Also

callGenotypes, validateCallsPedigree

Examples

```
## Not run:
#Read pre-processed data directly into AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling
BSRed <- callGenotypes(BSRed)
BSRed <- validateCallsPedigree(BSRed)
BSRed <- countFailedSNP(BSRed,inclPedErrors=TRUE)
print(range(pData(BSRed)$passRatio))

#Plot highlighting markers to be discarded
#NB! Such a high passRatio is not recommended
indGoodArrays <- pData(BSRed)$passRatio > 0.6
plotGenotypes(BSRed,indHighlight=which(!indGoodArrays))

## End(Not run)
```

---

createAlleleSet *Create AlleleSetIllumina or MultiSet objects*

Description

An "AlleleSetIllumina" (or "MultiSet") object is created, either by transforming a "BeadSetIllumina" object into an "AlleleSetIllumina" object, by reading from text-files containing pre-processed data, or by merging existing objects

Usage

```
createAlleleSet(BSData, beadInfo, normOpts, includeAB = FALSE)

createAlleleSetFromFiles(dataFiles, markers, arrays,
    phenoInfo = NULL, beadInfo = NULL, sep = "\t", quote = "")

createMultiSetFromFiles(dataFiles, markers, arrays,
    phenoInfo = NULL, beadInfo = NULL, sep = "\t", quote = "")

assignToAlleleSet(BSRed, BSAdd)
```

Arguments

BSData	"BeadSetIllumina" object
beadInfo	Data-frame containing an entry for each marker, and the columns "Name", "SNP", "ILMN.Strand", "Address", "Address2", and "Norm.ID", as exported from Illumina's GenomeStudio Genotyping Module (or relatives). May also contain columns relating to genotype calls (see callGenotypes)
normOpts	List containing at least the elements "dist" and "pNorm" (see setNormOptions)
includeAB	If TRUE, the Cartesian signal arrays "A" and "B" are returned as assayData entries in the new "AlleleSetIllumina" object
dataFiles	Character vector containing filenames where the different data-tables are saved (see makeFilenames)
markers	Index to markers in the dataFiles files
arrays	Index to arrays/samples in the dataFiles files
phenoInfo	Data-table with phenotype data. Argument is ignored if "phFile" is provided in dataFiles
sep	Field delimiter in text-files (see read.table)
quote	Quote-marks used for character strings (see read.table)
BSRed	"AlleleSetIllumina" object
BSAdd	"AlleleSetIllumina" object with data to include in BSRed

Details

A "BeadSetIllumina" object contains bead-type information, whereas an "AlleleSetIllumina" object contains marker information (for each Infinum I marker, there are two bead-types). The function createAlleleSet takes a "BeadSetIllumina" object as input and merges the "R" and "G" intensities into "A" and "B" intensities. The former relates to bead-types and the latter relates to markers as defined in beadInfo. The required polar coordinate intensities "intensity" and "theta" are estimated based on "A" and "B", and depend on the distance measures defined in normOpts. The angles "theta" are scaled such that {0, 90} degrees are represented by {0, 1}, and "intensity" vs. "theta" for single markers are usually plotted on Cartesian axes for genotype calling.

If data-files are available for all required assayData, phenoData, and featureData elements, an "AlleleSetIllumina" or a "MultiSet" object may be constructed with create-AlleleSetFromFiles

or createMultiSetFromFiles, respectively. The former has three required assayData elements whereas the latter has none.

Sometimes, and in particular after manual genotype calling, there is a need to update an "AlleleSetIllumina" object with new information. The function assignToAlleleSet adds any data in BSAdd to BSRed, overwriting previous data if there is a conflict

Value

Object of class "AlleleSetIllumina" or "MultiSet"

Author(s)

Lars Gidskehaug

See Also

AlleleSetIllumina, MultiSet, writeAlleleSet, makeFilenames

Examples

```
## Not run:
#Read raw data files into BeadSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
BSDataRaw <- readBeadSummaryOutput(path=rPath,recursive=TRUE)

#Find indexes to sub-bead pools
beadInfo <- read.table(paste(rPath,'beadData.txt',sep='/'),sep='\t',
    header=TRUE,as.is=TRUE)
rownames(beadInfo) <- make.names(beadInfo$Name)
normInd <- getNormInd(beadInfo,featureNames(BSDataRaw))

#Pre-process BSData
normOpts <- setNormOptions(minSize=10)
plotPreprocessing(BSDataRaw,normInd,normOpts,plotArray=1)
BSData <- preprocessBeadSet(BSDataRaw,normInd,normOpts)
print(BSData)
print(fData(BSData)[1:10,])
print(fvarMetadata(BSData))

#Convert to AlleleSetIllumina-object
BSRed <- createAlleleSet(BSData,beadInfo,normOpts)
print(BSRed)
print(fData(BSRed)[1:10,])
print(fvarMetadata(BSRed))

#Read pre-processed data directly into AlleleSetIllumina object
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo2 <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo2)
print(varLabels(BSRed))
```

```
## End(Not run)
```

`findClusters` *Suggest clusters based on histograms*

Description

Initial cluster centres are suggested based on the "theta" values of a single marker. Usually called by `getCenters` or `getSpecificCenters`

Usage

```
findClusters(theta, breaks = seq(-0.25, 1.25, 0.05), minBin = 2,
    plot = FALSE)
```

Arguments

`theta`	Numeric vector of polar coordinates angles for a single marker, as given in the `assayData` slot "theta" of objects of class "`AlleleSetIllumina`"
`breaks`	Histogram breakpoints. See `hist`
`minBin`	The minimum peak height below which peaks are set to zero
`plot`	If `TRUE`, histogram is plotted (for testing)

Value

A list containing

`clPeaks`	Suggested cluster centres
`clSizes`	Estimated number of samples in each cluster
`nCl`	Number of clusters

Note

This is a "quick and dirty" way of estimating cluster centres. The function `getCenters` is used as a wrapper to `findClusters` and returns interpreted output after calling the latter function several times with different arguments

Author(s)

Lars Gidskehaug

See Also

`getCenters`, `getSpecificCenters`, `createAlleleSet`

Examples

```
## Not run:
#Read pre-processed data directly into AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],markers=1:10,beadInfo=beadInfo)

#Tune resolution or filter to achieve monomorphic marker
print(findClusters(assayData(BSRed)$theta[1,],plot=TRUE))
print(findClusters(assayData(BSRed)$theta[1,],breaks=seq(-0.25,1.25,0.1),plot=TRUE))
print(findClusters(assayData(BSRed)$theta[1,],minBin=5,plot=TRUE))

#Tune resolution to achieve MSV-5 call
par(mfrow=c(3,1),mai=c(.5,.5,.5,.1))
plot(assayData(BSRed)$theta[2,],assayData(BSRed)$intensity[2,],pch='o')
print(findClusters(assayData(BSRed)$theta[2,],plot=TRUE))
print(findClusters(assayData(BSRed)$theta[2,],breaks=seq(-0.25,1.25,0.04),plot=TRUE))

## End(Not run)
```

`findPolyploidClusters` *K-means clustering*

Description

Wrapper for `kmeans`, allows samples of low presicion to be left out from the clustering and subsequently assigned to clusters

Usage

```
findPolyploidClusters(X, indSE = rep(TRUE, nrow(X)), centers,
    plot = FALSE, wss.update = TRUE, ...)
```

Arguments

`X`	Matrix with data for a single marker to be clustered, with three columns holding "theta", "intensity", and "SE" vectors (in that order) as from the `assayData` slot of an "`AlleleSetIllumina`" object
`indSE`	Logical vector of indexes to samples on which to base the clustering
`centers`	Numeric vector with "theta" starting values for the clustering
`plot`	If `TRUE`, histogram with bins encompassing the initial centre points is plotted
`wss.update`	The within-cluster sums of squares are returned from `kmeans` but not actually used in the genotype calling. If `FALSE`, time is saved by not recalculating the sums of squares after including initially left-out samples in the clusters
`...`	Additional arguments to `hist`

Details

Usually called from within the function callGenotypes or relatives. There the column of intensities is scaled with twice its median value times a scaling factor "rPenalty" (see setGenoOptions) to ensure (by default) relatively higher weight to the "theta" dimension during clustering.

All samples left out from the clustering are subsequently incorporated into the clusters. By leaving out samples of low precision, the resulting clusters may be more accurate.

Value

Object of class "kmeans"

Note

The "Hartigan-Wong" algorithm (see kmeans) is used by default, however this method returns an error if no points are closest to one or more centres. If such an error is returned it will be catched, and a second attempt at clustering will be performed using the "MacQueen" algorithm. A warning will be issued in those cases

Author(s)

Lars Gidskehaug

See Also

callGenotypes, getCenters, kmeans

Examples

```
## Not run:
#Read pre-processed data directly into AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],markers=1:10,beadInfo=beadInfo)

#Generate list of marker categories
gO <- setGenoOptions()
polyCent <- generatePolyCenters(ploidy=gO$ploidy)
print(polyCent)

#Estimate list of likely center points for an MSV-5 marker
ind <- 2
dev.new(); par(mfrow=c(3,1),mai=c(.5,.5,.5,.1))
polyCl <- findClusters(assayData(BSRed)$theta[ind,],
    breaks=seq(-0.25,1.25,0.04),plot=TRUE)
print(polyCl)

#Clustering using all samples
sclR <- median(assayData(BSRed)$intensity[ind,],na.rm=TRUE)*ind*gO$rPenalty
```

```
X <- matrix(cbind(assayData(BSRed)$theta[ind,],
                  assayData(BSRed)$intensity[ind,]/sclR,
                  assayData(BSRed)$SE[ind,]),ncol=3)
clObj <- findPolyploidClusters(X,centers=polyCl$clPeaks,plot=TRUE)
plot(X[,1],X[,2],col=clObj$cluster)
print(clObj)

## End(Not run)
```

`findSeTheta` *Scale pooled standard errors after polar transformation*

Description

After polar transformation of Cartesian intensity values, the estimated standard errors are no longer useful. This function normalizes the standard errors depending on the polar "intensity" value

Usage

```
findSeTheta(pooledSE.raw, R, dist = "manhattan", pNorm = NULL)
```

Arguments

`pooledSE.raw`	Matrix of pooled standard errors of the Cartesian intensities "A" and "B" (see `createAlleleSet`)
`R`	Matrix of polar coordinates intensities
`dist`	Distance measure. See `cart2pol`
`pNorm`	Minkowski norm. See `cart2pol`

Details

Usually called from within `createAlleleSet`. The standard errors of the Cartesian intensities "A" and "B" are not meaningful when the polar coordinates "theta" and "intensity" are plotted on Cartesian axes. In a plot of homoscedastic "B" vs. "A" (see `transformChannels`), the standard error of each bead-type is independent of the signal intensities. In a Cartesian plot of "intensity" vs. "theta", however, bead-types with low intensity will have a large uncertainty, and the precision of the points will increase with increasing intensity. This is beacause the arc-length of the first quadrant semi-circle, which increases with the distance from origin, gets a constant value of unity as the polar coordinates are plotted on Cartesian axes. The pooled standard errors are therefore scaled with the intensity dependent arc-length of the semi-circle between 0 and 90 degrees.

The arc-lengths by which the standard errors are scaled also depend on `dist` and `pNorm`. The circumference of a circle in Manhattan geometry, using a Euclidean metric, is `4*sqrt(2)*R`, and the circumference of a Euclidean circle is `2*pi*R`. It follows that the arc-lengths in the first quadrant only are `sqrt(2)*R` and `pi*R/2`, respectively. The more general arc-length of a Minkowski geometry circle is estimated by numerical integration along the the curve of the super-ellipse between 0 to 90 degrees.

Value

Matrix of transformed standard errors

Author(s)

Lars Gidskehaug

See Also

cart2pol, createAlleleSet

Examples

```
#A single standard error value for points of increasing intensity
R <- .1:10
pooledSE.raw <- 1
pooledSE.theta <- findSeTheta(pooledSE.raw=pooledSE.raw,R=R)
print(pooledSE.theta)
```

---

generatePolyCenters *Generate list of possible genotype categories*

Description

Generates a list of all possible genotype categories for a specific ploidy, including the theoretic allele ratios

Usage

```
generatePolyCenters(ploidy)
```

Arguments

ploidy	One of "di", "tetra", or "tetra.red". The first allows only monomorphics and SNPs, the second allows PSVs and MSVs as well. The option "tetra.red" is the same as "tetra", except "MSV-5" is not included. This may be used (especially for small samples) if many false positive MSV-5 markers are found, and a more restricted classification is needed

Details

Usually called by other functions

Value

List including

centers	List of possible B allele ratios for each cluster category
classification	List of possible cluster categories
size	Numeric vector with the maximum number of clusters within each category

Author(s)

Lars Gidskehaug

See Also

callGenotypes, getCenters, testHardyWeinberg, plotGenotypes

Examples

```
#For genotype calling of tetraploid genomes
polyCent <- generatePolyCenters(ploidy='tetra')
print(polyCent$classification)
```

---

getCenters *Estimate starting points for clustering*

Description

One or several starting points for one or more genotype categories are estimated, given genotype data for a single marker

Usage

```
getCenters(theta, gO = setGenoOptions(),
    breaks = seq(-0.25, 1.25, gO$binWidth),
    polyCent = generatePolyCenters(ploidy = gO$ploidy))

getSpecificCenters(theta, classification, gO = setGenoOptions(),
    breaks = seq(-0.25, 1.25, gO$binWidth),
    polyCent = generatePolyCenters(ploidy = gO$ploidy))
```

Arguments

theta	Numeric vector of "theta"-values for a marker, as given in the assayData slot of "AlleleSetIllumina" objects
gO	List of genotype calling options. See setGenoOptions
breaks	Histogram breakpoints. See hist

polyCent	List of all possible genotype categories with initial centre points for the clustering. See generatePolyCenters
classification	Character string with a single genotype category

Details

Usually called from within other functions. The purpose of getCenters is to suggest a few of the most likely cluster categories and corresponding starting points in ranked order. The function getSpecificCenters returns starting points for a given genotype category

Value

The function getCenters returns a ranked list with elements

ix	Numeric vector with index to categories returned from generatePolyCenters
centers	List of initial centre points of clusters in "theta"-dimension

The function getSpecificCenters returns a numeric vector of clustering starting values

Note

For ploidy="tetra", the function has been empirically tuned to find good starting point for each marker by calling findClusters repeatedly. Other ploidy has not been implemented, but the function will return non-ranked genotype categories with centre points corresponding to theoretical B allele ratios. A warning will be issued to alert the user that the suggested centre points are not optimized or ranked in any way

Author(s)

Lars Gidskehaug

See Also

findClusters, callGenotypes

Examples

```
## Not run:
#Read pre-processed data directly into AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],markers=1:10,beadInfo=beadInfo)

#Generate list of marker categories
gO <- setGenoOptions()
polyCent <- generatePolyCenters(ploidy=gO$ploidy)
print(polyCent)
```

```
#Suggest some candidate categories with initial centre points
ind <- 2
sConf <- getCenters(assayData(BSRed)$theta[ind,],gO=gO,polyCent=polyCent))
print(sConf)
print(polyCent$classification[sConf$ix])

## End(Not run)
```

`getNoiseDistributions` *Estimate median and robust spread of background noise*

Description

The genotypes of [A/T] and [C/G] markers are measured with Infinium I beads using an arbitrary (red or green) channel, and the alternate channel measures noise only. The median and median absolute deviation (mad) of each noise channel is estimated and returned for each array

Usage

```
getNoiseDistributions(BSData, subBeadPool = NULL, normInd,
    normOpts = setNormOptions(), plot = FALSE, newFigure = plot,
    maxPlots = 72, xlim = NULL, ...)

plotEstimatedNoise(BSData, noiseDist,
    normInd = rep(TRUE, nrow(BSData)), normOpts = setNormOptions(),
    newFigure = TRUE, maxPlots = 72, ...)
```

Arguments

`BSData`	"BeadSetIllumina" object, previously subjected to rotation and shearing but not to log- or nth-root transformation.
`subBeadPool`	Vector of one or more 1-digit integers (of class character or numeric) denoting sub-bead pool(s) to base estimations upon. For `subBeadPool = "x"`, the red channel noise is estimated based on markers with `featureData` column `Norm.ID = "20x"` and the green channel noise is based on markers with `Norm.ID = "10x"`. For these channels and markers, only noise is detected. If more than one integer is given, the full set of Infinium I markers corresponding to the specified sub-bead pools is used. If `NULL`, all Infinium I markers are used (default).
`normInd`	Matrix with logical indexes to sub-bead pool for each bead-type. See `getNormInd`
`normOpts`	List specifying pre-processing settings. See `setNormOptions`
`plot`	If `TRUE`, a the estimated and parametrized noise for each channel is plotted
`newFigure`	Logical indicating whether or not to clear the current device before plotting. If `FALSE`, an error will be produced if more than one array is specified

maxPlots	Numeric indicating the maximum allowed number of arrays to plot. Exceeding this limit will produce an error.
xlim	Range of x-axis
...	Additional arguments to plot
noiseDist	Matrix output from getNoiseDistribution

Details

Usually called by preprocessBeadSet.

There are three main groups of markers, as identified by the featureData column "Norm.ID" of BSData. Those identified by a single digit "x" are Infinium II beads, and those identified by three digits "10x" and "20x" are Infinium I beads. The difference between the latter is that "10x" beads are measured using the red channel only, whereas "20x" beads are measured with the green channel only. The background noise can thus be estimated based on the alternate channel.

Value

getNoiseDistribution returns a matrix (noiseDist) holding the median and mad for both channels, all arrays

plotEstimatedNoise is used for its side effects

Warning

Both these functions take non-transformed BSData as input, however transformations are performed inside the functions as specified in normOpts. It follows that noiseDist holds the median and mad of *transformed* data. Use plotEstimatedNoise with caution, as one input parameter is transformed and another is not. The function plotPreprocessing may be a good alternative

Note

Sub-bead pools with different 1-digit identifiers "x" are pooled together whenever more than a single sub-bead pool is given

Author(s)

Lars Gidskehaug

See Also

preprocessBeadSet, plotPreprocessing

Examples

```
## Not run:
#Read files into BeadSetIllumina-object
rPath <- system.file("extdata", package="beadarrayMSV")
BSDataRaw <- readBeadSummaryOutput(path=rPath,recursive=TRUE)

#Find indexes to sub-bead pools
```

```
beadInfo <- read.table(paste(rPath,'beadData.txt',sep='/'),sep='\t',
    header=TRUE,as.is=TRUE)
rownames(beadInfo) <- make.names(beadInfo$Name)
normInd <- getNormInd(beadInfo,featureNames(BSDataRaw))

#Pre-process
normOpts <- setNormOptions(minSize=50,breaks=200)
BSData <- shearRawSignal(BSDataRaw, normOpts = normOpts,plot=TRUE)
noiseDist <- getNoiseDistributions(BSData[,1:4], normInd = normInd,
    normOpts = normOpts, plot = TRUE)
print(noiseDist)
plotEstimatedNoise(BSData,noiseDist,normOpts=normOpts)

## End(Not run)
```

`getNormInd` *Retrieve sub-bead pool indexes*

Description

Creates an array of logical indexes to each bead-type for use in normalization

Usage

```
getNormInd(beadInfo, featureNames, normID = NULL, verbose = TRUE)
```

Arguments

`beadInfo`	Data-frame containg data on each marker in the experiment, specifically the "Address", "Address2", and "Norm.ID" connecting beads to markers (usually exported from Illumina's GenomeStudio Genotyping Module)
`featureNames`	Character vector with bead-type names (addresses)
`normID`	Character vector with one or more sub-bead pool indexes. If `NULL`, all `Norm.ID`'s found in `beadInfo` are used
`verbose`	If `TRUE`, the sum of each sub-bead pool is printed

Value

Matrix of size (`length(featureNames)` x `length(normID)`), with logical indices to the sub-bead pool to which each bead-type belongs

Note

Beads manufactured in different pools should be normalised separately

Author(s)

Lars Gidskehaug

See Also

preprocessBeadSet, getNoiseDistributions

Examples

```
## Not run:
#Common use includes BeadSetIllumina-object
rPath <- system.file("extdata", package="beadarrayMSV")
BSDataRaw <- readBeadSummaryOutput(path=rPath,recursive=TRUE)
beadInfo <- read.table(paste(rPath,'beadData.txt',sep='/'),sep='\t',
    header=TRUE,as.is=TRUE)
rownames(beadInfo) <- make.names(beadInfo$Name)
normInd <- getNormInd(beadInfo,featureNames(BSDataRaw))

## End(Not run)
```

---

getSingleCalls *Identify MSV-5 paralogs with equal genotypes*

Description

Identifies MSV-5 markers for which both paralogs are either "AA", "BB", or "AB"

Usage

```
getSingleCalls(BSRed)
```

Arguments

BSRed	"AlleleSetIllumina" (or "MultiSet") object containing only MSV-5 markers, with an assayData entry "call" (see callGenotypes) and a phenoData column "PedigreeID". The latter contains strings <p><mmm><fff><oo>, where "p", "mmm", "fff", and "oo" are unique identifiers for population, mother, father, and individual within full-sib group, respectively. "000" means founding parent, whereas "999" means unknown parent

Details

For use in other functions, such as unmixParalogues and assignParalogues. May be called initially by user in order to save time in the subsequent calculations. All monomorphic markers ["AA","AA"] and ["BB","BB"], as well as parental markers with genotype ["AB","AB"] are identified.

Value

Matrix of size dim(BSRed) containing calls in {0, 1/2, 1} for the markers in question, NA otherwise

Author(s)

Lars Gidskehaug

See Also

unmixParalogues, assignParalogues, AlleleSetIllumina, callGenotypes

Examples

```
## Not run:
#Read markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling and selection of some MSV-5s
BSRed <- callGenotypes(BSRed)
BSRed <- validateCallsPedigree(BSRed)
iMSV5 <- fData(BSRed)$Classification %in% 'MSV-5' &
    fData(BSRed)$Ped.Errors %in% 0
plotGenotypes(BSRed,markers=which(iMSV5))

#Find markers in question and compare
singleCalls <- getSingleCalls(BSRed[iMSV5,])
print(assayData(BSRed)$call[iMSV5,1:4])
print(singleCalls[,1:4])

## End(Not run)
```

---

locateParalogues *Match paralogs with chromosomes*

Description

Matches patterns of parental inherited alleles within half-siblings between MSV-5 paralogs and genetic map SNPs. The matches for each MSV-5 marker are summed in order to enable mapping of the paralogs to individual chromosomes

Usage

```
locateParalogues(BSSnp, paraCalls, inheritP, offspringLim = 7,
    ratioLim = 0.9)

plotCountsChrom(chromHits, markers = 1:16, ...)
```

Arguments

BSSnp	"AlleleSetIllumina" (or "MultiSet") object containing only SNP markers, with an assayData entry "call" (see callGenotypes) and a phenoData column "PedigreeID". The latter contains strings <p><mmm><fff><oo>, where "p", "mmm", "fff", and "oo" are unique identifiers for population, mother, father, and individual within full-sib group, respectively. "000" means founding parent, whereas "999" means unknown parent. It must also contain a featureData column "Chr.Name"
paraCalls	List with two matrix elements "father" and "mother" containing paralogue calls representing paternal and maternal inherited alleles in offspring, respectively (see unmixParalogues)
inheritP	List with two matrix elements "father" and "mother" containing genetic map marker calls representing paternal and maternal inherited alleles in offspring, respectively (see resolveInheritanceSNP)
offspringLim	In order for a match between a paralogue and a chromosome to be detected, the number of (informative) half-siblings must equal or exceed this numeric value (see setMergeOptions)
ratioLim	The patterns of paternal and maternal inherited alleles among half-sib family offspring are compared between MSV-5 paralogs and genetic map SNPs. The ratio of matching allele patterns between the two must equal or exceed this numeric value in order for a chromosome match to be detected (see setMergeOptions)
chromHits	A numeric array of size (markers x chromosomes x 2) with the average number of matches per chromosome for mothers and fathers separately. Part of the output from locateParalogues.
markers	Index to subset of MSV-5 markers to plot
...	Additional arguments to axis, to be used on the x-axes

Details

The individual paralogs in paraCalls are associated with the genetic map markers in inheritP. If a matching offspring is registered each time an informative allele in the paralogue corresponds with an informative allele in the mapped marker, the degree of association between the two is determined by counting the number of matches. It is not known whether an "A"-allele in the paralogue matches with an "A"- or "B"-allele in the tested marker, but the the combination that produces the highest number of matches is assumed. This means that any pattern of random mis-matches is equally probable as the same number of matches for two unlinked loci. The chance of linkage being falsely declared between two loci however decreases as the number and ratio of matches increase. Associations supported by too few informative meioses are therefore filtered away.

There is a 50% chance of inheriting either allele ("A" or "B") at any segregating locus, which means that a single match is produced by chance 50% of the times. This gives for instance a 6/2^5 (19%) probability that the alleles of two unlinked loci will match for four out of five offspring. Also, as we cannot tell mis-matches from matches, the probability of a false detection is doubled. As such a filter would yield far to many false positives, we need to reduce the probability of random associations further. The default filter counts only markers with at least offspringLim=7 informative meioses and at least ratioLim=90% matches/mis-matches to the paralogue. This threshold implies a random false positive match will occur in 2*11/2^10 (2,1%) of the tests. The total number of matches

across markers within each chromosome is divided by the number of tested markers, such that the chromosomes with the highest average number of matches can be found.

The plots produced by `plotCountsChrom` visualize the average scores produced by `locateParalogues`. A red (fathers) and black (mothers) line is plotted for each MSV-5 marker, with one or two peaks indicating the chromosome(s) the paralogs map to.

Value

The function `locateParalogues` returns a list with elements

`cPerMarker`	A numeric array of size (markers x chromosomes x 2) with the average number of matches per chromosome for mothers and fathers separately
`nCountsTot`	Matrix of size (markers x 2) with the total sum of matches per marker for mothers and fathers

`plotCountsChrom` is used for its side effects

Author(s)

Lars Gidskehaug

See Also

`plotCountsChrom`, `setMergeOptions`, `unmixParalogues`, `resolveInheritanceSNP`, `MultiSet`, `AlleleSetIllumina`, `assignParalogues`

Examples

```
## Not run:
#Read markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling and splitting of MSV-5 paralogs
BSRed <- callGenotypes(BSRed)
BSRed <- validateCallsPedigree(BSRed)
iMSV5 <- fData(BSRed)$Classification %in% 'MSV-5' &
    fData(BSRed)$Ped.Errors %in% 0
paraCalls <- unmixParalogues(BSRed[iMSV5,])

#Genetic map SNPs and inherited parental alleles
iSNP <- fData(BSRed)$Classification %in% 'SNP' &!
    is.na(fData(BSRed)$Chromosome)
inheritP <- resolveInheritanceSNP(BSRed[iSNP,])

#Match paralogs with map
```

```
chromHits <- locateParalogues(BSRed[iSNP,],paraCalls,inheritP)

#The example data and map are too small to detect most homeologies
plotCountsChrom(chromHits$cPerMarker,1:sum(iMSV5),at=1:15,
    labels=dimnames(chromHits$c)[[2]],las=2)

## End(Not run)
```

`makeDiploidCalls` *Constrain calls to diploid representation*

Description

Input calls for any diploid or tetraploid genome marker are restricted to diploid representation. This implies that non-segregating paralogs are ignored or set to missing, and non-resolvable paralogs are set to missing

Usage

```
makeDiploidCalls(calls, fData)
```

Arguments

`calls`	Numerical matrix with dimensions (markers x samples). The allele ratios are from the set {0, 1/4, 1/2, 3/4, 1}
`fData`	`featureData` data-table such as from an "`AlleleSetIllumina`" object, with rows corresponding to rows in `calls` and containing the column "Classification" (see `callGenotypes`). A column "Manual.Calls.R" will be used if present (see `callGenotypes.interactive`)

Details

Usually called by the function `translateTheta`

Value

A numerical matrix with the same dimensions as `calls`, however with values restricted to {0, 1/2, 1}

Author(s)

Lars Gidskehaug

See Also

`translateTheta`, `AlleleSetIllumina`

Examples

```
#Construct tiny example data
calls <- c(0,.5,1,0,.25,.5,.5,.75,1,.5,.5,.5,0,.25,.75)
categories <- c('SNP','MSV-a','MSV-b','PSV','MSV-5')
calls <- matrix(calls,nrow=5,byrow=TRUE,
    dimnames=list(categories,paste('S',1:3,sep='')))
fData <- data.frame(Classification=categories,row.names=categories)
print(calls)

#Make diploid
diploidCalls <- makeDiploidCalls(calls, fData)
print(diploidCalls)
```

`makeFilenames` *Generate filenames reflecting normalizations*

Description

Filenames including paths are generated to enable writing a set of "`AlleleSetIllumina`" datatables to text-files. These are used subsequently to read specified markers and arrays into the workspace

Usage

```
makeFilenames(tag, normOpts, path = ".")

modifyExistingFilenames(dataFiles, oldtag, newtag)
```

Arguments

`tag`	Character string included in the filenames which is unique to the current analysis
`normOpts`	List with pre-processing options used (see `setNormOptions`). Generated filnames will reflect the transformation and channel-normalisation performed on the data
`path`	Character string with the saving path
`dataFiles`	Character vector with previous output from `makeFilenames`
`oldtag`	Often several rounds of genotype-calling is performed based on the same, preprocessed data. This character string correspond to the unique tag used for the previous run(s).
`newtag`	A new character string tag to replace `oldtag` in files other than those preprocessed data-files which remain the same between runs.

Details

These files generate file names used for reading and writing between text files and workspace. When different runs of genotype calling is performed using the same, pre-processed data, `modifyExistingFilenames` updates only the elements which change with the new genotyping.

Value

Character vector with the fields

intFile	File name for intensity values
thFile	File name for theta values
seFile	File name for SE values
phFile	File name for phenotype data
callFile	File name for call values
resFile	File name for feature data such as call statistics
genoFile	File name for allele values

Warning

Warnings are issued if existing filenames are used, as this may lead to overwriting of these files in subsequent steps (no files are overwritten by the functions themselves)

Author(s)

Lars Gidskehaug

See Also

writeAlleleSet, createAlleleSetFromFiles, createMultiSetFromFiles, translateThetaFromFiles

Examples

```
#Create names
normOpts <- setNormOptions()
tag <- '1'
dataFiles <- makeFilenames(tag,normOpts)
print(dataFiles)

#Make new names for subsequent, alternate genotype calling runs
dataFiles <- modifyExistingFilenames(dataFiles,tag,'2')
```

---

manualCall *Interactive calling of genotype for single marker*

Description

Function usually called from within callGenotypes.interactive, in order to define clusters by clicking and dragging with the mouse

Usage

```
manualCall(marker, cntIdeal, classification, gg = NULL, close.gg = TRUE)
```

Arguments

marker	Data-frame containing the columns "Theta", "R", and optionally "PedCheck" and "PedigreeID" for a single marker. The first two correspond to assayData entries "theta" and "intensity" of an "AlleleSetIllumina" object. The "PedCheck" column corresponds to the output from validateSingleCall. The "PedigreeID" consists of strings <p><mmm><fff><oo>, where "p", "mmm", "fff", and "oo" are unique identifiers for population, mother, father, and individual within full-sib group, respectively. "000" means founding parent, whereas "999" means unknown parent. May also include a vector of B allele ratios "Call" with results from a previous clustering, in which case this is used as starting values
cntIdeal	A numeric vector of the allowed B allele ratios for a specific genotype category (see generatePolyCenters)
classification	Character string denoting genotype category (see generatePolyCenters)
gg	An instance of "ggobi"
close.gg	If TRUE, an updated data-frame marker is returned and gg is closed. Otherwise, gg is returned directly

Details

A "GGobi" interactive scatter-plot is produced. Round dots with colours purple, pink, red, blue, green and grey denote samples of "Theta" values 0, 1/4, 1/2, 3/4, 1, and NA, respectively. Orange and brown square dots indicate offspring and parent pedigree errors, respectively. Select ("brush") points by moving around the yellow rectangle visible on the screen using the left mouse button. Change the shape of the rectangle using the right mouse button.

If pedigree errors are found after clustering, a warning is issued, and the user is given the choice between un-assigning erroneous offspring, modifying the clusters, or disregarding the errors. Note that by setting erroneous samples to missing, the remaining calls may appear better than they are.

Value

Depending on the value of close.gg, a data-frame or an object of class "ggobi" is returned, containing marker data including a column "Call" with the B allele ratio for each subject

Author(s)

Lars Gidskehaug

See Also

ggobi, callGenotypes.interactive

Examples

```
## Not run:
#Read 10 markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
```

```
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],markers=1:10,
    beadInfo=beadInfo)

#Prepare a single marker
ind <- 2
marker <- data.frame(Theta=assayData(BSRed)$theta[ind,],
    R=assayData(BSRed)$intensity[ind,],
    PedigreeID=pData(BSRed)$PedigreeID,
    stringsAsFactors=FALSE)

#Cluster marker from scratch, assuming MSV-5
polyCent <- generatePolyCenters(ploidy="tetra")
iMSV5 <- 7
marker1 <- manualCall(marker,cntIdeal=polyCent$centers[[iMSV5]],
    classification=polyCent$classification[[iMSV5]],close.gg=FALSE)

## End(Not run)
```

`normalizeShearedChannels` *Channel normalization*

Description

Normalizes the red and green channels of each array

Usage

```
normalizeShearedChannels(trChannel, noiseDist,
    normOpts = setNormOptions())
```

Arguments

`trChannel`	A list with at least two matrices, "X" and "Y", containing red and green intensities, respectively. The dimensions of the matrices are (markers x samples)
`noiseDist`	A matrix of dimension (samples x 4), containing the estimated median and mad of each channel. See `getNoiseDistributions`
`normOpts`	List of pre-processing settings. See `setNormOptions`

Details

Depending on `normOpts$method`, the channels on each array are "quantile", "medianAF", or "linPeak" normalized (see `setNormOptions`). For quantile normalization, the **limma** package (Smyth and Speed, 2003) is used (see `normalizeQuantiles`). For "medianAF", the red channel is scaled such that `median(R/(R+G))` is close to one half. This is similar to the normalization suggested in Macgregor *et al.* (2008). If "linPeak" is chosen, both channels are linearly scaled by the quantile defined in `normOpts$scale`.

Value

A list similar to `trChannel` is returned with normalized data. A character string "method" is added to the list describing the normalization performed

Note

Though quantile normalization is often reported to work well, it represents a quite drastic form of manipulation of the data. It assumes that any difference in distribution between the channels is related to technical measurement error rather than biological variation. By forcing the data into identical distributions, there is a risk that some samples are forced into wrong clusters in the subsequent genotype calling. We therefore advice to use quantile normalization with caution, and only if it is seen to have a good effect on subsequent clustering.

Author(s)

Lars Gidskehaug

References

S. Macgregor *et al.* (2008) Highly cost-efficient genome-wide association studies using DNA pools and dense SNP arrays. *Nucleic Acids Res.* **36**(6):e35

G. K. Smyth and T. P. Speed (2003) Normalization of cDNA microarray data. *Methods* **31**:265-27

See Also

`preprocessBeadSet`, `normalizeQuantiles`

Examples

```
## Not run:
#Read files into BeadSetIllumina-object
rPath <- system.file("extdata", package="beadarrayMSV")
BSDataRaw <- readBeadSummaryOutput(path=rPath,recursive=TRUE)

#Find indexes to sub-bead pools
beadInfo <- read.table(paste(rPath,'beadData.txt',sep='/'),sep='\t',
    header=TRUE,as.is=TRUE)
rownames(beadInfo) <- make.names(beadInfo$Name)
normInd <- getNormInd(beadInfo,featureNames(BSDataRaw))

#Pre-process 1 array
normOpts <- setNormOptions(minSize=10)
BSData <- shearRawSignal(BSDataRaw, normOpts = normOpts)
noiseDist <- getNoiseDistributions(BSData, normInd = normInd,
    normOpts = normOpts)
trChannel <- transformChannels(assayData(BSData)$R,
    assayData(BSData)$G, normOpts = normOpts)
mafData <- normalizeShearedChannels(trChannel, noiseDist,
    normOpts = normOpts)
quantData <- normalizeShearedChannels(trChannel, noiseDist,
```

```
    normOpts = setNormOptions(method='quantNorm'))

#Plot distributions
dev.new()
par(mfrow=c(3,2))
hist(trChannel$X,breaks=30,col='red',main='Red channel')
hist(trChannel$Y,breaks=30,col='green',main='Green channel')
hist(mafData$X,breaks=30,col='red',main='medianAF')
hist(mafData$Y,breaks=30,col='green',main='medianAF')
hist(quantData$X,breaks=30,col='red',main='quantNorm')
hist(quantData$Y,breaks=30,col='green',main='quantNorm')

## End(Not run)
```

`plotGenotypes` *Plotting of genotyped markers*

Description

Produces plots of "intensity" vs. "theta" with clusters coloured according to genotype call. Optional highlighting of specified samples or of pedigree-errors

Usage

```
plotGenotypes(BSRed, markers = 1:min(nrow(BSRed), 64),
    indHighlight = NULL, ploidy = "tetra", indicate.SE = FALSE,
    retFrames = FALSE, nC = NULL, mai = NULL, mNoise = NULL,
    main = NULL)
```

Arguments

`BSRed`	"`AlleleSetIllumina`" object
`markers`	Index to markers to plot
`indHighlight`	Index to samples to highlight with yellow
`ploidy`	Character string reflecting the ploidy (copy-number) of the markers, defining the appearance of the clusters. Currently, only "tetra" is implemented
`indicate.SE`	If `TRUE`, size of points will reflect their precision (1/SE)
`retFrames`	If `TRUE`, a list of data-frames containing the coordinates for each marker is returned
`nC`	Number of columns of plots in the figure
`mai`	Size of margins. See `par`
`mNoise`	The "intensity"-limit below which no samples are called. If `NULL`, the mean `pData(BSRed)$noiseIntensity` (across arrays) is used
`main`	Vector of plot-titles. If `NULL`, the `featureNames` are used

Details

All specified markers are plotted in subplots on a single device, and no more than 50-100 markers should be plotted at the time. The classified samples are coloured according to their allele-ratio, and non-called samples are represented as black dots. A red line indicates the estimated noise-level.

There are several ways samples may be highlighted. If `indicate.SE` is `TRUE`, the size of the dots reflects the precision of the intensity-estimates. Else, the samples indicated with `indHighlight` are represented as yellow dots. If neither of these are given, pedigree-errors will be highlighted if such information is found in `BSRed`. Both `assayData` entries "ped.check.parents" (see `validateSingleCall`) and "ped.check" (see `validateCallsPedigree`) are recognised. Yellow circles and crosses indicate parent and offspring errors, respectively.

Value

The function `plotGenotypes` is called mainly for its side effects, however if `retFrames` is `TRUE` a list of data-frames containing the coordinates for each marker is returned

Author(s)

Lars Gidskehaug

See Also

`validateCallsPedigree`, `callGenotypes`

Examples

```
## Not run:
#Read pre-processed data directly into AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling and plotting
BSRed <- callGenotypes(BSRed)
plotGenotypes(BSRed,1:25)

#Pedigree validation and plotting
BSRed <- validateCallsPedigree(BSRed)
plotGenotypes(BSRed,1:25)

## End(Not run)
```

preprocessBeadSet *Pre-processing of BeadSetIllumina objects*

Description

Performs a sequence of pre-processing routines on objects of class "BeadSetIllumina"

Usage

```
setNormOptions(shearInf1 = TRUE, transf = "root",
    method = "medianAF",
    minSize = suggestSh(shearInf1)$minSize,
    prob = suggestSh(shearInf1)$prob,
    nBins = suggestSh(shearInf1)$nBins,
    dist = suggestTr(transf)$dist,
    pNorm = suggestTr(transf)$pNorm,
    nthRoot = suggestTr(transf)$nthRoot,
    offset = suggestTr(transf)$offset,
    scale = suggestNo(method)$scale,
    nSD = 3, breaks = 200)

plotPreprocessing(BSData, normInd,
    normOpts = setNormOptions(shearInf1 = !is.null(normInd)),
    plotArray = 1, ...)

preprocessBeadSet(BSData, normInd,
    normOpts = setNormOptions(shearInf1 = !is.null(normInd)))
```

Arguments

shearInf1	If TRUE, only the signal-containing channel of Infinium I beads are used to define the homozygote asymptotes for the affine transformation (rotation and shearing). This may be more accurate than using all beads, as the variation along the perpendicular axis is small
transf	Character string denoting transformation. One of "none", "log" (base 2), or "root" (defined by nthRoot)
method	Character string denoting channel normalization method for each array. One of "none", "quantNorm", "medianAF", or "linPeak". For quantile normalization, the **limma** package is required (Smyth and Speed, 2003). For "medianAF", the red channel is scaled such that median(R/(R+G)) is close to one half. If "linPeak" is chosen, both channels are linearly scaled by its scale'th quantile
minSize	The homozygote asymptotes are found by drawing a straight line through quantile points distributed in bins along each axis. Only bins containing more than minSize points are used
prob	Numeric probabiliy used in the quantile-function, defining the points through which the asymptotes are drawn

nBins	The number of bins into which to divide the points along each axis before the homozygote asymptotes are drawn
dist	Character string defining the distance measure used for polar coordinates transformation of the signal. One of "manhattan", "euclidean", or "minkowski". See cart2pol
pNorm	See cart2pol
nthRoot	Numeric used together with transf="root"
offset	A numeric offset added to each channel before transformation. Values below zero are set to NA during log- or root-transformation
scale	Used with method="linPeak"
nSD	The background signal is estimated as nSD times the estimated standard measurement error (found from the the parameterised noise levels for each channel)
breaks	The parameterisation of noise levels is based on a histogram of each channel, where the numeric breaks defines the smoothing (number of bins). See hist
BSData	"BeadSetIllumina" object not previously pre-processed
normInd	Matrix with logical indexes to sub-bead pool for each bead-type. See getNormInd
normOpts	List output from setNormOptions
...	Further arguments to plotEstimatedNoise
plotArray	Numeric index to a single array to plot

Details

Using setNormOptions, default pre-processing options are suggested, and any changes may be specified. The effects of different options are studied using plotPreprocessing for a number of arbitrary arrays. This produces four plots; i) raw data scatter, ii) scatter including the estimated asymptotes for the affine transformation (red/green) including the quantile points used (blue dots), iii) the noise levels for the red and green channel after transformation, parameterized signal superimposed, based on the non-signal channels of Infinium I beads, and iv) scatter after transformation including new axes (green) and estimated noise levels (red dots).

For the affine transformation, it is important that enough quantile points are included to get reliable asymptotes. If there are few blue dots in plot ii), decrease the minSize option or set shearInf1 to FALSE. If the grey lines in plot iii) are too coarse (too few points) to get a good noise-parameterisation, increase breaks. Note also how the noise levels are affected by different transformations.

Pay close regard to how the transformation affects the shapes of the clouds in plot iv). Ideally, three well defined clouds protrude from the estimated origin, corresponding to the homozygotes which fall on the estimated axes and the heterozygotes which fall 45 degrees in between. Imagine a rubber band stretched over the ends of the three clouds. If the rubber band is straight (no transformation), the "manhattan" (or 1-norm "minkowski") distance is the best option for polar coordinates. If the three points fall on a circle, the "euclidean" (or 2-norm "minkowski") distance is the best option. If the rubber band forms a shape intermediate between a circle and a square (e.g. 4th-root transformation), the 5-norm "minkowski" distance or similar may the best choice.

The function preprocessBeadSet calls several pre-processing routines in sequence. First shearRawSignal performs the affine transformations, then getNoiseDistributions estimates the distributions of the noise for each channel. Next, transformChannels transforms the signal, followed by transformation of the standard errors of each channel using transformSEs. In the end, normalizeShearedChannels performs channel normalisation for each array.

Value

Output from setNormOptions is a list with pre-processing options

The function plotPreprocessing is used for its side effects

Output from preprocessIllumina is a "BeadSetIllumina" object with pre-processed assayData entries. A column "noiseIntensity" is added to phenoData, this is the (parameterized) standard error times nSD

Note

If BSData contains a phenoData column "noiseIntensity", preprocessBeadSet assumes the data are already normalized and an error is produced

Author(s)

Lars Gidskehaug

References

G. K. Smyth and T. P. Speed. (2003) Normalization of cDNA microarray data. *Methods* **31**:265-27

See Also

readBeadSummaryOutput, getNormInd, shearRawSignal, getNoiseDistributions, transformChannels, transformSEs, normalizeShearedChannels, createAlleleSet, BeadSetIllumina

Examples

```
## Not run:
#Read files into BeadSetIllumina-object
rPath <- system.file("extdata", package="beadarrayMSV")
BSDataRaw <- readBeadSummaryOutput(path=rPath,recursive=TRUE)

#Find indexes to sub-bead pools
beadInfo <- read.table(paste(rPath,'beadData.txt',sep='/'),sep='\t',
    header=TRUE,as.is=TRUE)
rownames(beadInfo) <- make.names(beadInfo$Name)
normInd <- getNormInd(beadInfo,featureNames(BSDataRaw))

#Pre-process
normOpts <- setNormOptions(minSize=10)
plotPreprocessing(BSDataRaw,normInd,normOpts,plotArray=1)
BSData <- preprocessBeadSet(BSDataRaw,normInd,normOpts)
pData(BSData)

## End(Not run)
```

readBeadSummaryOutput *Read bead-summary intensities from two colour Illumina (Infinium) scanner*

Description

Reads text-files with bead summary output for each array and arranges the data in a "BeadSetIllumina" object

Usage

```
readBeadSummaryOutput(arrayNames = NULL, path = ".",
    pattern = "beadTypeFile.txt", recursive = FALSE, sep = ",",
    fullPaths = NULL, sepchar = "_", prList = NULL)
```

Arguments

arrayNames	Character vector containing names of arrays to be read. If fullPaths is specified, arrayNames searches only among these, otherwise it searches in current working directory. More commonly arrayNames is NULL, then all arrays in current working directory or as specified in fullPaths are read
path	Character string specifying the data-directory
pattern	Character string specifying the file-name ending of the files to be read
recursive	If fullPaths is not specified, the logical recursive regulates whether or not the function should look for files recursively
sep	Delimiter in text-files
fullPaths	Character vector containing the names of the files of interest, including the directories in which they are found (from the working directory path). Typically on a form similar to <chip>/<chip>_<row>_<pattern>. Useful when the current directory contains arrays from several experiments/families, and only a subset is to be loaded at the time. Usually used in combination with arrayNames=NULL
sepchar	Character used to bind togeher different parts of the file-names ("_" in the above example)
prList	Character vector with Illumina probe-IDs. Should be specified in the rare case that different arrays contain different probes.

Details

The scanner protocol must be set to save bead-summary data. The function expects the following data-fields in each file: "Illumicode", "N", "Mean GRN", "Dev GRN", "Mean RED", "Dev RED" (in that order). The first two are the bead-type ID and the number of detected beads. The rest contain a robust bead-type mean and the standard deviation of the signal for each channel.

The probes found in the first loaded array define prList unless otherwise specified. A warning is issued if additional probes are found in a subsequent array. It is an error if not all markers in prList are found in subsequent arrays.

Much of the functionality is adapted from the **beadarray** package (Dunning *et al.*, 2007). This package is an excellent resource for loading and analysing raw Illumina BeadArray data, including images.

Value

"BeadSetIllumina" object, with the assayData entries

R	Mean red signal ("Mean RED")
se.R	Standard error of the mean red signal ("Dev RED"/sqrt("N"))
G	Mean green signal ("Mean GRN")
se.G	Standard error of the mean green signal ("Dev GRN"/sqrt("N"))
no.beads	Number of detected beads

Author(s)

Lars Gidskehaug

References

M. J. Dunning *et al.* (2007) beadarray: R classes and methods for Illumina bead-based data. *Bioinformatics* **23**(16):2183-4

See Also

BeadSetIllumina, scatterArrays, preprocessBeadSet, createAlleleSet

Examples

```
## Not run:
#Read a BeadSetIllumina object using files in example data directory
rPath <- system.file("extdata", package="beadarrayMSV")
BSDataRaw <- readBeadSummaryOutput(path=rPath,recursive=TRUE)

#Print information from object
BSDataRaw
pData(BSDataRaw)
varMetadata(BSDataRaw)

#Alternatively:
sampleFile <- paste(rPath,'sampleData.txt',sep='/')
sampleInfo <- read.table(sampleFile,skip=8,sep='\t',header=TRUE,
    colClasses='character')
rownames(sampleInfo) <- make.names(paste(sampleInfo$chip,
    sampleInfo$row,sep='_'))
pattern <- 'beadTypeFile.txt'
fullPaths <- paste(sampleInfo$chip,'/',sampleInfo$chip,'_',
    sampleInfo$row,'_',pattern,sep='')
BSDataRaw <- readBeadSummaryOutput(fullPaths=fullPaths[1:4],
    path=rPath,pattern=pattern)
```

```
## Plot G vs. R
dev.new()
scatterArrays(BSDataRaw,smooth=FALSE)

## End(Not run)
```

`resolveInheritanceSNP` *Track parental alleles in offspring*

Description

Based on parental and offspring SNP marker genotypes, the paternal and maternal alleles in offspring are deduced

Usage

```
resolveInheritanceSNP(BSSnp)
```

Arguments

BSSnp	"AlleleSetIllumina" (or "MultiSet") object containing only SNP markers, with an assayData entry "call" (see callGenotypes) and a phenoData column "PedigreeID". The latter contains strings <p><mmm><fff><oo>, where "p", "mmm", "fff", and "oo" are unique identifiers for population, mother, father, and individual within full-sib group, respectively. "000" means founding parent, whereas "999" means unknown parent

Details

Although a linkage map is not needed for this function, it was written for the purpose of mapping MSV-5 paralogs to a map represented by the markers in BSSnp (see assignParalogues). For its intended use, BSSnp should therefore contain chromosome positions and genetic distances in featureData

Value

A list with elements

mother	Matrix of calls representing maternal inherited alleles
father	Matrix of calls representing paternal inherited alleles

Note

Offspring alleles from monomorpic parents are sometimes missing even though the inherited alleles can be directly deduced from the parental genotype. This is for efficiency of the algorithm, as monomorphic parents carry no linkage information

Author(s)

Lars Gidskehaug

See Also

AlleleSetIllumina, callGenotypes, assignParalogues, unmixParalogues

Examples

```
## Not run:
#Read markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling and selection of SNPs for linkage map
BSRed <- callGenotypes(BSRed)
BSRed <- validateCallsPedigree(BSRed)
iSNP <- fData(BSRed)$Classification %in% 'SNP' &! is.na(fData(BSRed)$Chromosome)

#Find informative alleles inherited from each parent
inheritP <- resolveInheritanceSNP(BSRed[iSNP,])

#Compare a few markers for an arbitrary, single triplet
iOffspring <- 1
iFather <- which(pData(BSRed)$PedigreeID %in% pData(BSRed)$Parent2[iOffspring])
iMother <- which(pData(BSRed)$PedigreeID %in% pData(BSRed)$Parent1[iOffspring])
print(assayData(BSRed)$call[iSNP,c(iMother,iFather,iOffspring)][6:10,])
print(inheritP$mother[6:10,c(iMother,iFather,iOffspring)])
print(inheritP$father[6:10,c(iMother,iFather,iOffspring)])

## End(Not run)
```

---

scatterArrays *Cartesian scatter-plots of two-colour intensities*

Description

Scatter-plots for a series of arrays are produced. Which intensities are plotted depends on whether the input BSData is an instance of "BeadSetIllumina" or "AlleleSetIllumina"

Usage

```
scatterArrays(BSData, arrays, markers = seq(1, nrow(BSData), 5),
    smooth = TRUE, newFigure = TRUE, maxPlots = 72, ...)
```

Arguments

BSData	"BeadSetIllumina" object (intensities "G" vs. "R" are plotted) or "AlleleSetIllumina" object (intensities "B" vs. "A" are plotted)
arrays	Indexes to arrays
markers	Indexes to markers
smooth	If TRUE, density-plots are used (see smoothScatter)
newFigure	Logical indicating whether or not to clear the current device before plotting. If FALSE, an error will be produced if more than one array is specified.
maxPlots	Numeric indicating the maximum allowed number of arrays to plot. Exceeding this limit will produce an error.
...	Additional graphical input parameters

Value

This function is called for its side effects

Author(s)

Lars Gidskehaug

See Also

BeadSetIllumina, AlleleSetIllumina, plotPreprocessing, plotGenotypes

Examples

```
## Not run:
#Read a BeadSetIllumina object using files in example data directory
rPath <- system.file("extdata", package="beadarrayMSV")
BSDataRaw <- readBeadSummaryOutput(path=rPath,recursive=TRUE)

## Plot G vs. R
dev.new()
scatterArrays(BSDataRaw,smooth=FALSE)

## End(Not run)
```

---

shearRawSignal *Affine transformation of axes*

Description

Mimicks the rotation and shearing of raw two-colour intensities performed in Illumina's GenomeStudio. To be performed before any other transformation or normalisation.

Usage

```
shearRawSignal(BSData, plot = FALSE, newFigure = plot,
    normOpts = setNormOptions(), maxPlots = 72, ...)

normalizeIllumina(rawData, indTrain = rep(TRUE, length(rawData$x)),
    normOpts = setNormOptions(), plot = FALSE, xylim = NULL,
    verbose = FALSE)
```

Arguments

BSData	"BeadSetIllumina" object
plot	If TRUE, a scatter-plot including the estimated homozygote asymptotes is produced for each array in BSData
newFigure	Logical indicating whether or not to clear the current device before plotting. If FALSE, an error will be produced if more than one array is specified
normOpts	List with at least three elements: "nBins", "minSize", and "prob". See setNormOptions
maxPlots	Numeric indicating the maximum allowed number of arrays to plot. Exceeding this limit will produce an error
...	Additional input parameters to normalizeIllumina
rawData	List containing the numeric vectors "x" and "y" with red and green intensity data for a single array
indTrain	Logical index vector specifying which points to base the normalization upon. It may also be a list containing two logical index vectors "10r" and "20g", containing indexes to all sub-bead pools with red and green Infinium I beads, respectively. In the latter case, the "10r"-points are used to fit the "AA" homozygote asymptote and the "20g"-points are used to fit the "BB" homozygote asymptote
xylim	If plot, these are the c(xLow,xHigh,yLow,yHigh) coordinates of the axes
verbose	If TRUE, the offset and slope of both asymptotes are printed

Details

shearRawSignal is usually used as a wrapper for normalizeIllumina. Even more commonly, preprocessBeadSet is used as a wrapper for both.

These functions perform an affine transformation which is similar to in Illumina's software (Peiffer *et al.*, 2006), and it differs only in how the homozygote asymptotes are estimated. Both axes are divided into a number of bins as specified by normOpts$nBins. The points specified by the quantile normOpts$prob in each bin exceeding normOpts$minSize points are used for a least squares fit of homozygote asymptotes. If normOpts$shearInf1 is TRUE, only Infinium I beads are used, and the quantiles are based on all points in the bins. Otherwise, all points are used, however the quantile is based on the normOpts$minSize smallest values only.

In the plot, the red and green lines give the estimated "AA" and "BB" homozygote asymptotes, respectively. The blue dots are the points upon which the rotation and shearing is based.

Value

A "BeadSetIllumina" object from shearRawSignal, or a list containing normalized signal from normalizeIllumina. The normalized red and green signal is transformed such that the "AA" and "BB" homozygotes asymptotes are perpendicular to each other

Author(s)

Lars Gidskehaug

References

D. A. Peiffer *et al.* (2006) High-resolution genomic profiling of chromosomal aberrations using Infinium whole-genome genotyping, *Genome Res.* **16**:1136-1148

See Also

preprocessBeadSet, setNormOptions

Examples

```
#Make artificial, heteroscedastic data
x1 <- 5 + exp(rnorm(1000))*100
y1 <- 100 + x1*.1 + x1*rnorm(1000,sd=.1)*.2
y2 <- 100 + exp(rnorm(1000))*70
x2 <- (y2-5)/10 + (y2-100)*rnorm(1000,sd=.1)*.2
rawData <- list()
rawData$x <- c(x1,x2)
rawData$y <- c(y1,y2)

#Affine transformation
normData <- normalizeIllumina(rawData,plot=FALSE,verbose=TRUE)

## Not run:
#Affine transformation with plotting
dev.new()
normOpts <- setNormOptions(minSize=10)
normData <- normalizeIllumina(rawData,normOpts=normOpts,plot=TRUE,verbose=TRUE)

#After rotation and shearing
dev.new()
plot(normData$x,normData$y,pch='.',main='Affine transformation',xlab='R',ylab='G')
abline(v=0,h=0,col='blue')

## End(Not run)
```

`testHardyWeinberg` *Test for Hardy-Weinberg equilibrium*

Description

For SNP and MSV markers, the sample is tested for being in Hardy-Weinberg (HW) equilibrium. Estimated B allele frequencies are returned

Usage

```
testHardyWeinberg(sizes, bestConf,
    polyCent = generatePolyCenters(ploidy="di"),
    afList = seq(0, 0.5, 0.05))
```

Arguments

`sizes`	Numeric vector with the size of each cluster in the current genotype category
`bestConf`	Character string with genotype category, or numeric index to correct genotype category in `polyCent`
`polyCent`	List with all genotype categories and the allele ratios corresponding to the different clusters. See `generatePolyCenters`
`afList`	Numeric vector of values within [0, 0.5], denoting which B allele frequencies to test in the case of two segregating paralogs (see below)

Details

The null hypothesis of the test is that the population is in HW frequencies. As most populations deviate more or less from HW equilibrium, strict control of this test is usually not recommended. Still, it can a very powerful test to detect failed clustering, by using a sufficiently low significant level to account for naturally deviating samples. If the sample contain subjects from different populations however, the power of the test may be very low.

At duplicated loci, the observed B allele frequency (BAF) is in fact the mean BAF across both paralogues. For MSV-5 markers, which are segregating in both paralogs, the individual BAFs may be estimated assuming different candidate values at one paralogue. Several values of BAF for one paralogue are set in `afList`, such that the BAF for the other paralogue is given. A value of `0.5` means the BAF is the same at both loci. All values are tested for HW, and the most likely BAF at both paralogs are those resulting in the highest p-value. The accuracy of these estimates increases with the degree of HW equilibrium.

Value

Data-frame with the chi-square test statistic, the degrees of freedom, p-value, and both estimated B allele frequencies

Author(s)

Lars Gidskehaug

See Also

callGenotypes

Examples

```
#Arbitrary MSV-5 marker
sizes <- c(30, 200, 1600, 700, 400)
polyCent <- generatePolyCenters(ploidy='tetra')
HWstats <- testHardyWeinberg(sizes,'MSV-5',polyCent=polyCent)
print(HWstats)
```

---

transformChannels *Signal transformation*

Description

Signal and standard error estimates are subjected to log or nth-root transformation

Usage

```
transformChannels(X, Y = NULL, normOpts = setNormOptions())

transformSEs(X, se.X, normOpts = setNormOptions())
```

Arguments

X	Matrix of (mean bead type) intensity data for red or green channel
Y	Matrix of intensity data for the alternate channel
normOpts	List specifying pre-processing settings. See setNormOptions
se.X	Matrix holding the standard error of the mean bead intensities

Details

The channel(s) are transformed according to the specifications in normOpts.

As signal intensities are estimated as (robust) means within bead-types on an array, the estimated standard errors are in fact standard errors of means. The standard errors of means cannot be log or nth-root transformed in order to get the standard error of the transformed signal mean. To avoid loading the bead-level data into the workspace, the function transformSEs estimates the transformed standard errors using a first order Taylor-expansion around the mean. In our experience, these estimates are accurate except for very low signal intensities (below noise level).

Value

The output from `transformChannels` or `transformSEs` is a list containg at least two of the following items

`X`	Transformed X
`Y`	Transformed Y
`SE`	Estimated standard errors of transformed signal mean
`lstr`	Character string with description of transformation

Author(s)

Lars Gidskehaug

See Also

`preprocessBeadSet`

Examples

```
#Simulate intensity data
X <- matrix(exp(rnorm(4000))*100,nrow=1000,ncol=4,
    dimnames=list(1:1000,paste('Array',1:4)))
Y <- matrix(exp(rnorm(4000))*70,nrow=1000,ncol=4,
    dimnames=list(1:1000,paste('Array',1:4)))

#Transform signal
normOpts <- setNormOptions(offset=0)
trChannel <- transformChannels(X,Y,normOpts)

## Not run:
#Plot one array before and after transformation
dev.new()
plot(X[,1],Y[,1],pch='o',main='Raw data')
dev.new()
plot(trChannel$X[,1],trChannel$Y[,1],pch='o',
    main=paste(trChannel$lstr,'transformed data'))

## End(Not run)

#Simulate a single bead type represented by 12 beads
beadLevelX <- rnorm(12,mean=800,sd=10)
sd.X <- sd(beadLevelX)
X <- mean(beadLevelX)
se.X <- sd.X/sqrt(length(beadLevelX))

#Transformed signal (4th-root)
transfX <- mean(beadLevelX^(1/4))   #true value
print(transfX)
transfX.1 <- X^(1/4)                #good approximation
print(transfX.1)
```

```
#Transformed standard error (4th-root)
transfSE <- sd(beadLevelX^(1/4))/sqrt(length(beadLevelX))  #true value
print(transfSE)
transfSE.1 <- se.X^(1/4)                                   #bad approximation
print(transfSE.1)
transfSE.2 <- transformSEs(X,se.X,normOpts=normOpts)       #good approximation
print(transfSE.2$SE)
```

`translateTheta` *Convert genotype calls to allele information*

Description

Genotype calls represented as numeric values (allele ratios) within [0, 1] are converted to character strings containing allele information "A", "T", "C", and "G"

Usage

```
translateTheta(calls, resInfo, type = "regular")

translateThetaCombined(BSRed, mergedCalls = NULL)

translateThetaFromFiles(dataFiles, mergedCalls = NULL,
    markerStep = 1000, sep = "\t", quote = "")
```

Arguments

`calls`	Numeric matrix with calls {0, 1/2, 1} representing allele ratios for each sample. Each row is a unique marker or paralogue (specified with `type`)
`resInfo`	Data table containing `featureData`, including the columns "Classification", "SNP", and "ILMN.Strand". These hold the genotype categories from `callGenotypes` and the SNP and TOP/BOT-category of the BeadArray markers (see `createAlleleSet`)
`type`	One of "regular", "single", or "merged" (see details below)
`BSRed`	`"AlleleSetIllumina"` object containing an `assayData` entry "call" and a `featureData` column "Classification" (see `callGenotypes`)
`mergedCalls`	Matrix with calls from resolved MSV-5 paralogs (see `assignParalogues`)
`dataFiles`	Character vector containing file names (see `makeFilenames`)
`markerStep`	The maximum number of markers loaded into the workspace at the time
`sep`	Field delimiter in text-files (see `read.table`)
`quote`	Quote-marks used for character strings (see `read.table`)

Details

The main difference between `translateTheta` and `translateThetaCombined` is that the former can only handle call-values {0, 1/2, 2} whereas the latter handles values {0, 1/4, 1/2, 3/4, 1}. In effect this means that markers from duplicated genome regions have to be handled in a special way if analysed with `translateTheta`. If `type == "regular"`, the markers are treated as if they were all from a diploid region. This implies that all non-segregating paralogs of "MSV-a" and "MSV-b" markers are ignored, effectively turning these markers into SNPs. Markers classified as "MSV-5" or "PSV" are set to missing (see `makeDiploidCalls`). If `type == "single"`, `calls` is expected to contain resolved "MSV-5" paralogs named with "-Para1" or "-Para2" (see `unmixParalogues`). If `type == "merged"`, resolved "MSV-5" paralogs named according to their respective chromosomes, "-ChromX", are expected (see `assignParalogues`). The main use of this function is to prepare genotype calls for mapping software which requires diploid markers.

With `translateThetaCombined`, there is always one element per marker, as required by the `"AlleleSetIllumina"`. If `mergedCalls` is given, the "MSV-5" paralogs will be resolved, otherwise only the ratio of the alleles across paralogs will be returned. The function `translateThetaFromFiles` performs the same operations on data sequentially loaded into the workspace, and the genotypes are written to file `dataFiles$genoFile` as they are found.

Value

Output from `translateTheta` is a matrix whose dimensions depend on the input data. If `calls` has one row per marker (i.e. `type == "regular"`), the number of rows in the output matrix also equals the number of markers. If `calls` has one row per paralogue (i.e. `type != "regular"`), the number of rows in the output matrix also equals the number of paralogs. Each element is a character string "x y" denoting the two alleles ("A", "T", "C", "G", or "-" for missing).

In contrast, the output from `translateThetaCombined` is an `AlleleSetIllumina` object with an added `assayData` entry "genotype". The elements of this matrix representing diploid markers are given as "xy", un-resolved tetraploid markers are given as "xyzw", and resolved tetraploid markers are given as "xy,zw" (paralogs separated by comma). The letters correspond to any of the 4 bases or "-" for missing.

The function `translateThetaFromFiles` are used for its side effects.

Author(s)

Lars Gidskehaug

See Also

`makeDiploidCalls`, `unmixParalogues`, `assignParalogues`, `makeFilenames`, `callGenotypes`

Examples

```
## Not run:
#Read 25 markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
```

```
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],markers=1:25,beadInfo=beadInfo)

#Genotype calling
BSRed <- callGenotypes(BSRed)
genotypes <- translateTheta(assayData(BSRed)$call,fData(BSRed),type='regular')
print(cbind(fData(BSRed)$Classification,genotypes[,1:3])[1:10,])

#Alternative output
BSRed <- translateThetaCombined(BSRed)
print(cbind(fData(BSRed)$Classification,assayData(BSRed)$genotype[,1:3])[1:10,])

## End(Not run)
```

`unmixParalogues` *Partially resolve the paralogs of MSV-5s*

Description

Based on parental and offspring MSV-5 marker genotypes, the paternal and maternal alleles in offspring are partally deduced. Those calls that can be split into two individual paralogs are returned. As the chromosome numbers of the individual paralogs are not known, the individual genotype calls are assigned assuming known location of the paralogs of each parent separately.

Usage

```
unmixParalogues(BSRed, singleCalls = getSingleCalls(BSRed))
```

Arguments

`BSRed`	"`AlleleSetIllumina`" (or "`MultiSet`") object containing only MSV-5 markers, with an `assayData` entry "call" (see `callGenotypes`) and a `phenoData` column "PedigreeID". The latter contains strings <p><mmm><fff><oo>, where "p", "mmm", "fff", and "oo" are unique identifiers for population, mother, father, and individual within full-sib group, respectively. "000" means founding parent, whereas "999" means unknown parent
`singleCalls`	Matrix holding the calls for all markers whose both paralogs are "AA", "AB", or "BB" (see `getSingleCalls`)

Details

This is the first step towards resolving MSV-5 paralogs, in the sense that the *mixed* signal from a duplicated marker is split into two paralogue-specific signals. Within the half-sib family of one parent at the time, the individual paralogs are arbitrarily named. Then, for each informative meiosis, the offspring genotype call for each paralogue is set to be either zero or one, depending on whether the allele inherited from the parent in question is "A" or "B", respectively. This is repeated for all parents, yielding two sparse data-tables, one for the male and one for the female parent half-sib families. The alternate parents' calls are set to missing. The data-tables may separately be used

for linkage mapping using other software, however better results are expected if the tables can be merged first. This involves positioning of the paralogs on their respective chromosomes and requires a linkage map (see assignParalogues).

Value

A list with elements

mother	Matrix of calls representing maternal inherited alleles
father	Matrix of calls representing paternal inherited alleles

The number of rows in both matrices is twice the number of markers, and the names of the paralogs are appended with "_Para1" or "_Para2". Critically, these names are NOT consistent between the tables

Note

Offspring alleles from monomorpic parents are sometimes missing even though the inherited alleles can be directly deduced from the parental genotype. This is for efficiency of the algorithm, as monomorphic parents carry no linkage information

Author(s)

Lars Gidskehaug

See Also

AlleleSetIllumina, callGenotypes, assignParalogues, getSingleCalls, translateTheta

Examples

```
## Not run:
#Read markers into an AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling and selection of some MSV-5s
BSRed <- callGenotypes(BSRed)
BSRed <- validateCallsPedigree(BSRed)
iMSV5 <- fData(BSRed)$Classification %in% 'MSV-5' & fData(BSRed)$Ped.Errors %in% 0
plotGenotypes(BSRed,markers=which(iMSV5))

#Partial resolving of paralogs
paraCalls <- unmixParalogues(BSRed[iMSV5,])

#Compare for an arbitrary, single triplet
iOffspring <- 1
```

```
iFather <- which(pData(BSRed)$PedigreeID %in% pData(BSRed)$Parent2[iOffspring])
iMother <- which(pData(BSRed)$PedigreeID %in% pData(BSRed)$Parent1[iOffspring])
print(assayData(BSRed)$call[iMSV5,c(iMother,iFather,iOffspring)])
print(paraCalls$mother[,c(iMother,iFather,iOffspring)])
print(paraCalls$father[,c(iMother,iFather,iOffspring)])

## End(Not run)
```

---

validateCallsPedigree *Pedigree validation of genotypes*

Description

Checks that genotypes follow basic rules of inheritance

Usage

```
validateCallsPedigree(BSRed)

validateSingleCall(marker, classification)
```

Arguments

BSRed	"AlleleSetIllumina" object containing an assayData entry "call" as well as a featureData column "Classification" (see callGenotypes), or alternatively a column "Manual.Calls.R" (see callGenotypes.interactive). It must also contain a phenoData column "PedigreeID". This contains strings <p><mmm><fff><oo>, where "p", "mmm", "fff", and "oo" are unique identifiers for population, mother, father, and individual within full-sib group, respectively. "000" means founding parent, whereas "999" means unknown parent
marker	Data frame with at least the columns "Call" and "PedigreeID" for a single marker. The first corresponds to the "AlleleSetIllumina" assayData entry "call", the latter is described above.
classification	Character string with genotype call category for the marker (see generatePolyCenters)

Details

validateCallsPedigree is usually faster and easier to use compared to validateSingleCall. The latter is more often called by other functions, however it has a slightly more detailed outut.

Value

Output from validateCallsPedigree is an "AlleleSetIllumina" object with an added assayData entry "ped.check". This logical matrix is FALSE for all offspring violating the pedigree information, an all parents are NA

Output from validateSingleCall is a numeric vector with {0, 1, 2} denoting no error, offspring error, and parent error, respectively

Author(s)

Lars Gidskehaug

See Also

callGenotypes, plotGenotypes

Examples

```
## Not run:
#Read pre-processed data directly into AlleleSetIllumina object
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- setNormOptions()
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Genotype calling, pedigree validation and plotting
BSRed <- callGenotypes(BSRed)
BSRed <- validateCallsPedigree(BSRed)
plotGenotypes(BSRed,1:25)

## End(Not run)
```

---

writeAlleleSet *Write AlleleSetIllumina data to files*

Description

Writes assayData, phenoData, and/or featureData of "AlleleSetIllumina" objects to text-files

Usage

```
writeAlleleSet(dataFiles, BSRed, append = TRUE, sep = "\t",
    quote = FALSE)
```

Arguments

dataFiles	List with file-names (see makeFilenames). Only elements of BSRed indicated here are saved
BSRed	"AlleleSetIllumina" object
append	If TRUE, append data rather than overwrite files (see write.table)
sep	Field delimiter string (see write.table)
quote	If TRUE, character and factor columns are quoted (see write.table)

Details

The data-tables specified is written to text-files for subsequent loading using createAlleleSetFromFiles (or similar). The pre-processed *data*-files "intFile", "thFile", "seFile", and "phFile" are used to estimate the *result*-files "callFile", "resFile", and "genoFile". Subsequent overwriting of data-files may therefore invalidate result-file. Therefore, no data-files will be saved if result-files are saved in the same run.

Value

This function is used for its side effects

Warning

Use with caution, as files will be overwritten without question if so specified

Note

The assayData-entries "intensity", "theta", and "SE" are written array-wise to file. This enables pre-processing of a subset of arrays at the time. The entries "call" and "genotype" are written marker-wise, which enables genotype-calling of a subset of markers at the time

Author(s)

Lars Gidskehaug

See Also

createAlleleSetFromFiles, makeFilenames, AlleleSetIllumina

Examples

```
## Not run:
#Create an AlleleSetIllumina object using files in example data directory
rPath <- system.file("extdata", package="beadarrayMSV")
normOpts <- list(transf='root',method='medianAF')
dataFiles <- makeFilenames('testdata',normOpts,rPath)
beadFile <- paste(rPath,'beadData_testdata.txt',sep='/')
beadInfo <- read.table(beadFile,sep='\t',header=TRUE,as.is=TRUE)
BSRed <- createAlleleSetFromFiles(dataFiles[1:4],beadInfo=beadInfo)

#Write some data to files
writeFiles <- makeFilenames('test2',normOpts,path='.')
writeAlleleSet(writeFiles[1:4],BSRed[1:10,1:5])

## End(Not run)
```

Index

∗Topic **classes**
AlleleSetIllumina-class, 3
BeadSetIllumina-class, 8
∗Topic **datasets**
BSRed.193, 10
∗Topic **package**
beadarrayMSV-package, 2

AlleleSetIllumina, *3*, *6, 7*, *10*, *13*, *15*, *18–23*, *27*, *32–37*, *39*, *42*, *49–51*, *57–63*
AlleleSetIllumina-class, 3
assignParalogues, *3*, 5, *32, 33*, *35*, *49, 50*, *57, 58*, *60*
assignToAlleleSet, *3*, *15, 16*
assignToAlleleSet *(*createAlleleSet*)*, 19
axis, *34*

beadarrayMSV *(*beadarrayMSV-package*)*, 2
beadarrayMSV-package, 2
BeadSetIllumina, *3–5*, *19, 20*, *29*, *44–48*, *50–53*
BeadSetIllumina-class, 8
BSRed.193, 10

callGenotypes, *3–5*, 12, *18–20*, *24*, *27, 28*, *32–34*, *36*, *43*, *49, 50*, *55*, *57–62*
callGenotypes.interactive, *3*, *36*, *38, 39*, *61*
cart2pol, 17, *25, 26*, *45*
countFailedSNP, 18
createAlleleSet, *4, 5*, *10*, 19, *22*, *25, 26*, *46*, *48*, *57*
createAlleleSetFromFiles, *3, 4*, *38*, *63*
createAlleleSetFromFiles *(*createAlleleSet*)*, 19
createMultiSetFromFiles, *38*
createMultiSetFromFiles *(*createAlleleSet*)*, 19

eSet, *3–5*, *8–10*

findClusters, 22, *28*
findPolyploidClusters, *16*, 23
findSeTheta, 25

generatePolyCenters, *13, 14*, *16*, 26, *28*, *39*, *54*, *61*
getCenters, *14*, *16*, *22*, *24*, *27*, 27
getNoiseDistributions, 29, *32*, *40*, *45, 46*
getNormInd, *29*, 31, *45, 46*
getSingleCalls, *6, 7*, 32, *59, 60*
getSpecificCenters, *22*, *28*
getSpecificCenters *(*getCenters*)*, 27
ggobi, *16*, *39*

hist, *22, 23*, *27*, *45*

initialize,AlleleSetIllumina-method *(*AlleleSetIllumina-class*)*, 3
initialize,BeadSetIllumina-method *(*BeadSetIllumina-class*)*, 8

k-means, *14*
kmeans, *23, 24*

locateParalogues, *6, 7*, 33

makeDiploidCalls, 36, *58*
makeFilenames, *20, 21*, 37, *57, 58*, *62, 63*
manualCall, *16*, 38
modifyExistingFilenames *(*makeFilenames*)*, 37
MultiSet, *5–7*, *19–21*, *32*, *34, 35*, *49*, *59*

normalizeIllumina *(*shearRawSignal*)*, 51
normalizeQuantiles, *40, 41*
normalizeShearedChannels, 40, *45, 46*

par, *42*
plotCountsChrom, *7*, *35*
plotCountsChrom *(*locateParalogues*)*, 33
plotEstimatedNoise, *45*

plotEstimatedNoise
(getNoiseDistributions), 29
plotGenotypes, 3, 14, 16, 27, 42, 51, 62
plotPreprocessing, 30, 51
plotPreprocessing (preprocessBeadSet), 44
preprocessBeadSet, 3, 5, 10, 13, 18, 30, 32, 41, 44, 48, 52, 53, 56

quantile, 44

read.table, 20, 57
readBeadSummaryOutput, 3, 10, 46, 47
resolveInheritanceSNP, 6, 7, 34, 35, 49

scatterArrays, 48, 50
setGenoOptions, 24, 27
setGenoOptions (callGenotypes), 12
setMergeOptions, 6, 7, 34, 35
setMergeOptions (assignParalogues), 5
setNormOptions, 18, 20, 29, 37, 40, 52, 53, 55
setNormOptions (preprocessBeadSet), 44
shearRawSignal, 45, 46, 51
smoothScatter, 51

testHardyWeinberg, 16, 27, 54
transformChannels, 25, 45, 46, 55
transformSEs, 45, 46
transformSEs (transformChannels), 55
translateTheta, 36, 57, 60
translateThetaCombined
(translateTheta), 57
translateThetaFromFiles, 38
translateThetaFromFiles
(translateTheta), 57

unmixParalogues, 6, 7, 32–35, 50, 58, 59

validateCallsPedigree, 3, 4, 15, 16, 18, 19, 43, 61
validateSingleCall, 15, 39, 43
validateSingleCall
(validateCallsPedigree), 61
Versioned, 4, 9
VersionedBiobase, 4, 9

write.table, 62
writeAlleleSet, 3, 21, 38, 62