Package 'RAIDS'

May 9, 2024

Type Package

Title Accurate Inference of Genetic Ancestry from Cancer Sequences

Description This package implements specialized algorithms that enable genetic ancestry inference from various cancer sequences sources (RNA, Exome and Whole-Genome sequences). This package also implements a simulation algorithm that generates synthetic cancer-derived data. This code and analysis pipeline was designed and developed for the following publication: Belleau, P et al. Genetic Ancestry Inference from Cancer-Derived Molecular Data across Genomic and Transcriptomic Platforms. Cancer Res 1 January 2023; 83 (1): 49–58.

Version 1.3.0

License Apache License (>= 2)

Encoding UTF-8

NeedsCompilation no

VignetteBuilder knitr

Depends R (>= 4.2.0), gdsfmt, SNPRelate, stats, utils, GENESIS

Imports S4Vectors, GenomicRanges, ensembldb, BSgenome, AnnotationDbi, methods, class, pROC, IRanges, AnnotationFilter, rlang, VariantAnnotation, MatrixGenerics,

Suggests testthat, knitr, rmarkdown, BiocStyle, withr, GenomeInfoDb, BSgenome.Hsapiens.UCSC.hg38, EnsDb.Hsapiens.v86

BugReports https://github.com/KrasnitzLab/RAIDS/issues

URL https://krasnitzlab.github.io/RAIDS/

biocViews Genetics, Software, Sequencing, WholeGenome, PrincipalComponent, GeneticVariability, DimensionReduction

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

git_url https://git.bioconductor.org/packages/RAIDS

git_branch devel

git_last_commit 8c31b43

git_last_commit_date 2024-04-30

Repository Bioconductor 3.20

Date/Publication 2024-05-08

Author Pascal Belleau [cre, aut] (<https://orcid.org/0000-0002-0802-1071>),
Astrid Deschênes [aut] (<https://orcid.org/0000-0001-7846-6749>),
David A. Tuveson [aut] (<https://orcid.org/0000-0002-8017-2712>),
Alexander Krasnitz [aut]

Maintainer Pascal Belleau <pascal_belleau@hotmail.com>

Contents

`RAIDS-package` *RAIDS: Accurate Inference of Genetic Ancestry from Cancer Sequences*

Description

The RAIDS package implements specialized algorithms that enable ancestry inference from various cancer data sources (RNA, Exome and Whole-Genome sequencing).

Details

The RAIDS package also implements simulation algorithm that generates synthetic cancer-derived data.

This code and analysis pipeline was designed and developed for the following publication:

Pascal Belleau, Astrid Deschênes, Nyasha Chambwe, David A. Tuveson, Alexander Krasnitz; Genetic Ancestry Inference from Cancer-Derived Molecular Data across Genomic and Transcriptomic Platforms. Cancer Res 1 January 2023; 83 (1): 49–58. https://doi.org/10.1158/0008-5472.CAN-22-0682

Value

RAIDS

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Maintainer: Pascal Belleau pascal_belleau@hotmail.com

References

Pascal Belleau, Astrid Deschênes, Nyasha Chambwe, David A. Tuveson, Alexander Krasnitz; Genetic Ancestry Inference from Cancer-Derived Molecular Data across Genomic and Transcriptomic Platforms. Cancer Res 1 January 2023; 83 (1): 49–58. https://doi.org/10.1158/0008-5472.CAN-22-0682

See Also

- `runExomeAncestry` This function runs most steps leading to the ancestry inference call on a specific exome profile.
- `runExomeAncestry` This function runs most steps leading to the ancestry inference call on a specific RNA profile.
- `estimateAllelicFraction` This function estimates the allelic fraction of the pruned SNVs for a specific sample and add the information to the associated GDS Sample file. The allelic fraction estimation method is adapted to the type of study (DNA or RNA).
- `computeSyntheticROC` This function calculate the AUROC of the inferences for specific values of D and K using the inferred ancestry results from the synthetic profiles.
- `generateMapSnvSel` The function applies a cut-off filter to the SNV information file to retain only the SNV that have a frequency superior or equal to the specified cut-off in at least one super population.

`add1KG2SampleGDS` *Add the genotype information for the list of pruned SNVs into the Profile GDS file*

Description

The function extracts the information about the pruned SNVs from the 1KG GDS file and adds entries related to the pruned SNVs in the Profile GDS file. The nodes are added to the Profile GDS file: 'sample.id', 'snp.id', 'snp.chromosome', 'snp.position', 'snp.index', 'genotype' and 'lap'.

Usage

```
add1KG2SampleGDS(gdsReference, fileProfileGDS, currentProfile, studyID)
```

Arguments

`gdsReference`	an object of class gds.class (a GDS file), the opened 1KG GDS file.
`fileProfileGDS`	a `character` string representing the path and file name of the Profile GDS file. The Profile GDS file must exist.
`currentProfile`	a `character` string corresponding to the sample identifier associated to the current list of pruned SNVs.
`studyID`	a `character` string corresponding to the study identifier associated to the current list of pruned SNVs.

Value

The function returns `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## The data.frame containing the information about the study
## The 3 mandatory columns: "studyID", "study.desc", "study.platform"
## The entries should be strings, not factors (stringsAsFactors=FALSE)
studyDF <- data.frame(study.id="MYDATA",
                      study.desc="Description",
                      study.platform="PLATFORM",
                      stringsAsFactors=FALSE)

## Temporary Profile file
fileProfile <- file.path(tempdir(), "ex2.gds")

## Copy required file
file.copy(file.path(dataDir, "ex1_demo_with_pruning.gds"),
        fileProfile)

## Open 1KG file
gds1KG <- snpgdsOpen(fileGDS)

## Compute the list of pruned SNVs for a specific profile 'ex1'
## and save it in the Profile GDS file 'ex2.gds'
add1KG2SampleGDS(gdsReference=gds1KG,
        fileProfileGDS=fileProfile,
        currentProfile=c("ex1"),
        studyID=studyDF$study.id)

## Close the 1KG GDS file (important)
closefn.gds(gds1KG)

## Check content of Profile GDS file
## The 'pruned.study' entry should be present
content <- openfn.gds(fileProfile)
content

## Close the Profile GDS file (important)
closefn.gds(content)
```

```
## Remove Profile GDS file (created for demo purpose)
unlink(fileProfile, force=TRUE)
```

---

addBlockInGDSAnnot *Add block information in a Population Reference GDS Annotation file*

Description

This function appends the information for one specific type of blocks into a Population Reference GDS Annotation file. More specifically, the node 'block.annot' is created if it does not exists. This node contains a `data.frame` which will be append the description of the current block. The node 'block' is also created if it does not exists. This node is a `matrix` that will contain all the entries for the current block. All the values for a specific block type are contained in a single column that corresponds to the row number in the 'block.annot' node.

Usage

```
addBlockInGDSAnnot(gds, listBlock, blockName, blockDesc)
```

Arguments

`gds`	an object of class `gds` opened in writing mode.
`listBlock`	a `array` of `integer` representing all the entries for the current block.
`blockName`	a `character` string representing the unique block name.
`blockDesc`	a `character` string representing the description of the current block.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Temporary GDS Annotation file in current directory
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_Annot_14.gds")

## Create and open the GDS file
GDS_file_tmp  <- createfn.gds(filename=gdsFilePath)
```

```
## One block
blockType <- "EAS.0.05.500k"

## The description of the block
blockDescription <- "EAS population blocks based on 500k windows"

## The values for each entry related to the block (integers)
blockEntries <- c(1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3)

RAIDS:::addBlockInGDSAnnot(gds=GDS_file_tmp, listBlock=blockEntries,
    blockName=blockType, blockDesc=blockDescription)

## Read 'block.annot' node
read.gdsn(index.gdsn(GDS_file_tmp, "block.annot"))

## Read 'block' node
read.gdsn(index.gdsn(GDS_file_tmp, "block"))

## Close GDS file
closefn.gds(gdsfile=GDS_file_tmp)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

`addGDSRef` *Create a "sample.ref" node i a GDS file with the information about the related/unrelated state of the reference samples*

Description

This function creates a "sample.ref" node in the GDS file. The node contains a vector of integers with value of 1 when the samples are used as references and 0 otherwise. The information used to fill the "sample.ref" node comes from the RDS file that contains the information about the unrelated reference samples.

Usage

```
addGDSRef(gdsReference, filePart)
```

Arguments

`gdsReference`	an object of class gds.class (a GDS file), the opened GDS file.
`filePart`	a `character` string representing the path and file name of a RDS file containing the information about the related and unrelated samples in the reference dataset. The RDS file must exist. The RDS file must contains a `vector` of `character` strings called "unrels" with the name of the unrelated samples.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Locate RDS with unrelated/related status for Reference samples
dataDir <- system.file("extdata", package="RAIDS")
rdsFilePath <- file.path(dataDir, "unrelatedPatientsInfo_Demo.rds")

## Temporary GDS file
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_11.gds")

## Create and open the GDS file
tmpGDS  <- createfn.gds(filename=gdsFilePath)
## Create "sample.id" node (the node must be present)
sampleIDs <- c("HG00104", "HG00109", "HG00110")
add.gdsn(node=tmpGDS, name="sample.id", val=sampleIDs)

## Create "sample.ref" node in GDS file using RDS information
RAIDS:::addGDSRef(gdsReference=tmpGDS, filePart=rdsFilePath)

## Read sample reference data.frame
read.gdsn(index.gdsn(node=tmpGDS, path="sample.ref"))

## Close GDS file
closefn.gds(gdsfile=tmpGDS)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

---

addGDSStudyPruning *Add the pruned.study entry related to the SNV dataset in the Profile GDS file*

Description

This function adds the names of the SNVs into the node called "pruned.study" in GDS Sample file. If a "pruned.study" entry is already present, the entry is deleted and a new entry is created.

Usage

```
addGDSStudyPruning(gdsProfile, pruned)
```

Arguments

gdsProfile	an object of class gds.class (a GDS file), the opened Profile GDS file.
pruned	a vector of character string representing the name of the SNVs.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Create a temporary GDS file in an test directory
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_1.gds")

## Create and open the GDS file
tmpGDS  <- createfn.gds(filename=gdsFilePath)

## Vector of low allelic fraction
study <- c("s19222", 's19588', 's19988', 's20588', 's23598')

## Add segments to the GDS file
RAIDS:::addGDSStudyPruning(gdsProfile=tmpGDS, pruned=study)

## Read lap information from GDS file
read.gdsn(index.gdsn(node=tmpGDS, path="pruned.study"))

## Close GDS file
closefn.gds(gdsfile=tmpGDS)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

addGeneBlockGDSRefAnnot *Append information associated to blocks, as indexes, into the Population Reference SNV Annotation GDS file*

Description

The function appends the information about the blocks into the Population Reference SNV Annotation GDS file. The information is extracted from the Population Reference GDS file.

Usage

```
addGeneBlockGDSRefAnnot(
  gdsReference,
  gdsRefAnnotFile,
  winSize = 10000,
  ensDb,
  suffixBlockName
)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened Reference GDS file.
gdsRefAnnotFile	a character string representing the file name corresponding the Reference SNV Annotation GDS file. The function will open it in write mode and close it after. The file must exist.
winSize	a single positive integer representing the size of the window to use to group the SNVs when the SNVs are in a non-coding region. Default: 10000L.
ensDb	An object with the ensembl genome annotation Default: EnsDb.Hsapiens.v86.
suffixBlockName	a character string that identify the source of the block and that will be added to the block description into the Reference SNV Annotation GDS file, as example: Ensembl.Hsapiens.v86.

Value

The integer 0L when the function is successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(SNPRelate)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

fileAnnotGDS <- file.path(tempdir(), "ex1_good_small_1KG_Ann_GDS.gds")

## Required library
if (requireNamespace("EnsDb.Hsapiens.v86", quietly=TRUE)) {

    file.copy(file.path(dataDir, "tests",
        "ex1_NoBlockGene.1KG_Annot_GDS.gds"), fileAnnotGDS)

    ## Making a "short cut" on the ensDb object
    edb <- EnsDb.Hsapiens.v86::EnsDb.Hsapiens.v86

    ## GDS Reference file
    fileReferenceGDS  <- file.path(dataDir, "tests",
                "ex1_good_small_1KG.gds")


        ## Open the reference GDS file (demo version)
        gds1KG <- snpgdsOpen(fileReferenceGDS)

        ## Append information associated to blocks
        addGeneBlockGDSRefAnnot(gdsReference=gds1KG,
            gdsRefAnnotFile=fileAnnotGDS,
            ensDb=edb,
            suffixBlockName="EnsDb.Hsapiens.v86")

        gdsAnnot1KG <- openfn.gds(fileAnnotGDS)
        print(gdsAnnot1KG)
        print(read.gdsn(index.gdsn(gdsAnnot1KG, "block.annot")))

       ## Close GDS files
       closefn.gds(gds1KG)
       closefn.gds(gdsAnnot1KG)


    ## Remove temporary file
    unlink(fileAnnotGDS, force=TRUE)

}
```

---

addRef2GDS1KG *Add the information about the unrelated patients to the Reference GDS file*

Description

This function adds the information about the unrelated patients to the Reference GDS file. More specifically, it creates the field `sample.ref` which as the value `1` when the sample is unrelated and the value `0` otherwise. The `sample.ref` is filled based on the information present in the input RDS file.

Usage

```
addRef2GDS1KG(fileNameGDS, filePart)
```

Arguments

`fileNameGDS`	a `character` string representing the path and file name of the GDS file that contains the Reference information. The Reference GDS file must contain the SNP information, the genotyping information and the pedigree information from Reference dataset. The extension of the file must be '.gds'.
`filePart`	a `character` string representing the path and file name of the RDS file that contains the information about the Reference patients that are unrelated. The extension of the file must be '.rds'. The file must exists.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Locate RDS with unrelated/related status for 1KG samples
dataDir <- system.file("extdata", package="RAIDS")
rdsFilePath <- file.path(dataDir, "unrelatedPatientsInfo_Demo.rds")

## Create a temporary GDS file in an test directory
dataDir <- system.file("extdata/tests", package="RAIDS")
gdsFilePath <- file.path(dataDir, "GDS_TEMP_201.gds")

## Create and open the GDS file
tmpGDS  <- createfn.gds(filename=gdsFilePath)
## Create "sample.id" node (the node must be present)
sampleIDs <- c("HG00104", "HG00109", "HG00110")
add.gdsn(node=tmpGDS, name="sample.id", val=sampleIDs)

## Create "snp.id" node (the node must be present)
snpIDs <- c("s1", "s2", "s3", "s4", "s5", "s6")
add.gdsn(node=tmpGDS, name="snp.id", val=snpIDs)

## Create "snp.position" node (the node must be present)
snpPositions <- c(16102, 51478, 51897, 51927, 54489, 54707)
```

```
add.gdsn(node=tmpGDS, name="snp.position", val=snpPositions)

## Create "snp.chromosome" node (the node must be present)
snpPositions <- c(1, 1, 1, 1, 1, 1)
add.gdsn(node=tmpGDS, name="snp.chromosome", val=snpPositions)

## Create "genotype" node (the node must be present)
genotype <- matrix(rep(1, 18), ncol = 3)
add.gdsn(node=tmpGDS, name="genotype", val=genotype)

## Close GDS file
closefn.gds(tmpGDS)

## Create  "sample.ref" node in GDS file using RDS information
addRef2GDS1KG(fileNameGDS=gdsFilePath, filePart=rdsFilePath)

## Read sample reference data.frame
fileGDS <- openfn.gds(gdsFilePath, readonly=TRUE)
read.gdsn(index.gdsn(node=fileGDS, path="sample.ref"))
closefn.gds(gdsfile=fileGDS)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

---

addStudy1Kg *Append information about the 1KG samples into the Profile GDS file*

Description

The information about the samples present in the 1KG GDS file is added into the GDS Sample file. Only the information about the unrelated samples from the 1OOO Genome Study are copied into the GDS Sample file. The information is only added to the GDS Sample file when the 1KG Study is not already present in the GDS Sample file. The sample information for all selected samples is appended to the GDS Sample file "study.annot" node. The study information is appended to the GDS Sample file "study.list" node.

Usage

```
addStudy1Kg(gdsReference, fileProfileGDS, verbose = FALSE)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened 1KG GDS file.
fileProfileGDS	a character string representing the path and file name of the GDS Sample file. The GDS Sample file must exist.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(gdsfmt)

## Get the temp folder
tempDir <- tempdir()

## Create a temporary 1KG GDS file and add needed information
fileName1KG <- file.path(tempDir, "GDS_TEMP_addStudy1Kg_1KG.gds")
gds1KG  <- createfn.gds(filename=fileName1KG)
add.gdsn(gds1KG, "sample.id", c("HTT101", "HTT102", "HTT103"))

samples <- data.frame(sex=c(1, 1, 2), pop.group=c("GBR", "GIH", "GBR"),
    superPop=c("EUR", "SAS", "EUR"), batch=rep(0, 3),
    stringsAsFactors = FALSE)

add.gdsn(gds1KG, "sample.annot", samples)
add.gdsn(gds1KG, "sample.ref", c(1,0, 1))
sync.gds(gds1KG)

## Create a temporary Profile GDS file
fileNameProfile <- file.path(tempDir, "GDS_TEMP_addStudy1Kg_Sample.gds")
gdsProfile <- createfn.gds(fileNameProfile)

study.list <- data.frame(study.id=c("HTT Study"),
    study.desc=c("Important Study"),
    study.platform=c("Panel"), stringsAsFactors=FALSE)

add.gdsn(gdsProfile, "study.list", study.list)

study.annot <- data.frame(data.id=c("TOTO1"), case.id=c("TOTO1"),
                sample.type=c("Study"), diagnosis=c("Study"),
                source=rep("IGSR"), study.id=c("Study"),
                stringsAsFactors=FALSE)

add.gdsn(gdsProfile, "study.annot", study.annot)
sync.gds(gdsProfile)
closefn.gds(gdsProfile)

## Append information about the 1KG samples into the Profile GDS file
## The Profile GDS file will contain 'study.list' and 'study.annot' entries
addStudy1Kg(gdsReference=gds1KG, fileProfileGDS=fileNameProfile,
    verbose=TRUE)
```

```
closefn.gds(gds1KG)
unlink(fileNameProfile, recursive=TRUE, force=TRUE)
unlink(fileName1KG, recursive=TRUE, force=TRUE)
unlink(tempDir)
```

`addStudyGDSSample` *Add information related to a specific study and specific samples into a GDS Sample file*

Description

This function add entries related to 1) a specific study and 2) specific samples into a GDS Sample file. The study information is appended to the GDS Sample file "study.list" node when the node is already present in the file. Otherwise, the node is created and then, the information is added. The sample information for all selected samples is appended to the GDS Sample file "study.annot" node when the node is already present in the file. Otherwise, the node is created and then, the information is added.

Usage

```
addStudyGDSSample(gdsProfile, pedProfile, batch, listSamples, studyDF, verbose)
```

Arguments

`gdsProfile`	an object of class gds.class (a GDS file), the opened GDS file.
`pedProfile`	a `data.frame` with the sample information. The `data.frame` must have the columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis" and "Source". The unique sample identifier of the `data.frame` is the "Name.ID" column and the row names of the `data.frame` must be the "Name.ID" values.
`batch`	a `integer` corresponding the batch associated to the study.
`listSamples`	a `vector` of `character` string representing the samples (samples identifiers) that are saved into the GDS. All the samples must be present in the 'pdeDF' `data.frame`. If `NULL`, all samples present in the `dfPedProfile` are used.
`studyDF`	a `data.frame` with at least the 3 columns: "study.id", "study.desc" and "study.platform". The three columns are in character string format (no factor).
`verbose`	a `logical` indicating if messages should be printed to show how the different steps in the function.

Value

a `vector` of `character` strings representing the sample identifiers that have been saved in the GDS Sample file.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Create a temporary GDS file in an current directory
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_11.gds")

## Create and open the GDS file
tmpGDS  <- createfn.gds(filename=gdsFilePath)

## Create a PED data frame with sample information
ped1KG <- data.frame(Name.ID=c("1KG_sample_01", "1KG_sample_02"),
        Case.ID=c("1KG_sample_01", "1KG_sample_02"),
        Sample.Type=rep("Reference", 2), Diagnosis=rep("Reference", 2),
        Source=rep("IGSR", 2), stringsAsFactors=FALSE)

## Create a Study data frame with information about the study
## All samples are associated to the same study
studyInfo <- data.frame(study.id="Ref.1KG",
        study.desc="Unrelated samples from 1000 Genomes",
        study.platform="GRCh38 1000 genotypes",
        stringsAsFactors=FALSE)

## Add the sample information to the GDS Sample file
## The information for all samples is added (listSamples=NULL)
RAIDS:::addStudyGDSSample(gdsProfile=tmpGDS, pedProfile=ped1KG, batch=1,
        listSamples=NULL, studyDF=studyInfo, verbose=FALSE)

## Read study information from GDS Sample file
read.gdsn(index.gdsn(node=tmpGDS, path="study.list"))

## Read sample information from GDS Sample file
read.gdsn(index.gdsn(node=tmpGDS, path="study.annot"))

## Close GDS file
closefn.gds(gdsfile=tmpGDS)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

`addUpdateLap` *Add information related to low allelic fraction associated to the SNV dataset for a specific sample into a GDS file*

Description

The function adds the information related to low allelic fraction associated to the SNV dataset for a specific sample into a GDS file, more specifically, in the "lap" node. The "lap" node must already be present in the GDS file.

Usage

```
addUpdateLap(gdsProfile, snpLap)
```

Arguments

gdsProfile	an object of class gds.class (a GDS file), a GDS file.
snpLap	a vector of numeric value representing the low allelic fraction for each SNV present in the SNV dataset. The values should be between 0 and 0.50. The length of the vector should correspond to the number of SNVs present in the "snp.id" entry of the GDS sample file.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Create a temporary GDS file
gdsFilePath <- file.path(tempdir(), "GDS_TEMP.gds")

## Create and open the GDS file
gdsFile  <- createfn.gds(filename=gdsFilePath)

## Create a "lap" node
add.gdsn(node=gdsFile, name="lap", val=rep(10L, 12))
sync.gds(gdsFile)

## Vector of low allelic fraction
lap <- c(0.1, 0.23, 0.34, 0.00, 0.12, 0.11, 0.33, 0.55)

## Add segments to the GDS file
RAIDS:::addUpdateLap(gdsProfile=gdsFile, snpLap=lap)

## Read lap information from GDS file
read.gdsn(index.gdsn(node=gdsFile, path="lap"))

## Close GDS file
closefn.gds(gdsfile=gdsFile)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

addUpdateSegment *Add information related to segments associated to the SNV dataset for a specific sample into a GDS file*

Description

The function adds the information related to segments associated to the SNV dataset for a specific sample into a GDS file, more specifically, in the "segment" node. If the "segment" node already exists, the previous information is erased.

Usage

```
addUpdateSegment(gdsProfile, snpSeg)
```

Arguments

gdsProfile	an object of class gds.class (a GDS file), a GDS Sample file.
snpSeg	a vector of integer representing the segment identifiers associated to each SNV selected for the specific sample. The length of the vector should correspond to the number of SNVs present in the "snp.id" entry of the GDS sample file.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Temporary GDS file
gdsFilePath <- file.path(tempdir(), "GDS_TEMP.gds")

## Create and open the GDS file
GDS_file_tmp  <- createfn.gds(filename=gdsFilePath)

## Vector of segment identifiers
segments <- c(1L, 1L, 1L, 2L, 2L, 3L, 3L)

## Add segments to the GDS file
RAIDS:::addUpdateSegment(gdsProfile=GDS_file_tmp, snpSeg=segments)

## Read segments information from GDS file
read.gdsn(index.gdsn(node=GDS_file_tmp, path="segment"))
```

```
## Close GDS file
closefn.gds(gdsfile=GDS_file_tmp)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

---

appendGDSgenotype *Append information related to profile genotypes into a Population Reference GDS file (associated node already present in the GDS)*

Description

This function appends the genotype fields with the associated information into the Population Reference GDS file for the selected profiles. The associated node must already present in the GDS file.

Usage

```
appendGDSgenotype(gds, listSample, pathGeno, fileSNPsRDS, verbose)
```

Arguments

gds	an object of class gds.class (a GDS file), the opened Population Reference GDS file.
pathGeno	a character string representing the path where the reference genotyping files for each sample are located. The name of the genotyping files must correspond to the individual identification (Individual.ID) in the pedigree file.
fileSNPsRDS	a character string representing the path and file name of the RDS file that contains the indexes of the retained SNPs. The file must exist. The file must be a RDS file.
verbose	a logical indicating if the function must print messages when running.
listSamples	a character string representing the path and file name of the RDS file that contains the indexes of the retained SNPs. The file must exist. The file must be a RDS file.

Value

The integer 0 when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Path where the demo genotype CSV files are located
pathGeno <- file.path(dataDir, "demoProfileGenotypes")

## The RDS file containing the pedigree information
pedigreeFile <- file.path(dataDir, "PedigreeReferenceDemo.rds")

## The RDS file containing the indexes of the retained SNPs
snpIndexFile <- file.path(dataDir, "listSNPIndexes_Demo.rds")

## The RDS file containing the filtered SNP information
filterSNVFile <- file.path(dataDir, "mapSNVSelected_Demo.rds")

## Temporary Reference GDS file
tempRefGDS <- file.path(tempdir(), "Ref_TEMP02.gds")

## Create temporary Reference GDS file
newGDS <- createfn.gds(tempRefGDS)
put.attr.gdsn(newGDS$root, "FileFormat", "SNP_ARRAY")

## Read the pedigree file
ped1KG <- readRDS(pedigreeFile)

## Add information about samples to the Reference GDS file
listSampleGDS <- RAIDS:::generateGDSRefSample(gdsReference=newGDS,
                dfPedReference=ped1KG, listSamples=NULL)

## Add SNV information to the Reference GDS
RAIDS:::generateGDSSNPinfo(gdsReference=newGDS, fileFreq=filterSNVFile,
                verbose=FALSE)

## Add genotype information to the Reference GDS for the 3 first samples
RAIDS:::generateGDSgenotype(gds=newGDS, pathGeno=pathGeno,
        fileSNPsRDS=snpIndexFile, listSamples=listSampleGDS[1:3],
        verbose=FALSE)

## Append genotype information to the Reference GDS for the other samples
RAIDS:::appendGDSgenotype(gds=newGDS, pathGeno=pathGeno,
        fileSNPsRDS=snpIndexFile,
        listSample=listSampleGDS[4:length(listSampleGDS)],
        verbose=FALSE)

## Close file
closefn.gds(newGDS)

## Remove temporary files
```

```
unlink(tempRefGDS, force=TRUE)
```

`appendGDSgenotypeMat` *Appends the genotype information for specific samples (1 column == 1 profile) into a GDS file*

Description

This function appends the genotype information into a GDS file. More specifically, the genotype information is added to the "genotype" node. The "genotype" node must already be present in the GDS file. The genotype information is a matrix with the rows corresponding to SNVs and columns corresponding to samples. The number of rows of the new genotype information must correspond to the number of rows of the matrix present in the "genotype" node.

Usage

```
appendGDSgenotypeMat(gds, matG)
```

Arguments

`gds`	an object of class gds.class (a GDS file), the opened Profile GDS file.
`matG`	a `matrix` of `integer` representing the genotypes of the SNVs for one or multiple samples. The rows correspond to SNVs and the columns correspond to samples. The number of rows must correspond to the number of rows of the matrix present in the "genotype" node.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Create a temporary GDS file
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_06.gds")

## Create and open the GDS file
tmpGDS  <- createfn.gds(filename=gdsFilePath)

## Create a "genotype" node with initial matrix
genoInitial <- matrix(rep(0L, 10), nrow=2)
```

```
add.gdsn(node=tmpGDS, name="genotype", val=genoInitial)
sync.gds(tmpGDS)

## New genotype information to be added
newGenotype <- matrix(rep(1L, 6), nrow=2)

## Add segments to the GDS file
RAIDS:::appendGDSgenotypeMat(gds=tmpGDS, matG=newGenotype)

## Read genotype information from GDS file
## The return matrix should be a combination of both initial matrix
## and new matrix (column binded)
read.gdsn(index.gdsn(node=tmpGDS, path="genotype"))

## Close GDS file
closefn.gds(gdsfile=tmpGDS)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

`appendGDSRefSample` *Append fields related to samples into a GDS file*

Description

This function appends the fields related to samples into a GDS file. The information is extracted from the `data.frame` passed to the function and is added to the "sample.annot" and "sample.id" nodes. The "sample.id" and "sample.annot" nodes must already exist. If the samples are part of a study, the function addStudyGDSSample() must be used.

Usage

```
appendGDSRefSample(
  gdsReference,
  dfPedReference,
  batch = 1,
  listSamples = NULL,
  verbose = TRUE
)
```

Arguments

`gdsReference`	an object of class gds.class (a GDS file), the opened GDS file.
`dfPedReference`	a `data.frame` with the information about the sample(s). The `data.frame` must have the columns: "sample.id", "Name.ID", "sex", "pop.group" and "superPop". The unique identifier for the sample(s) is the "Name.ID" column and the row names of the `data.frame` must correspond to the "Name.ID" column.

batch	a integer representing the batch identifier.
listSamples	a vector of character string with the selected sample(s). If NULL, all samples are used.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: TRUE.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Create a temporary GDS file in an test directory
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_03.gds")

## Create and open the GDS file
tmpGDS <- createfn.gds(filename=gdsFilePath)

## Create "sample.id" node (the node must be present)
add.gdsn(node=tmpGDS, name="sample.id", val=c("sample_01",
        "sample_02"))

## Create "sample.annot" node (the node must be present)
add.gdsn(node=tmpGDS, name="sample.annot", val=data.frame(
            Name.ID=c("sample_01", "sample_02"),
            sex=c(1,1),  # 1:Male  2: Female
            pop.group=c("ACB", "ACB"),
            superPop=c("AFR", "AFR"),
            batch=c(1, 1),
            stringsAsFactors=FALSE))

sync.gds(gdsfile=tmpGDS)

## Create a data.frame with information about samples
sample_info <- data.frame(Name.ID=c("sample_04", "sample_05",
                                    "sample_06"),
                          sex=c(1,2,1),  # 1:Male  2: Female
                          pop.group=c("ACB", "ACB", "ACB"),
                          superPop=c("AFR", "AFR", "AFR"),
                          stringsAsFactors=FALSE)

## The row names must be the sample identifiers
rownames(sample_info) <- sample_info$Name.ID

## Add information about 2 samples to the GDS file
```

```
RAIDS:::appendGDSRefSample(gdsReference=tmpGDS,
        dfPedReference=sample_info,
        batch=2, listSamples=c("sample_04", "sample_06"), verbose=FALSE)

## Read sample identifier list
## Only "sample_04" and "sample_06" should have been added
read.gdsn(index.gdsn(node=tmpGDS, path="sample.id"))

## Read sample information from GDS file
## Only "sample_04" and "sample_06" should have been added
read.gdsn(index.gdsn(node=tmpGDS, path="sample.annot"))

## Close GDS file
closefn.gds(gdsfile=tmpGDS)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

`appendGDSSampleOnly` *Append sample names into a GDS file*

Description

This function append the sample identifiers into the "samples.id" node of a GDS file.

Usage

```
appendGDSSampleOnly(gds, listSamples)
```

Arguments

`gds`	an object of class gds.class (a GDS file), the opened GDS file.
`listSample`	a `vector` of `character` string representing the sample identifiers to be added to GDS file.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Temporary GDS file in current directory
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_04.gds")

## Create and open the GDS file
GDS_file_tmp  <- createfn.gds(filename=gdsFilePath)

## Create "sample.id" node (the node must be present)
add.gdsn(node=GDS_file_tmp, name="sample.id", val=c("sample_01",
        "sample_02"))

sync.gds(gdsfile=GDS_file_tmp)

## Add information about 2 samples to the GDS file
RAIDS:::appendGDSSampleOnly(gds=GDS_file_tmp,
        listSamples=c("sample_03", "sample_04"))

## Read sample identifier list
## Only "sample_03" and "sample_04" should have been added
read.gdsn(index.gdsn(node=GDS_file_tmp, path="sample.id"))

## Close GDS file
closefn.gds(gdsfile=GDS_file_tmp)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

`calcAFMLRNA` *Compute the log likelihood ratio based on the coverage of each allele in a specific block (gene in the case of RNA-seq)*

Description

This function sums the log of read depth of the lowest depth divide by the total depth of the position minus of likelihood of the allelic fraction of 0.5 for a block. If the phase is known, the SNVs in the same haplotype are grouped together.

Usage

```
calcAFMLRNA(snpPosHetero)
```

Arguments

snpPosHetero — a data.frame containing the SNV information for a specific block (gene if RNA-seq). The data.frame must contain those columns:

- cnt.ref a single integer representing the coverage for the reference allele.
- cnt.alt a single integer representing the coverage for the alternative allele.
- phase a single integer indicating the phase of the variant if known, 3 if not known

Value

a list for the block with the information relative to the heterozygotes. The list contains:

- lR a single numeric representing the sum of the log of read depth of the lowest depth divide by the total depth of the position minus of likelihood of the allelic fraction of 0.5.
- aFraction a single numeric representing the allele fraction estimation.
- sumAlleleLow a integer representing the sum of the allele read depth of the lowest read allele depth
- sumAlleleHigh a integer representing the sum of the allele read depth of the highest read allele depth

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Loading demo dataset containing SNV information
data(snpPositionDemo)

## Only use a subset of heterozygote SNVs related to one block
subset <- snpPositionDemo[which(snpPositionDemo$block.id == 2750 &
                  snpPositionDemo$hetero), c("cnt.ref", "cnt.alt", "phase")]

result <- RAIDS:::calcAFMLRNA(subset)

head(result)
```

---

computeAlleleFraction *Compute the allelic fraction for each imbalanced segment*

Description

This function computes the allelic fraction for each segment different than 0.5. The allelic fraction of the segment can be decomposed in sub-segments.

Usage

```
computeAlleleFraction(snpPos, w = 10, cutOff = -3)
```

Arguments

snpPos	a data.frame containing the genotype information for a SNV dataset.
w	a single positive numeric representing the size of the window to compute the allelic fraction. Default: 10.
cutOff	a numeric representing the cut-off for considering a region imbalanced when comparing likelihood to gave allelic fraction change and likelihood not to have allelic fraction change. Default: -3.

Value

a matrix of numeric with 3 columns where each row represent a segment of imbalanced SNVs. The first column represents the position, in snpPos, of the first SNV in the segment. The second column represents the position, in the snpPos, of the last SNV in the segment. The third column represents the lower allelic frequency of the segment and is NA when the value cannot be calculated. The value NULL is returned when none of the SNVs tested positive for the imbalance.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Data frame with SNV information for the specified chromosome (chr 1)
snpInfo <- data.frame(cnt.tot=c(41, 17, 27, 15, 11, 37, 16, 32),
    cnt.ref=c(40, 17, 27, 15, 4, 14, 16, 32),
    cnt.alt=c(0, 0, 0, 0, 7, 23, 0, 0),
    snp.pos=c(3722256, 3722328, 3767522, 3868160, 3869467, 4712655,
        6085318, 6213145),
    snp.chr=c(rep(1, 8)),
    normal.geno=c(rep(1, 8)),
    pruned=c(TRUE, TRUE, FALSE, TRUE, FALSE, rep(TRUE, 3)),
    snp.index=c(160, 162, 204, 256, 259, 288, 366, 465),
    keep=rep(TRUE, 8), hetero=c(rep(FALSE, 4), TRUE, TRUE, rep(FALSE, 2)),
    homo=c(rep(TRUE, 4), FALSE, FALSE, TRUE, TRUE),
    lap=rep(-1, 8), LOH=rep(0, 8), imbAR=rep(-1, 8),
    stringAsFactor=FALSE)

## The function returns NULL when there is not imbalanced SNVs
RAIDS:::computeAlleleFraction(snpPos=snpInfo, w=10, cutOff=-3)
```

computeAllelicFractionDNA *Estimate the allelic fraction of the pruned SNVs for a specific DNA-seq profile*

Description

The function creates a `data.frame` containing the allelic fraction for the pruned SNV dataset specific to a DNA-seq profile

Usage

```
computeAllelicFractionDNA(
  gdsReference,
  gdsSample,
  currentProfile,
  studyID,
  chrInfo,
  minCov = 10L,
  minProb = 0.999,
  eProb = 0.001,
  cutOffLOH = -5,
  cutOffHomoScore = -3,
  wAR = 9L,
  verbose
)
```

Arguments

`gdsReference`	an object of class `gds.class` (a GDS file), the opened Reference GDS file.
`gdsSample`	an object of class `gds.class` (a GDS file), the opened Profile GDS file.
`currentProfile`	a `character` string corresponding to the sample identifier as used in `pruningSample` function.
`studyID`	a `character` string corresponding to the name of the study as used in `pruningSample` function.
`chrInfo`	a `vector` of `integer` values representing the length of the chromosomes.
`minCov`	a single positive `integer` representing the minimum required coverage. Default: `10L`.
`minProb`	a single `numeric` between `0` and `1` representing the probability that the calculated genotype call is correct. Default: `0.999`.
`eProb`	a single `numeric` between 0 and 1 representing the probability of sequencing error. Default: `0.001`.
`cutOffLOH`	a single `numeric` representing the cutoff, in log, for the homozygote score to assign a region as LOH. Default: `-5`.

`cutOffHomoScore`
: a single `numeric` representing the cutoff, in log, that the SNVs in a block are called homozygote by error. Default: `-3`.

`wAR`
: a single positive `integer` representing the size-1 of the window used to compute an empty box. Default: `9L`.

`verbose`
: a `logicial` indicating if the function should print message when running.

Value

a `data.frame` containing the allelic information for the pruned SNV dataset with coverage > `minCov`. The `data.frame` contains those columns:

- cnt.tot a `integer` representing the total allele count
- cnt.ref a `integer` representing the reference allele count
- cnt.alt a `integer` representing the alternative allele count
- snp.pos a `integer` representing the position on the chromosome
- snp.chr a `integer` representing the chromosome
- normal.geno a `integer` representing the genotype (0=wild-type reference; 1=heterozygote; 2=homozygote alternative; 3=unkown)
- pruned a `logical` indicating if the SNV is retained after pruning
- snp.index a `integer` representing the index position of the SNV in the Reference GDS file that contains all SNVs
- keep a `logical` indicating if the genotype exists for the SNV
- hetero a `logical` indicating if the SNV is heterozygote
- homo a `logical` indicating if the SNV is homozygote
- lap a `numeric` representing the lower allelic fraction
- LOH a `integer` indicating if the SNV is in an LOH region (0=not LOH, 1=in LOH)
- imbAR a `integer` indicating if the SNV is in an imbalanced region (-1=not classified as imbalanced or LOH, 0=in LOH; 1=tested positive for imbalance in at least 1 window)

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## Temporary Profile GDS file for one profile in temporary directory
fileProfile <- file.path(tempdir(), "ex1.gds")
```

```
## Copy the Profile GDS file demo that has been pruned and annotated
file.copy(file.path(dataDir, "ex1_demo_with_pruning_and_1KG_annot.gds"),
                 fileProfile)

## Open the reference GDS file (demo version)
gds1KG <- snpgdsOpen(fileGDS)

## Open Profile GDS file for one profile
profileGDS <- openfn.gds(fileProfile)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

    ## The function returns a data frame containing the allelic fraction info
    result <- RAIDS:::computeAllelicFractionDNA(gdsReference=gds1KG,
        gdsSample=profileGDS, currentProfile="ex1", studyID="MYDATA",
        chrInfo=chrInfo, minCov=10L,
        minProb=0.999, eProb=0.001, cutOffLOH=-5,
        cutOffHomoScore=-3, wAR=9L, verbose=FALSE)
    head(result)

    ## Close both GDS files (important)
    closefn.gds(profileGDS)
    closefn.gds(gds1KG)

    ## Remove Profile GDS file (created for demo purpose)
    unlink(fileProfile, force=TRUE)

}
```

`computeAllelicFractionRNA` *Estimate the allelic fraction of the pruned SNVs for a specific RNA-seq sample*

Description

The function creates a `data.frame` containing the allelic fraction for the pruned SNV dataset specific to a RNA-seq sample.

Usage

```
computeAllelicFractionRNA(
  gdsReference,
```

```
  gdsSample,
  gdsRefAnnot,
  currentProfile,
  studyID,
  blockID,
  chrInfo,
  minCov = 10L,
  minProb = 0.999,
  eProb = 0.001,
  cutOffLOH = -5,
  cutOffAR = 3,
  verbose
)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened Reference GDS file.
gdsSample	an object of class gds.class (a GDS file), the opened Profile GDS file.
gdsRefAnnot	an object of class gds.class (a GDS file), the opened Reference SNV Annotation GDS file.
currentProfile	a character string corresponding to the sample identifier as used in pruningSample function.
studyID	a character string corresponding to the name of the study as used in pruningSample function.
blockID	a character string corresponding to the field gene block in the GDS gdsRefAnnot to use split by gene.
chrInfo	a vector of integer values representing the length of the chromosomes.
minCov	a single positive integer representing the minimum required coverage. Default: 10L.
minProb	a single numeric between 0 and 1 representing the probability that the calculated genotype call is correct. Default: 0.999.
eProb	a single numeric between 0 and 1 representing the probability of sequencing error. Default: 0.001.
cutOffLOH	a single numeric log of the score to be LOH. Default: -5.
cutOffAR	a single numeric representing the cutoff, in log score, to tag SNVs located in a gene has having an allelic fraction different 0.5 Default: 3.
verbose	a logicial indicating if the function should print message when running.

Value

a data.frame containing the allelic information for the pruned SNV dataset with coverage > minCov. The data.frame contains those columns:

- cnt.tot a integer representing the total allele count
- cnt.ref a integer representing the reference allele count

- cnt.alt a `integer` representing the alternative allele count
- snp.pos a `integer` representing the position on the chromosome
- snp.chr a `integer` representing the chromosome
- normal.geno a `integer` representing the genotype (0=wild-type reference; 1=heterozygote; 2=homozygote alternative; 3=unkown)
- pruned a `logical` indicating if the SNV is retained after pruning
- snp.index a `integer` representing the index position of the SNV in the Reference GDS file that contains all SNVs
- keep a `logical` indicating if the genotype exists for the SNV
- hetero a `logical` indicating if the SNV is heterozygote
- homo a `logical` indicating if the SNV is homozygote
- block.id a `integer` indicating the unique identifier of the block in the Population Reference Annotation GDS file that contains the current SNV
- phase a `integer` indicating the phase of the variant if known, `3` if not known
- lap a `numeric` indicating lower allelic fraction
- LOH a `integer` indicating if the SNV is in an LOH region (0=not LOH, 1=in LOH)
- imbAR a `integer` indicating if the SNV is in an imbalanced region (-1=not classified as imbalanced or LOH, 0=in LOH; 1=tested positive for imbalance in at least 1 window)
- freq a `numeric` indicating the frequency of the variant in the the reference

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(SNPRelate)

#' ## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")
fileAnnotGDS <- file.path(dataDir, "ex1_good_small_1KG_Annot.gds")

## Temporary Profile GDS file for one profile in temporary directory
fileProfile <- file.path(tempdir(), "ex1.gds")

## Copy the Profile GDS file demo that has been pruned and annotated
file.copy(file.path(dataDir, "ex1_demo_with_pruning_and_1KG_annot.gds"),
                 fileProfile)

## Open the reference GDS file (demo version)
gds1KG <- snpgdsOpen(fileGDS)
gdsRefAnnot <- openfn.gds(fileAnnotGDS)

## Open Profile GDS file for one profile
```

```
profileGDS <- openfn.gds(fileProfile)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

    ## The function returns a data frame containing the allelic fraction info
    result <- RAIDS:::computeAllelicFractionRNA(gdsReference=gds1KG,
        gdsSample=profileGDS, gdsRefAnnot=gdsRefAnnot,
        currentProfile="ex1", studyID="MYDATA",
        blockID="GeneS.Ensembl.Hsapiens.v86",
        chrInfo=chrInfo, minCov=10L, minProb=0.999, eProb=0.001,
        cutOffLOH=-5, cutOffAR=3, verbose=FALSE)
    head(result)

    ## Close both GDS files (important)
    closefn.gds(profileGDS)
    closefn.gds(gds1KG)
    closefn.gds(gdsRefAnnot)

    ## Remove Profile GDS file (created for demo purpose)
    unlink(fileProfile, force=TRUE)

}
```

`computeAllelicImbDNAChr` *Verify if SNVs are in an imbalance region*

Description

The function verifies, for each SNV present in the data frame, if the SNV is in an imbalance region.

Usage

```
computeAllelicImbDNAChr(snpPos, chr, wAR = 10, cutOffEmptyBox = -3)
```

Arguments

`snpPos`	a `data.frame` containing the SNV information for the chromosome specified by the `chr` argument. The `data.frame` must contain: • cnt.tot a single `integer` representing the total coverage for the SNV. • cnt.ref a single `integer` representing the coverage for the reference allele. • cnt.alt a single `integer` representing the coverage for the alternative allele.

- snp.pos a single `integer` representing the SNV position.
- snp.chr a single `integer` representing the SNV chromosome.
- normal.geno a single `numeric` indicating the genotype of the SNV. The possibles are: `0` (wild-type homozygote), `1` (heterozygote), `2` (altenative homozygote), `3` indicating that the normal genotype is unknown.
- pruned a `logical` indicating if the SNV is retained after pruning
- snp.index a `integer` representing the index position of the SNV in the Reference GDS file that contains all SNVs
- keep a `logical` indicating if the genotype exists for the SNV
- hetero a `logical` indicating if the SNV is heterozygote
- homo a `logical` indicating if the SNV is homozygote
- lap a `numeric` indicating lower allelic fraction
- LOH a `integer` indicating if the SNV is in an LOH region (0=not LOH, 1=in LOH)

`chr` a single positive `integer` for the current chromosome.

`wAR` a single positive `integer` representing the size-1 of the window used to compute an empty box. Default: `10`.

`cutOffEmptyBox` a `numeric` representing the cut-off for considering a region imbalanced when comparing likelihood to be imbalanced and likelihood not to be imbalanced. Default: `-3`.

Value

a `vector` of `integer` indicating if the SNV is in an imbalanced region (-1=not classified as imbalanced or LOH, 0=in LOH; 1=tested positive for imbalance in at least 1 window). The vector as an entry for each SNV present in the input `snpPos`.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(gdsfmt)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## Open the reference GDS file (demo version)
gds1KG <- snpgdsOpen(fileGDS)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
```

```
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

    ## Data frame with SNV information for the specified chromosome (chr 1)
    snpInfo <- data.frame(cnt.tot=c(41, 17, 27, 15, 11, 37, 16, 32),
        cnt.ref=c(40, 17, 27, 15, 4, 14, 16, 32),
        cnt.alt=c(0, 0, 0, 0, 7, 23, 0, 0),
        snp.pos=c(3722256, 3722328, 3767522, 3868160, 3869467, 4712655,
            6085318, 6213145),
        snp.chr=c(rep(1, 8)),
        normal.geno=c(rep(1, 8)), pruned=c(TRUE, TRUE, FALSE, TRUE,
            FALSE, TRUE, TRUE, TRUE),
        pruned=c(TRUE, TRUE, FALSE, TRUE, FALSE, rep(TRUE, 3)),
        snp.index=c(160, 162, 204, 256, 259, 288, 366, 465),
        keep=rep(TRUE, 8),
        hetero=c(rep(FALSE, 4), TRUE, TRUE, rep(FALSE, 2)),
        homo=c(rep(TRUE, 4), FALSE, FALSE, TRUE, TRUE),
        lap=rep(-1, 8), LOH=rep(0, 8), imbAR=rep(-1, 8),
        stringAsFactor=FALSE)

    ## The function returns a data frame containing the information about
    ## the LOH regions in the specified chromosome
    result <- RAIDS:::computeAllelicImbDNAChr(snpPos=snpInfo, chr=1, wAR=10,
                cutOffEmptyBox=-3)
    head(result)

    ## Close GDS file (important)
    closefn.gds(gds1KG)

}
```

`computeAncestryFromSyntheticFile` *Select the optimal K and D parameters using the synthetic data and infer the ancestry of a specific profile*

Description

The function select the optimal K and D parameters for a specific profile. The results on the synthetic data are used for the parameter selection. Once the optimal parameters are selected, the ancestry is inferred for the specific profile.

Usage

```
computeAncestryFromSyntheticFile(
  gdsReference,
  gdsProfile,
  listFiles,
```

```
  currentProfile,
  spRef,
  studyIDSyn,
  np = 1L,
  listCatPop = c("EAS", "EUR", "AFR", "AMR", "SAS"),
  fieldPopIn1KG = "superPop",
  fieldPopInfAnc = "SuperPop",
  kList = seq(2, 15, 1),
  pcaList = seq(2, 15, 1),
  algorithm = c("exact", "randomized"),
  eigenCount = 32L,
  missingRate = NaN,
  verbose = FALSE
)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened 1KG GDS file.
gdsProfile	an object of class gds.class (a GDS file), the opened Profile GDS file.
listFiles	a vector of character strings representing the name of files that contain the results of ancestry inference done on the synthetic profiles for multiple values of *D* and *K*. The files must exist.
currentProfile	a character string representing the profile identifier of the current profile on which ancestry will be inferred.
spRef	a vector of character strings representing the known super population ancestry for the 1KG profiles. The 1KG profile identifiers are used as names for the vector.
studyIDSyn	a character string corresponding to the study identifier. The study identifier must be present in the GDS Sample file.
np	a single positive integer representing the number of threads. Default: 1L.
listCatPop	a vector of character string representing the list of possible ancestry assignations. Default: ("EAS", "EUR", "AFR", "AMR", "SAS").
fieldPopIn1KG	a character string representing the name of the column that contains the known ancestry for the reference profiles in the Reference GDS file.
fieldPopInfAnc	a character string representing the name of the column that will contain the inferred ancestry for the specified profiles. Default: "SuperPop".
kList	a vector of integer representing the list of values tested for the *K* parameter. The *K* parameter represents the number of neighbors used in the K-nearest neighbor analysis. If NULL, the value seq(2,15,1) is assigned. Default: seq(2,15,1).
pcaList	a vector of integer representing the list of values tested for the *D* parameter. The *D* parameter represents the number of dimensions used in the PCA analysis. If NULL, the value seq(2,15,1) is assigned. Default: seq(2,15,1).
algorithm	a character string representing the algorithm used to calculate the PCA. The 2 choices are "exact" (traditional exact calculation) and "randomized" (fast PCA with randomized algorithm introduced in Galinsky et al. 2016). Default: "exact".

eigenCount	a single integer indicating the number of eigenvectors that will be in the output of the snpgdsPCA function; if 'eigenCount' <= 0, then all eigenvectors are returned. Default: 32L.
missingRate	a numeric value representing the threshold missing rate at with the SNVs are discarded; the SNVs are retained in the snpgdsPCA with "<= missingRate" only; if NaN, no missing threshold. Default: NaN.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Value

a list containing 4 entries:

- pcaSample a list containing the information related to the eigenvectors. The list contains those 3 entries:
  - sample.id a character string representing the unique identifier of the current profile.
  - eigenvector.ref a matrix of numeric containing the eigenvectors for the reference profiles.
  - eigenvector a matrix of numeric containing the eigenvectors for the current profile projected on the PCA from the reference profiles.
- paraSample a list containing the results with different D and K values that lead to optimal parameter selection. The list contains those entries:
  - dfPCA a data.frame containing statistical results on all combined synthetic results done with a fixed value of D (the number of dimensions). The data.frame contains those columns:
    - D a numeric representing the value of D (the number of dimensions).
    - median a numeric representing the median of the minimum AUROC obtained (within super populations) for all combination of the fixed D value and all tested K values.
    - mad a numeric representing the MAD of the minimum AUROC obtained (within super populations) for all combination of the fixed D value and all tested K values.
    - upQuartile a numeric representing the upper quartile of the minimum AUROC obtained (within super populations) for all combination of the fixed D value and all tested K values.
    - k a numeric representing the optimal K value (the number of neighbors) for a fixed D value.
  - dfPop a data.frame containing statistical results on all combined synthetic results done with different values of D (the number of dimensions) and K (the number of neighbors). The data.frame contains those columns:
    - D a numeric representing the value of D (the number of dimensions).
    - K a numeric representing the value of K (the number of neighbors).
    - AUROC.min a numeric representing the minimum accuracy obtained by grouping all the synthetic results by super-populations, for the specified values of D and K.
    - AUROC a numeric representing the accuracy obtained by grouping all the synthetic results for the specified values of D and K.
    - Accu.CM a numeric representing the value of accuracy of the confusion matrix obtained by grouping all the synthetic results for the specified values of D and K.

  - dfAUROC a data.frame the summary of the results by super-population. The data.frame contains those columns:
    * pcaD a numeric representing the value of D (the number of dimensions).
    * K a numeric representing the value of K (the number of neighbors).
    * Call a character string representing the super-population.
    * L a numeric representing the lower value of the 95% confidence interval for the AUROC obtained for the fixed values of super-population, D and K.
    * AUR a numeric representing the AUROC obtained for the fixed values of super-population, D and K.
    * H a numeric representing the higher value of the 95% confidence interval for the AUROC obtained for the fixed values of super-population, D and K.
  - D a numeric representing the optimal D value (the number of dimensions) for the specific profile.
  - K a numeric representing the optimal K value (the number of neighbors) for the specific profile.
  - listD a numeric representing the optimal D values (the number of dimensions) for the specific profile. More than one D is possible.

- KNNSample a list containing the inferred ancestry using different D and K values. The list contains those entries:
  - sample.id a character string representing the unique identifier of the current profile.
  - matKNN a data.frame containing the inferred ancestry for different values of K and D. The data.frame contains those columns:
    * sample.id a character string representing the unique identifier of the current profile.
    * D a numeric representing the value of D (the number of dimensions) used to infer the ancestry.
    * K a numeric representing the value of K (the number of neighbors) used to infer the ancestry.
    * SuperPop a character string representing the inferred ancestry for the specified D and K values.
- Ancestry a data.frame containing the inferred ancestry for the current profile. The data.frame contains those columns:
  - sample.id a character string representing the unique identifier of the current profile.
  - D a numeric representing the value of D (the number of dimensions) used to infer the ancestry.
  - K a numeric representing the value of K (the number of neighbors) used to infer the ancestry.
  - SuperPop a character string representing the inferred ancestry.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library
library(gdsfmt)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

## The Reference GDS file
path1KG <- system.file("extdata/tests", package="RAIDS")

## Open the Reference GDS file
gdsRef <- snpgdsOpen(file.path(path1KG, "ex1_good_small_1KG.gds"))

## Path to the demo synthetic results files
## List of the KNN result files from PCA run on synthetic data
dataDirRes <- system.file("extdata/demoAncestryCall/ex1", package="RAIDS")
listFilesName <- dir(file.path(dataDirRes), ".rds")
listFiles <- file.path(file.path(dataDirRes) , listFilesName)

# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoAncestryCall", package="RAIDS")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

## Run the ancestry inference on one profile called 'ex1'
## The values of K and D used for the inference are selected using the
## synthetic results
resCall <- computeAncestryFromSyntheticFile(gdsReference=gdsRef,
                            gdsProfile=gdsProfile,
                            listFiles=listFiles,
                            currentProfile=c("ex1"),
                            spRef=demoKnownSuperPop1KG,
                            studyIDSyn=studyID, np=1L)

## The ancestry called with the optimal D and K values
resCall$Ancestry

## Close the GDS files (important)
closefn.gds(gdsProfile)
closefn.gds(gdsRef)
```

computeKNNRefSample *Run a k-nearest neighbors analysis on one specific profile*

Description

The function runs k-nearest neighbors analysis on a one specific profile. The function uses the 'knn' package.

Usage

```
computeKNNRefSample(
  listEigenvector,
  listCatPop = c("EAS", "EUR", "AFR", "AMR", "SAS"),
  spRef,
  fieldPopInfAnc = "SuperPop",
  kList = seq(2, 15, 1),
  pcaList = seq(2, 15, 1)
)
```

Arguments

listEigenvector	a list with 3 entries: 'sample.id', 'eigenvector.ref' and 'eigenvector'. The list represents the PCA done on the 1KG reference profiles and one specific profile projected onto it. The 'sample.id' entry must contain only one identifier (one profile).
listCatPop	a vector of character string representing the list of possible ancestry assignations. Default: c("EAS", "EUR", "AFR", "AMR", "SAS").
spRef	vector of character strings representing the known super population ancestry for the 1KG profiles. The 1KG profile identifiers are used as names for the vector.
fieldPopInfAnc	a character string representing the name of the column that will contain the inferred ancestry for the specified profile. Default: "SuperPop".
kList	a vector of integer representing the list of values tested for the *K* parameter. The *K* parameter represents the number of neighbors used in the K-nearest neighbor analysis. If NULL, the value seq(2,15,1) is assigned. Default: seq(2,15,1).
pcaList	a vector of integer representing the list of values tested for the *D* parameter. The D parameter represents the number of dimensions used in the PCA analysis. If NULL, the value seq(2, 15, 1) is assigned. Default: seq(2, 15, 1).

Value

a `list` containing 4 entries:

- `sample.id` a `vector` of `character` strings representing the identifier of the profile analysed.
- `matKNN` a `data.frame` containing the super population inference for the profile for different values of PCA dimensions `D` and k-neighbors values `K`. The fourth column title corresponds to the `fieldPopInfAnc` parameter. The `data.frame` contains 4 columns:
  - `sample.id` a `character` string representing the identifier of the profile analysed.
  - `D` a `numeric` strings representing the value of the PCA dimension used to infer the ancestry.
  - `K` a `numeric` strings representing the value of the k-neighbors used to infer the ancestry..
  - `fieldPopInfAnc` a `character` string representing the inferred ancestry.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Load the demo PCA on the synthetic profiles projected on the
## demo 1KG reference PCA
data(demoPCASyntheticProfiles)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

## The PCA with 1 profile projected on the 1KG reference PCA
## Only one profile is retained
pca <- demoPCASyntheticProfiles
pca$sample.id <- pca$sample.id[1]
pca$eigenvector <- pca$eigenvector[1, , drop=FALSE]

## Projects profile on 1KG PCA
results <- computeKNNRefSample(listEigenvector=pca,
    listCatPop=c("EAS", "EUR", "AFR", "AMR", "SAS"),
    spRef=demoKnownSuperPop1KG, fieldPopInfAnc="SuperPop",
    kList=seq(10, 15, 1), pcaList=seq(10, 15, 1))

## The assigned ancestry to the profile for different values of K and D
head(results$matKNN)
```

`computeKNNRefSynthetic` *Run a k-nearest neighbors analysis on a subset of the synthetic dataset*

Description

The function runs k-nearest neighbors analysis on a subset of the synthetic data set. The function uses the 'knn' package.

Usage

```
computeKNNRefSynthetic(
  gdsProfile,
  listEigenvector,
  listCatPop = c("EAS", "EUR", "AFR", "AMR", "SAS"),
  studyIDSyn,
  spRef,
  fieldPopInfAnc = "SuperPop",
  kList = seq(2, 15, 1),
  pcaList = seq(2, 15, 1)
)
```

Arguments

`gdsProfile`	an object of class `SNPRelate::SNPGDSFileClass`, the opened Profile GDS file.
`listEigenvector`	a `list` with 3 entries: 'sample.id', 'eigenvector.ref' and 'eigenvector'. The `list` represents the PCA done on the 1KG reference profiles and the synthetic profiles projected onto it.
`listCatPop`	a `vector` of `character` string representing the list of possible ancestry assignations. Default: `c("EAS", "EUR", "AFR", "AMR", "SAS")`.
`studyIDSyn`	a `character` string corresponding to the study identifier. The study identifier must be present in the Profile GDS file.
`spRef`	`vector` of `character` strings representing the known super population ancestry for the 1KG profiles. The 1KG profile identifiers are used as names for the `vector`.
`fieldPopInfAnc`	a `character` string representing the name of the column that will contain the inferred ancestry for the specified data set. Default: `"SuperPop"`.
`kList`	a `vector` of `integer` representing the list of values tested for the K parameter. The K parameter represents the number of neighbors used in the K-nearest neighbors analysis. If `NULL`, the value `seq(2, 15, 1)` is assigned. Default: `seq(2, 15, 1)`.
`pcaList`	a `vector` of `integer` representing the list of values tested for the D parameter. The D parameter represents the number of dimensions used in the PCA analysis. If `NULL`, the value `seq(2, 15, 1)` is assigned. Default: `seq(2, 15, 1)`.

Value

a `list` containing 4 entries:

- `sample.id` a `vector` of `character` strings representing the identifiers of the synthetic profiles analysed.

- sample1Kg a vector of character strings representing the identifiers of the 1KG reference profiles used to generate the synthetic profiles.
- sp a vector of character strings representing the known super population ancestry of the 1KG reference profiles used to generate the synthetic profiles.
- matKNN a data.frame containing the super population inference for each synthetic profiles for different values of PCA dimensions D and k-neighbors values K. The fourth column title corresponds to the fieldPopInfAnc parameter. The data.frame contains 4 columns:
  - sample.id a character string representing the identifier of the synthetic profile analysed.
  - D a numeric strings representing the value of the PCA dimension used to infer the super population.
  - K a numeric strings representing the value of the k-neighbors used to infer the super population.
  - fieldPopInfAnc value a character string representing the inferred ancestry.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Load the demo PCA on the synthetic profiles projected on the
## demo 1KG reference PCA
data(demoPCASyntheticProfiles)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

## Projects synthetic profiles on 1KG PCA
results <- computeKNNRefSynthetic(gdsProfile=gdsProfile,
    listEigenvector=demoPCASyntheticProfiles,
    listCatPop=c("EAS", "EUR", "AFR", "AMR", "SAS"), studyIDSyn=studyID,
    spRef=demoKnownSuperPop1KG)

## The inferred ancestry for the synthetic profiles for different values
## of D and K
head(results$matKNN)
```

```
## Close Profile GDS file (important)
closefn.gds(gdsProfile)
```

`computeLOHBlocksDNAChr` *Identify regions of LOH on one chromosome using homozygote SNVs*

Description

The function identifies regions of LOH on a specific chromosome using the homozygote SNVs present on the chromosome.

Usage

```
computeLOHBlocksDNAChr(gdsReference, chrInfo, snpPos, chr, genoN = 1e-04)
```

Arguments

`gdsReference`	an object of class `SNPRelate::SNPGDSFileClass`, an opened Reference GDS file.
`chrInfo`	a `vector` of `integer` representing the length of the chromosomes. As an example, the information ca be obtained from package 'BSgenome.Hsapiens.UCSC.hg38'.
`snpPos`	a `data.frame` containing the SNV information for the chromosome specified by the `chr` argument. The `data.frame` must contain: • cnt.tot a single `integer` representing the total coverage for the SNV. • cnt.ref a single `integer` representing the coverage for the reference allele. • cnt.alt a single `integer` representing the coverage for the alternative allele. • snp.pos a single `integer` representing the SNV position. • snp.chr a single `integer` representing the SNV chromosome. • normal.geno a single `numeric` indicating the genotype of the SNV. The possibles are: `0` (wild-type homozygote), `1` (heterozygote), `2` (altenative homozygote), `3` indicating that the normal genotype is unknown. • pruned a `logical` indicating if the SNV is retained after pruning • snp.index a `integer` representing the index position of the SNV in the Reference GDS file that contains all SNVs • keep a `logical` indicating if the genotype exists for the SNV • hetero a `logical` indicating if the SNV is heterozygote • homo a `logical` indicating if the SNV is homozygote
`chr`	a single positive `integer` for the current chromosome. The `chrInfo` parameter must contain the value for the specified chromosome.
`genoN`	a single `numeric` between 0 and 1 representing the probability of sequencing error. Default: `0.0001`.

Value

a data.frame with the informations about LOH on a specific chromosome. The data.frame contains those columns:

- chr a integer representing the current chromosome
- start a integer representing the starting position on the box containing only homozygote SNVs (or not SNV). The first box starts at position 1.
- end a integer representing the end position on the box containing only homozygote SNVs (or not SNV). The last box ends at the length of the chromosome.
- logLHR a numeric representing the LOH score basde on population frequencies. It is the sum of the log10 of the frequencies of the observed gegenotype minus the the sum of the log10 of the higher frequent genotype. (-100 when normal genotype are present)
- LH1 a numeric representing the probability to be heterozygote based on the coverage of each allele when normal genotype is present
- LM1 a numeric representing the max probability for the read coverage at the position
- homoScore a numeric representing LH1 - LM1
- nbSNV a integer representing th number of SNVs in the box
- nbPruned a integer representing the number of pruned SNVs in the box
- nbNorm a integer representing of the number of heterozygote genotypes for the normal SNVs in the block
- LOH a integer representing a flag, if 1 it means the block is satisfying the criteria to be LOH. The value is not assigned in this function; the value 0 is assigned

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo Reference GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## Open the Reference GDS file (demo version)
gds1KG <- snpgdsOpen(fileGDS)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <-
        GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]
```

```
    ## Data frame with SNV information for the specified chromosome (chr 1)
    snpInfo <- data.frame(cnt.tot=c(41, 17, 27, 15, 11, 37, 16, 32),
        cnt.ref=c(40, 17, 27, 15, 4, 14, 16, 32),
        cnt.alt=c(0, 0, 0, 0, 7, 23, 0, 0),
        snp.pos=c(3722256, 3722328, 3767522, 3868160, 3869467, 4712655,
            6085318, 6213145),
        snp.chr=c(rep(1, 8)),
        normal.geno=c(rep(3, 8)), pruned=c(TRUE, TRUE, FALSE, TRUE, FALSE,
            TRUE, TRUE, TRUE),
        pruned=c(TRUE, TRUE, FALSE, TRUE, FALSE, rep(TRUE, 3)),
        snp.index=c(160, 162, 204, 256, 259, 288, 366, 465),
        keep=rep(TRUE, 8), hetero=c(rep(FALSE, 4), TRUE,
            TRUE, rep(FALSE, 2)),
        homo=c(rep(TRUE, 4), FALSE, FALSE, TRUE, TRUE),
        stringAsFactor=FALSE)

    ## The function returns a data frame containing the information about
    ## the LOH regions in the specified chromosome
    result <- RAIDS:::computeLOHBlocksDNAChr(gdsReference=gds1KG,
        chrInfo=chrInfo, snpPos=snpInfo, chr=1L, genoN=0.0001)
    head(result)

    ## Close Reference GDS file (important)
    closefn.gds(gds1KG)

}
```

`computePCAMultiSynthetic` *Project synthetic profiles onto existing principal component axes generated using the reference 1KG profiles*

Description

The function projects the synthetic profiles onto existing principal component axes generated using the reference 1KG profiles. The reference profiles used to generate the synthetic profiles have previously been removed from the set of reference profiles.

Usage

```
computePCAMultiSynthetic(
  gdsProfile,
  listPCA,
  sampleRef,
  studyIDSyn,
  verbose = FALSE
)
```

Arguments

gdsProfile	an object of class gds.class (a GDS file), an opened Profile GDS file.
listPCA	a list containing the PCA object generated with the 1KG reference profiles (excluding the ones used to generate the synthetic data set) in an entry called "pca.unrel".
sampleRef	a vector of character strings representing the identifiers of the 1KG reference profiles that have been used to generate the synthetic profiles that are going to be analysed here. The sub-continental identifiers are used as names for the vector.
studyIDSyn	a character string corresponding to the study identifier. The study identifier must be present in the Profile GDS file.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Value

a list containing 3 entries:

- sample.id a vector of character strings representing the identifiers of the synthetic profiles that have been projected onto the 1KG PCA.
- eigenvector.ref a matrix of numeric with the eigenvectors of the 1KG reference profiles used to generate the PCA.
- eigenvector a matrix of numeric with the eigenvectors of the synthetic profiles projected onto the 1KG PCA.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Loading demo PCA on subset of 1KG reference dataset
data(demoPCA1KG)

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")

# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

samplesRM <- c("HG00246", "HG00325", "HG00611", "HG01173", "HG02165",
    "HG01112", "HG01615", "HG01968", "HG02658", "HG01850", "HG02013",
    "HG02465", "HG02974", "HG03814", "HG03445", "HG03689", "HG03789",
    "NA12751", "NA19107", "NA18548", "NA19075", "NA19475", "NA19712",
    "NA19731", "NA20528", "NA20908")
names(samplesRM) <- c("GBR", "FIN", "CHS","PUR", "CDX", "CLM", "IBS",
    "PEL", "PJL", "KHV", "ACB", "GWD", "ESN", "BEB", "MSL", "STU", "ITU",
```

```
    "CEU", "YRI", "CHB", "JPT", "LWK", "ASW", "MXL", "TSI", "GIH")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

## Projects synthetic profiles on 1KG PCA
results <- computePCAMultiSynthetic(gdsProfile=gdsProfile,
    listPCA=demoPCA1KG,
    sampleRef=samplesRM, studyIDSyn=studyID, verbose=FALSE)

## The eigenvectors for the synthetic profiles
head(results$eigenvector)

## Close Profile GDS file (important)
closefn.gds(gdsProfile)
```

`computePCARefRMMulti` *Calculate Principal Component Analysis (PCA) on SNV genotype data set*

Description

The functions calculates the principal component analysis (PCA) for a list of pruned SNVs present in a Profile GDS file. The snpgdsPCA function is used to do the calculation.

Usage

```
computePCARefRMMulti(
  gdsProfile,
  refProfileIDs,
  listRM,
  np = 1L,
  algorithm = "exact",
  eigenCount = 32L,
  missingRate = 0.025,
  verbose
)
```

Arguments

`gdsProfile`	an object of class SNPGDSFileClass, the opened Profile GDS file.
`refProfileIDs`	a `vector` of reference 1KG profile identifiers that are present in the Profile GDS file. Those profiles minus the one present in the `listRM` vector will be used to run the PCA analysis.
`listRM`	a `vector` of `character` strings containing the identifiers for the reference samples that need to be removed for the PCA analysis.

np	a single positive integer representing the number of CPU that will be used. Default: 1L.
algorithm	a character string representing the algorithm used to calculate the PCA. The 2 choices are "exact" (traditional exact calculation) and "randomized" (fast PCA with randomized algorithm introduced in Galinsky et al. 2016). Default: "exact".
eigenCount	a single integer indicating the number of eigenvectors that will be in the output of the snpgdsPCA function; if 'eigenCount' <= 0, then all eigenvectors are returned. Default: 32L.
missingRate	a numeric value representing the threshold missing rate at with the SNVs are discarded; the SNVs are retained in the snpgdsPCA function with "<= missingRate" only; if NaN, no missing threshold. Default: 0.025.
verbose	a logical indicating if message information should be printed.

Value

a list containing 2 entries:

- pruned a vector of SNV identifiers specifying selected SNVs for the PCA analysis.
- pca.unrel a snpgdsPCAClass object containing the eigenvalues as generated by snpgdsPCA function.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library
library(SNPRelate)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

## Profiles that should be removed from the PCA analysis
## Those profiles has been used to generate the synthetic data set
samplesRM <- c("HG00246", "HG00325", "HG00611", "HG01173", "HG02165",
    "HG01112", "HG01615", "HG01968", "HG02658", "HG01850", "HG02013",
    "HG02465", "HG02974", "HG03814", "HG03445", "HG03689", "HG03789",
    "NA12751", "NA19107", "NA18548", "NA19075", "NA19475", "NA19712",
    "NA19731", "NA20528", "NA20908")
```

```
## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

## Compute PCA for the 1KG reference profiles excluding
## the profiles used to generate the synthetic profiles
results <- RAIDS:::computePCARefRMMulti(gdsProfile=gdsProfile,
    refProfileIDs=names(demoKnownSuperPop1KG), listRM=samplesRM, np=1L,
    algorithm="exact", eigenCount=32L, missingRate=0.025, verbose=FALSE)

## The PCA on the pruned SNVs data set for selected profiles
head(results$pca.unrel$eigenvect)

## Close Profile GDS file (important)
closefn.gds(gdsProfile)
```

`computePCARefSample` *Project specified profile onto PCA axes generated using known reference profiles*

Description

This function generates a PCA using the know reference profiles. Them, it projects the specified profile onto the PCA axes.

Usage

```
computePCARefSample(
  gdsProfile,
  currentProfile,
  studyIDRef = "Ref.1KG",
  np = 1L,
  algorithm = c("exact", "randomized"),
  eigenCount = 32L,
  missingRate = NaN,
  verbose = FALSE
)
```

Arguments

`gdsProfile`	an object of class gds.class, an opened Profile GDS file.
`currentProfile`	a single `character` string representing the profile identifier.
`studyIDRef`	a single `character` string representing the study identifier.
`np`	a single positive `integer` representing the number of CPU that will be used. Default: `1L`.

algorithm	a character string representing the algorithm used to calculate the PCA. The 2 choices are "exact" (traditional exact calculation) and "randomized" (fast PCA with randomized algorithm introduced in Galinsky et al. 2016). Default: "exact".
eigenCount	a single integer indicating the number of eigenvectors that will be in the output of the snpgdsPCA function; if 'eigen.cnt' <= 0, then all eigenvectors are returned. Default: 32L.
missingRate	a numeric value representing the threshold missing rate at with the SNVs are discarded; the SNVs are retained in the snpgdsPCA with "<= missingRate" only; if NaN, no missing threshold. Default: NaN.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Value

a list containing 3 entries:

- sample.id a character string representing the unique identifier of the analyzed profile.
- eigenvector.ref a matrix of numeric representing the eigenvectors of the reference profiles.
- eigenvector a matrix of numeric representing the eigenvectors of the analyzed profile.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library
library(gdsfmt)

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoAncestryCall", package="RAIDS")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

## Project a profile onto a PCA generated using reference profiles
## The reference profiles come from 1KG
resPCA <- computePCARefSample(gdsProfile=gdsProfile,
    currentProfile=c("ex1"), studyIDRef="Ref.1KG", np=1L, verbose=FALSE)
resPCA$sample.id
resPCA$eigenvector

## Close the GDS files (important)
```

```
closefn.gds(gdsProfile)
```

---

computePoolSyntheticAncestryGr

Run a PCA analysis and a K-nearest neighbors analysis on a small set of synthetic data using all 1KG profiles except the ones used to generate the synthetic profiles

---

Description

The function runs a PCA analysis using 1 synthetic profile from each sub-continental population. The reference profiles used to create those synthetic profiles are first removed from the list of 1KG reference profiles that generates the reference PCA. Then, the retained synthetic profiles are projected on the 1KG PCA space. Finally, a K-nearest neighbors analysis using a range of K and D values is done.

Usage

```
computePoolSyntheticAncestryGr(
  gdsProfile,
  sampleRM,
  spRef,
  studyIDSyn,
  np = 1L,
  listCatPop = c("EAS", "EUR", "AFR", "AMR", "SAS"),
  fieldPopInfAnc = "SuperPop",
  kList = seq(2, 15, 1),
  pcaList = seq(2, 15, 1),
  algorithm = c("exact", "randomized"),
  eigenCount = 32L,
  missingRate = 0.025,
  verbose = FALSE
)
```

Arguments

gdsProfile	an object of class SNPRelate::SNPGDSFileClass, the opened Profile GDS file.
sampleRM	a vector of character strings representing the identifiers of the 1KG reference profiles that should not be used to create the reference PCA. There should be one per sub-continental population. Those profiles are removed because those have been used to generate the synthetic profiles that are going to be analysed here. The sub-continental identifiers are used as names for the vector.
spRef	vector of character strings representing the known super population ancestry for the 1KG profiles. The 1KG profile identifiers are used as names for the vector.

studyIDSyn	a character string corresponding to the study identifier. The study identifier must be present in the Profile GDS file.
np	a single positive integer representing the number of threads. Default: 1L.
listCatPop	a vector of character string representing the list of possible ancestry assignations. Default: ("EAS", "EUR", "AFR", "AMR", "SAS").
fieldPopInfAnc	a character string representing the name of the column that will contain the inferred ancestry for the specified dataset. Default: "SuperPop".
kList	a vector of integer representing the list of values tested for the *K* parameter. The *K* parameter represents the number of neighbors used in the K-nearest neighbor analysis. If NULL, the value seq(2,15,1) is assigned. Default: seq(2,15,1).
pcaList	a vector of integer representing the list of values tested for the *D* parameter. The *D* parameter represents the number of dimensions used in the PCA analysis. If NULL, the value seq(2,15,1) is assigned. Default: seq(2,15,1).
algorithm	a character string representing the algorithm used to calculate the PCA. The 2 choices are "exact" (traditional exact calculation) and "randomized" (fast PCA with randomized algorithm introduced in Galinsky et al. 2016). Default: "exact".
eigenCount	a single integer indicating the number of eigenvectors that will be in the output of the snpgdsPCA function; if 'eigenCount' <= 0, then all eigenvectors are returned. Default: 32L.
missingRate	a numeric value representing the threshold missing rate at with the SNVs are discarded; the SNVs are retained in the snpgdsPCA function with "<= missingRate" only; if NaN, no missing threshold. Default: 0.025.
verbose	a logical indicating if message information should be printed. Default: FALSE.

Value

a list containing the following entries:

- sample.ida vector of character strings representing the identifiers of the synthetic profiles.
- sample1Kga vector of character strings representing the identifiers of the reference 1KG profiles used to generate the synthetic profiles.
- spa vector of character strings representing the known ancestry for the reference 1KG profiles used to generate the synthetic profiles.
- matKNNa data.frame containing 4 columns. The first column 'sample.id' contains the name of the synthetic profile. The second column 'D' represents the dimension D used to infer the ancestry. The third column 'K' represents the number of neighbors K used to infer the ancestry. The fourth column 'SuperPop' contains the inferred ancestry.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library
library(gdsfmt)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)


# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

samplesRM <- c("HG00246", "HG00325", "HG00611", "HG01173", "HG02165",
    "HG01112", "HG01615", "HG01968", "HG02658", "HG01850", "HG02013",
    "HG02465", "HG02974", "HG03814", "HG03445", "HG03689", "HG03789",
    "NA12751", "NA19107", "NA18548", "NA19075", "NA19475", "NA19712",
    "NA19731", "NA20528", "NA20908")
names(samplesRM) <- c("GBR", "FIN", "CHS","PUR", "CDX", "CLM", "IBS",
    "PEL", "PJL", "KHV", "ACB", "GWD", "ESN", "BEB", "MSL", "STU", "ITU",
    "CEU", "YRI", "CHB", "JPT", "LWK", "ASW", "MXL", "TSI", "GIH")

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

## Run a PCA analysis and a K-nearest neighbors analysis on a small set
## of synthetic data
results <- computePoolSyntheticAncestryGr(gdsProfile=gdsProfile,
    sampleRM=samplesRM, studyIDSyn=studyID, np=1L,
    spRef=demoKnownSuperPop1KG,
    kList=seq(10,15,1), pcaList=seq(10,15,1), eigenCount=15L)

## The ancestry inference for the synthetic data using
## different K and D values
head(results$matKNN)

## Close Profile GDS file (important)
closefn.gds(gdsProfile)
```

---

computeSyntheticConfMat

Calculate the confusion matrix of the inferences for specific values of D and K using the inferred ancestry results from the synthetic profiles.

Description

The function calculates the confusion matrix of the inferences for fixed values of D and K using the inferred ancestry results done on the synthetic profiles.

Usage

```
computeSyntheticConfMat(
  matKNN,
  matKNNAncestryColumn,
  pedCall,
  pedCallAncestryColumn,
  listCall
)
```

Arguments

matKNN	a data.frame containing the inferred ancestry results for fixed values of *D* and *K*. The data.frame must contained those columns: "sample.id", "D", "K" and the fourth column name must correspond to the matKNNAncestryColumn argument.
matKNNAncestryColumn	a character string representing the name of the column that contains the inferred ancestry for the specified synthetic profiles. The column must be present in the matKNN argument.
pedCall	a data.frame containing the information about the super-population information from the 1KG GDS file for profiles used to generate the synthetic profiles. The data.frame must contained a column named as the pedCallAncestryColumn argument.
pedCallAncestryColumn	a character string representing the name of the column that contains the known ancestry for the reference profiles in the Reference GDS file. The column must be present in the pedCall argument.
listCall	a vector of character strings representing the list of possible ancestry assignations.

Value

list containing 2 entries:

- confMat a matrix representing the confusion matrix
- matAccuracy a data.frame containing the statistics associated to the confusion matrix

Author(s)

Pascal Belleau, Astrid Deschênes and Alex Krasnitz

Examples

```
## Loading demo dataset containing pedigree information for synthetic
## profiles and known ancestry of the profiles used to generate the
## synthetic profiles
data(pedSynthetic)
```

```
## Loading demo dataset containing the inferred ancestry results
## for the synthetic data
data(matKNNSynthetic)

## The inferred ancestry results for the synthetic data using
## values of D=6 and K=5
matKNN <- matKNNSynthetic[matKNNSynthetic$K == 6 & matKNNSynthetic$D == 5, ]

## Compile the confusion matrix using the
## synthetic profiles for fixed values of  D and K values
results <- RAIDS:::computeSyntheticConfMat(matKNN=matKNN,
    matKNNAncestryColumn="SuperPop",
    pedCall=pedSynthetic, pedCallAncestryColumn="superPop",
    listCall=c("EAS", "EUR", "AFR", "AMR", "SAS"))

results$confMat
results$matAccuracy
```

`computeSyntheticROC` *Calculate the AUROC of the inferences for specific values of D and K using the inferred ancestry results from the synthetic profiles.*

Description

The function calculates the AUROC of the inferences for specific values of D and K using the inferred ancestry results from the synthetic profiles. The calculations are done on each super-population separately as well as on all the results together.

Usage

```
computeSyntheticROC(
  matKNN,
  matKNNAncestryColumn,
  pedCall,
  pedCallAncestryColumn,
  listCall = c("EAS", "EUR", "AFR", "AMR", "SAS")
)
```

Arguments

`matKNN`	a `data.frame` containing the inferred ancestry results for fixed values of *D* and *K*. On of the column names of the `data.frame` must correspond to the `matKNNAncestryColumn` argument.
`matKNNAncestryColumn`	a `character` string representing the name of the column that contains the inferred ancestry for the specified synthetic profiles. The column must be present in the `matKNN` argument.

pedCall — a data.frame containing the information about the super-population information from the 1KG GDS file for profiles used to generate the synthetic profiles. The data.frame must contained a column named as the pedCallAncestryColumn argument. The row names must correspond to the sample identifiers (mandatory).

pedCallAncestryColumn — a character string representing the name of the column that contains the known ancestry for the reference profiles in the Reference GDS file. The column must be present in the pedCall argument.

listCall — a vector of character strings representing the list of all possible ancestry assignations. Default: c("EAS", "EUR", "AFR", "AMR", "SAS").

Value

list containing 3 entries:

- matAUROC.All a data.frame containing the AUROC for all the ancestry results.
- matAUROC.Call a data.frame containing the AUROC information for each super-population.
- listROC.Call a list containing the output from the roc function for each super-population.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Loading demo dataset containing pedigree information for synthetic
## profiles and known ancestry of the profiles used to generate the
## synthetic profiles
data(pedSynthetic)

## Loading demo dataset containing the inferred ancestry results
## for the synthetic data
data(matKNNSynthetic)

## The inferred ancestry results for the synthetic data using
## values of D=6 and K=5
matKNN <- matKNNSynthetic[matKNNSynthetic$K == 6 & matKNNSynthetic$D == 5, ]

## Compile statistics from the
## synthetic profiles for fixed values of D and K
results <- RAIDS:::computeSyntheticROC(matKNN=matKNN,
    matKNNAncestryColumn="SuperPop",
    pedCall=pedSynthetic, pedCallAncestryColumn="superPop",
    listCall=c("EAS", "EUR", "AFR", "AMR", "SAS"))

results$matAUROC.All
results$matAUROC.Call
results$listROC.Call
```

createStudy2GDS1KG *Create the Profile GDS file(s) for one or multiple specific profiles using the information from a RDS Sample description file and the 1KG GDS file*

Description

The function uses the information for the Reference GDS file and the RDS Sample Description file to create the Profile GDS file. One Profile GDS file is created per profile. One Profile GDS file will be created for each entry present in the `listProfiles` parameter.

Usage

```
createStudy2GDS1KG(
  pathGeno = file.path("data", "sampleGeno"),
  filePedRDS = NULL,
  pedStudy = NULL,
  fileNameGDS,
  batch = 1,
  studyDF,
  listProfiles = NULL,
  pathProfileGDS = NULL,
  genoSource = c("snp-pileup", "generic", "VCF"),
  verbose = FALSE
)
```

Arguments

`pathGeno`	a `character` string representing the path to the directory containing the VCF output of SNP-pileup for each sample. The SNP-pileup files must be compressed (gz files) and have the name identifiers of the samples. A sample with "Name.ID" identifier would have an associated file called if genoSource is "VCF", then "Name.ID.vcf.gz", if genoSource is "generic", then "Name.ID.generic.txt.gz" if genoSource is "snp-pileup", then "Name.ID.txt.gz".
`filePedRDS`	a `character` string representing the path to the RDS file that contains the information about the sample to analyse. The RDS file must include a `data.frame` with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in `character` strings. The `data.frame` must contain the information for all the samples passed in the `listSamples` parameter. Only `filePedRDS` or `pedStudy` can be defined.
`pedStudy`	a `data.frame` with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in `character` strings (no factor). The `data.frame` must contain the information for all the samples passed in the `listSamples` parameter. Only `filePedRDS` or `pedStudy` can be defined.

fileNameGDS	a character string representing the file name of the Reference GDS file. The file must exist.
batch	a single positive integer representing the current identifier for the batch. Beware, this field is not stored anymore. Default: 1.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).
listProfiles	a vector of character string corresponding to the profile identifiers that will have a Profile GDS file created. The profile identifiers must be present in the "Name.ID" column of the Profile RDS file passed to the filePedRDS parameter. If NULL, all profiles present in the filePedRDS are selected. Default: NULL.
pathProfileGDS	a character string representing the path to the directory where the Profile GDS files will be created. Default: NULL.
genoSource	a character string with two possible values: 'snp-pileup', 'generic' or 'VCF'. It specifies if the genotype files are generated by snp-pileup (Facets) or are a generic format CSV file with at least those columns: 'Chromosome', 'Position', 'Ref', 'Alt', 'Count', 'File1R' and 'File1A'. The 'Count' is the depth at the specified position; 'FileR' is the depth of the reference allele and 'File1A' is the depth of the specific alternative allele. Finally the file can be a VCF file with at least those genotype fields: GT, AD, DP.
verbose	a logical indicating if message information should be printed. Default: FALSE.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## The data.frame containing the information about the study
## The 3 mandatory columns: "study.id", "study.desc", "study.platform"
## The entries should be strings, not factors (stringsAsFactors=FALSE)
studyDF <- data.frame(study.id = "MYDATA",
                      study.desc = "Description",
                      study.platform = "PLATFORM",
                      stringsAsFactors = FALSE)

## The data.frame containing the information about the samples
## The entries should be strings, not factors (stringsAsFactors=FALSE)
samplePED <- data.frame(Name.ID=c("ex1", "ex2"),
                    Case.ID=c("Patient_h11", "Patient_h12"),
                    Diagnosis=rep("Cancer", 2),
```

```
                    Sample.Type=rep("Primary Tumor", 2),
                    Source=rep("Databank B", 2), stringsAsFactors=FALSE)
rownames(samplePED) <- samplePED$Name.ID

## Create the Profile GDS File for samples in 'listSamples' vector
## (in this case, samples "ex1")
## The Profile GDS file is created in the pathProfileGDS directory
result <- createStudy2GDS1KG(pathGeno=dataDir,
            pedStudy=samplePED, fileNameGDS=fileGDS,
            studyDF=studyDF, listProfiles=c("ex1"),
            pathProfileGDS=tempdir(),
            genoSource="snp-pileup",
            verbose=FALSE)

## The function returns 0L when successful
result

## The Profile GDS file 'ex1.gds' has been created in the
## specified directory
list.files(tempdir())

## Remove Profile GDS file (created for demo purpose)
unlink(file.path(tempdir(), "ex1.gds"), force=TRUE)
```

---

demoKnownSuperPop1KG *The known super population ancestry of the demo 1KG reference profiles.*

Description

The object is a `vector`.

Usage

```
data(demoKnownSuperPop1KG)
```

Format

The `vector` containing the know super population ancestry for the demo 1KG reference profiles.

Details

This object can be used to test the `computeKNNRefSynthetic` and `computePoolSyntheticAncestryGr` functions.

Value

The `vector` containing the know super population ancestry for the demo 1KG reference profiles.

See Also

- computeKNNRefSynthetic for running a k-nearest neighbors analysis on a subset of the synthetic data set.
- computePoolSyntheticAncestryGr for running a PCA analysis using 1 synthetic profile from each sub-continental population.

Examples

```
## Required library
library(gdsfmt)

## Load the demo PCA on the synthetic profiles projected on the
## demo 1KG reference PCA
data(demoPCASyntheticProfiles)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

## Projects synthetic profiles on 1KG PCA
results <- computeKNNRefSynthetic(gdsProfile=gdsProfile,
    listEigenvector=demoPCASyntheticProfiles,
    listCatPop=c("EAS", "EUR", "AFR", "AMR", "SAS"), studyIDSyn=studyID,
    spRef=demoKnownSuperPop1KG)

## The inferred ancestry for the synthetic profiles for different values
## of D and K
head(results$matKNN)

## Close Profile GDS file (important)
closefn.gds(gdsProfile)
```

---

demoPCA1KG *The PCA results of the demo 1KG reference dataset for demonstration purpose. Beware that the PCA has been run on a very small subset of the 1KG reference dataset and should not be used to call ancestry inference on a real profile.*

Description

The object is a `list`.

Usage

```
data(demoPCA1KG)
```

Format

The list containing the PCA results for a small subset of the reference 1KG dataset. The list contains 2 entries:

- pruned a vector of SNV identifiers specifying selected SNVs for the PCA analysis.
- pca.unrel a snpgdsPCAClass object containing the eigenvalues as generated by snpgdsPCA function.

Details

This object can be used to test the computePCAMultiSynthetic function.

Value

The list containing the PCA results for a small subset of the reference 1KG dataset. The list contains 2 entries:

- pruned a vector of SNV identifiers specifying selected SNVs for the PCA analysis.
- pca.unrel a snpgdsPCAClass object containing the eigenvalues as generated by snpgdsPCA function.

Examples

```
## Required library
library(gdsfmt)

## Loading demo PCA on subset of 1KG reference dataset
data(demoPCA1KG)

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")

# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

samplesRM <- c("HG00246", "HG00325", "HG00611", "HG01173", "HG02165",
    "HG01112", "HG01615", "HG01968", "HG02658", "HG01850", "HG02013",
    "HG02465", "HG02974", "HG03814", "HG03445", "HG03689", "HG03789",
    "NA12751", "NA19107", "NA18548", "NA19075", "NA19475", "NA19712",
    "NA19731", "NA20528", "NA20908")
names(samplesRM) <- c("GBR", "FIN", "CHS","PUR", "CDX", "CLM", "IBS",
    "PEL", "PJL", "KHV", "ACB", "GWD", "ESN", "BEB", "MSL", "STU", "ITU",
    "CEU", "YRI", "CHB", "JPT", "LWK", "ASW", "MXL", "TSI", "GIH")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))
```

```
## Projects synthetic profiles on demo 1KG PCA
results <- computePCAMultiSynthetic(gdsProfile=gdsProfile,
    listPCA=demoPCA1KG, sampleRef=samplesRM, studyIDSyn=studyID,
    verbose=FALSE)

## The eigenvectors for the synthetic profiles
head(results$eigenvector)

## Close Profile GDS file (important)
closefn.gds(gdsProfile)
```

`demoPCASyntheticProfiles` *The PCA result of demo synthetic profiles projected on the demo subset 1KG reference PCA.*

Description

The object is a `list`.

Usage

```
data(demoPCASyntheticProfiles)
```

Format

The `list` containing the PCA result of demo synthetic profiles projected on the demo subset 1KG reference PCA. The `list` contains 3 entries:

- sample.id a `character` string representing the unique identifier of the synthetic profiles.
- eigenvector.ref a `matrix` of `numeric` containing the eigenvectors for the reference profiles.
- eigenvector a `matrix` of `numeric` containing the eigenvectors for the current synthetic profiles projected on the demo PCA 1KG reference profiles.

Details

This object can be used to test the `computeKNNRefSynthetic` function.

Value

The `list` containing the PCA result of demo synthetic profiles projected on the demo subset 1KG reference PCA. The `list` contains 3 entries:

- sample.id a `character` string representing the unique identifier of the synthetic profiles.
- eigenvector.ref a `matrix` of `numeric` containing the eigenvectors for the reference profiles.
- eigenvector a `matrix` of `numeric` containing the eigenvectors for the current synthetic profiles projected on the demo PCA 1KG reference profiles.

See Also

- computeKNNRefSynthetic for running a k-nearest neighbors analysis on a subset of the synthetic data set.

Examples

```
## Required library
library(gdsfmt)

## Load the demo PCA on the synthetic profiles projected on the
## demo 1KG reference PCA
data(demoPCASyntheticProfiles)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

## Path to the demo Profile GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")

## Open the Profile GDS file
gdsProfile <- snpgdsOpen(file.path(dataDir, "ex1.gds"))

# The name of the synthetic study
studyID <- "MYDATA.Synthetic"

## Projects synthetic profiles on 1KG PCA
results <- computeKNNRefSynthetic(gdsProfile=gdsProfile,
    listEigenvector=demoPCASyntheticProfiles,
    listCatPop=c("EAS", "EUR", "AFR", "AMR", "SAS"), studyIDSyn=studyID,
    spRef=demoKnownSuperPop1KG)

## The inferred ancestry for the synthetic profiles for different values
## of D and K
head(results$matKNN)

## Close Profile GDS file (important)
closefn.gds(gdsProfile)
```

---

demoPedigreeEx1 *The pedigree information about a demo profile called 'ex1'.*

Description

The object is a data.frame.

Usage

```
data(demoPedigreeEx1)
```

Format

The `data.frame` containing the information about a demo profile called 'ex1'. the `data.frame` has 5 columns:

- Name.ID a `character` string representing the unique identifier of the profile.
- Case.ID a `character` string representing the unique identifier of the case associated to the profile.
- Sample.Type a `character` string describing the type of profile.
- Diagnosis a `character` string describing the diagnosis of the profile.
- Source a `character` string describing the source of the profile.

Details

This object can be used to test the `runExomeAncestry` function.

Value

The `data.frame` containing the information about a demo profile called 'ex1'. the `data.frame` has 5 columns:

- Name.ID a `character` string representing the unique identifier of the profile.
- Case.ID a `character` string representing the unique identifier of the case associated to the profile.
- Sample.Type a `character` string describing the type of profile.
- Diagnosis a `character` string describing the diagnosis of the profile.
- Source a `character` string describing the source of the profile.

See Also

- `runExomeAncestry` for running runs most steps leading to the ancestry inference call on a specific exome profile.

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

#####################################################################
## Load the information about the profile
#####################################################################
data(demoPedigreeEx1)
head(demoPedigreeEx1)

#####################################################################
```

```
## The 1KG GDS file and the 1KG SNV Annotation GDS file
## need to be located in the same directory
## Note that the 1KG GDS file used for this example is a
## simplified version and CANNOT be used for any real analysis
######################################################################
path1KG <- file.path(dataDir, "tests")

fileReferenceGDS  <- file.path(path1KG, "ex1_good_small_1KG.gds")
fileAnnotGDS <- file.path(path1KG, "ex1_good_small_1KG_Annot.gds")

######################################################################
## The Sample SNP pileup files (one per sample) need
## to be located in the same directory.
######################################################################
pathGeno <- file.path(dataDir, "example", "snpPileup")

######################################################################
## The path where the Profile GDS Files (one per sample)
## will be created need to be specified.
######################################################################
pathProfileGDS <- file.path(tempdir(), "out.tmp")

pathOut <- file.path(tempdir(), "res.out")

######################################################################
## A data frame containing general information about the study
## is also required. The data frame must have
## those 3 columns: "studyID", "study.desc", "study.platform"
######################################################################
studyDF <- data.frame(study.id="MYDATA",
                      study.desc="Description",
                      study.platform="PLATFORM",
                      stringsAsFactors=FALSE)

######################################################################
## Fix seed to ensure reproducible results
######################################################################
set.seed(2043)

gds1KG <- snpgdsOpen(fileReferenceGDS)
dataRef <- select1KGPop(gds1KG, nbProfiles=2L)
closefn.gds(gds1KG)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]


        runExomeAncestry(pedStudy=demoPedigreeEx1, studyDF=studyDF,
```

```
            pathProfileGDS=pathProfileGDS,
            pathGeno=pathGeno, pathOut=pathOut,
            fileReferenceGDS=fileReferenceGDS,
            fileReferenceAnnotGDS=fileAnnotGDS,
            chrInfo=chrInfo, syntheticRefDF=dataRef,
            genoSource="snp-pileup")

        unlink(pathProfileGDS, recursive=TRUE, force=TRUE)
        unlink(pathOut, recursive=TRUE, force=TRUE)

}
```

`estimateAllelicFraction` *Estimate the allelic fraction of the pruned SNVs for a specific profile*

Description

The function estimates the allelic fraction of the SNVs for a specific profile and add the information to the associated Profile GDS file. The allelic fraction estimation method is adapted to the type of study (DNA or RNA).

Usage

```
estimateAllelicFraction(
  gdsReference,
  gdsProfile,
  currentProfile,
  studyID,
  chrInfo,
  studyType = c("DNA", "RNA"),
  minCov = 10L,
  minProb = 0.999,
  eProb = 0.001,
  cutOffLOH = -5,
  cutOffHomoScore = -3,
  wAR = 9,
  cutOffAR = 3,
  gdsRefAnnot = NULL,
  blockID = NULL,
  verbose = FALSE
)
```

Arguments

`gdsReference`	an object of class `gds.class` (a GDS file), the opened Reference GDS file.
`gdsProfile`	an object of class `gds.class` (a GDS file), the opened Profile GDS file.

currentProfile	a character string corresponding to the sample identifier as used in pruningSample function.
studyID	a character string corresponding to the name of the study as used in pruningSample function.
chrInfo	a vector of integer values representing the length of the chromosomes. See 'details' section.
studyType	a character string representing the type of study. The possible choices are: "DNA" and "RNA". The type of study affects the way the estimation of the allelic fraction is done. Default: "DNA".
minCov	a single positive integer representing the minimum required coverage. Default: 10L.
minProb	a single numeric between 0 and 1 representing the probability that the calculated genotype call is correct. Default: 0.999.
eProb	a single numeric between 0 and 1 representing the probability of sequencing error. Default: 0.001.
cutOffLOH	a single numeric representing the cutoff, in log, for the homozygote score to assign a region as LOH. Default: -5.
cutOffHomoScore	a single numeric representing the cutoff, in log, that the SNVs in a block are called homozygote by error. Default: -3.
wAR	a single positive integer representing the size-1 of the window used to compute an empty box. Default: 9.
cutOffAR	a single numeric representing the cutoff, in log score, that the SNVs in a gene are allelic fraction different 0.5 Default: 3.
gdsRefAnnot	an object of class gds.class (a GDS file), the opened Reference SNV Annotation GDS file. This parameter is RNA specific. Default: NULL.
blockID	a character string corresponding to the block identifier in gdsRefAnnot. **This parameter is RNA specific.** Default: NULL
verbose	a logicial indicating if the function should print message when running. Default: FALSE.

Details

The chrInfo parameter contains the length of the chromosomes. The length of the chromosomes can be obtain through the seqlengths library.

As example, for hg38 genome:

```
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {
   chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]
}
```

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(gdsfmt)

## Path to the demo 1KG GDS file located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## Profile GDS file for one profile
fileProfile <- file.path(tempdir(), "ex1.gds")

## Copy the Profile GDS file demo that has been pruned and annotated
## into current directory
file.copy(file.path(dataDir, "ex1_demo_with_pruning_and_1KG_annot.gds"),
                            fileProfile)

## Open the reference GDS file (demo version)
gds1KG <- snpgdsOpen(fileGDS)

## Profile GDS file for one profile
profileGDS <- openfn.gds(fileProfile, readonly=FALSE)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

    ## Estimate the allelic fraction of the pruned SNVs
    estimateAllelicFraction(gdsReference=gds1KG, gdsProfile=profileGDS,
        currentProfile="ex1", studyID="MYDATA", chrInfo=chrInfo,
        studyType="DNA", minCov=10L, minProb=0.999, eProb=0.001,
        cutOffLOH=-5, cutOffHomoScore=-3, wAR=9, cutOffAR=3,
        gdsRefAnnot=NULL, blockID=NULL)

    ## The allelic fraction is saved in the 'lap' node of Profile GDS file
    ## The 'lap' entry should be present
    profileGDS

    ## Close both GDS files (important)
    closefn.gds(profileGDS)
    closefn.gds(gds1KG)

    ## Remove Profile GDS file (created for demo purpose)
    unlink(fileProfile, force=TRUE)
```

```
}
```

`generateGDS1KG` *Generate the GDS file that will contain the information from Reference data set (reference data set)*

Description

This function generates the GDS file that will contain the information from Reference. The function also add the samples information, the SNP information and the genotyping information into the GDS file.

Usage

```
generateGDS1KG(
  pathGeno = file.path("data", "sampleGeno"),
  filePedRDS,
  fileSNVIndex,
  fileSNVSelected,
  fileNameGDS,
  listSamples = NULL,
  verbose = FALSE
)
```

Arguments

`pathGeno`	a `character` string representing the path where the 1K genotyping files for each sample are located. The name of the genotyping files must correspond to the individual identification (Individual.ID) in the pedigree file. Default: `"./data/sampleGeno"`.
`filePedRDS`	a `character` string representing the path and file name of the RDS file that contains the pedigree information. The file must exist. The file must be a RDS file.
`fileSNVIndex`	a `character` string representing the path and file name of the RDS file that contains the indexes of the retained SNPs. The file must exist. The file must be a RDS file.
`fileSNVSelected`	a `character` string representing the path and file name of the RDS file that contains the filtered SNP information. The file must exist. The file must be a RDS file.
`fileNameGDS`	a `character` string representing the path and file name of the GDS file that will be created. The GDS file will contain the SNP information, the genotyping information and the pedigree information from 1000 Genomes. The extension of the file must be '.gds'.

listSamples	a vector of character string corresponding to samples (must be the sample.ids) that will be retained and added to the GDS file. When NULL, all the samples are retained. Default: NULL.
verbose	a logical indicating if the funciton must print messages when running. Default: FALSE.

Details

More information about GDS file format can be found at the Bioconductor gdsfmt website: https://bioconductor.org/packages/

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Path to the CSV genoytype files
pathGeno <- file.path(dataDir, "demoProfileGenotypes")

## The RDS file containing the pedigree information
pedigreeFile <- file.path(dataDir, "PedigreeReferenceDemo.rds")

## The RDS file containing the indexes of the retained SNPs
snpIndexFile <- file.path(dataDir, "listSNPIndexes_Demo.rds")

## The RDS file containing the filtered SNP information
filterSNVFile <- file.path(dataDir, "mapSNVSelected_Demo.rds")

## Temporary Reference GDS file
tempRefGDS <- file.path(tempdir(), "1KG_TEMP.gds")

## Create a temporary Reference GDS file
generateGDS1KG(pathGeno=pathGeno, filePedRDS=pedigreeFile,
            fileSNVIndex=snpIndexFile, fileSNVSelected=filterSNVFile,
            fileNameGDS=tempRefGDS, listSamples=NULL)

## Remove temporary files
unlink(tempRefGDS, force=TRUE)
```

generateGDS1KGgenotypeFromSNPPileup
Append the genotype information from a profile into the associated Profile GDS File

Description

This function append the genotype information from a specific profile into the Profile GDS file. The genotype information is extracted from a SNV file as generated by SNP-pileup or other tools.

Usage

```
generateGDS1KGgenotypeFromSNPPileup(
  pathGeno,
  listSamples,
  listPos,
  offset,
  minCov = 10,
  minProb = 0.999,
  seqError = 0.001,
  dfPedProfile,
  batch,
  studyDF,
  pathProfileGDS,
  genoSource,
  verbose
)
```

Arguments

pathGeno	a character string representing the path to the directory containing the VCF output of SNP-pileup for each sample. The SNP-pileup files must be compressed (gz files) and have the name identifiers of the samples. A sample with "Name.ID" identifier would have an associated file called if genoSource is "VCF", then "Name.ID.vcf.gz", if genoSource is "generic", then "Name.ID.generic.txt.gz" if genoSource is "snp-pileup", then "Name.ID.txt.gz".
listSamples	a vector of character string corresponding to the sample identifiers that will have a Profile GDS file created. The sample identifiers must be present in the "Name.ID" column of the data.frame passed to the dfPedProfile parameter.
listPos	a data.frame containing 2 columns. The first column, called "snp.chromosome" contains the name of the chromosome where the SNV is located. The second column, called "snp.position" contains the position of the SNV on the chromosome.
offset	a integer to adjust if the genome start at 0 or 1.
minCov	a single positive integer representing the minimum coverage needed to keep the SNVs in the analysis. Default: 10.

minProb	a single positive numeric between 0 and 1 representing the probability that the base change at the SNV position is not an error. Default: 0.999.
seqError	a single positive numeric between 0 and 1 representing the sequencing error rate. Default: 0.001.
dfPedProfile	a data.frame with the information about the sample(s). Those are mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis" and "Source". All columns must be in character strings format. The data.frame must contain the information for all the samples passed in the listSamples parameter.
batch	a single positive integer representing the current identifier for the batch. Beware, this field is not stored anymore.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings.
pathProfileGDS	a character string representing the path to the directory where the GDS Sample files will be created.
genoSource	a character string with two possible values: 'snp-pileup', 'generic' or 'VCF'. It specifies if the genotype files are generated by snp-pileup (Facets) or are a generic format CSV file with at least those columns: 'Chromosome', 'Position', 'Ref', 'Alt', 'Count', 'File1R' and 'File1A'. The 'Count' is the depth at the specified position; 'FileR' is the depth of the reference allele and 'File1A' is the depth of the specific alternative allele. Finally the file can be a VCF file with at least those genotype fields: GT, AD, DP.
verbose	a logical indicating if the function must print messages when running.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Current directory
dataDir <- file.path(tempdir())

## Copy required file into current directory
file.copy(from=file.path(system.file("extdata/tests", package="RAIDS"),
                    "ex1.txt.gz"), to=dataDir)

## The data.frame containing the information about the study
## The 3 mandatory columns: "study.id", "study.desc", "study.platform"
## The entries should be strings, not factors (stringsAsFactors=FALSE)
studyDF <- data.frame(study.id = "MYDATA",
                        study.desc = "Description",
                        study.platform = "PLATFORM",
                        stringsAsFactors = FALSE)
```

```
## The data.frame containing the information about the samples
## The entries should be strings, not factors (stringsAsFactors=FALSE)
samplePED <- data.frame(Name.ID=c("ex1", "ex2"),
                    Case.ID=c("Patient_h11", "Patient_h12"),
                    Diagnosis=rep("Cancer", 2),
                    Sample.Type=rep("Primary Tumor", 2),
                    Source=rep("Databank B", 2), stringsAsFactors=FALSE)
rownames(samplePED) <- samplePED$Name.ID

## List of SNV positions
listPositions <- data.frame(snp.chromosome=c(rep(1, 10)),
        snp.position=c(3467333, 3467428, 3469375, 3469387, 3469502, 3469527,
        3469737, 3471497, 3471565, 3471618))

## Append genotype information to the Profile GDS file
result <- RAIDS:::generateGDS1KGgenotypeFromSNPPileup(pathGeno=dataDir,
            listSamples=c("ex1"), listPos=listPositions,
            offset=-1, minCov=10, minProb=0.999, seqError=0.001,
            dfPedProfile=samplePED, batch=1, studyDF=studyDF,
            pathProfileGDS=dataDir, genoSource="snp-pileup",
            verbose=FALSE)

## The function returns OL when successful
result

## The Profile GDS file 'ex1.gds' has been created in the
## specified directory
list.files(dataDir)

## Unlink Profile GDS file (created for demo purpose)
unlink(file.path(dataDir, "ex1.gds"))
unlink(file.path(dataDir, "ex1.txt.gz"))
```

`generateGDSgenotype` *Add information related to profile genotypes into a Population Reference GDS file*

Description

This function adds the genotype fields with the associated information into the Population Reference GDS file for the selected profiles.

Usage

```
generateGDSgenotype(gds, pathGeno, fileSNPsRDS, listSamples, verbose)
```

Arguments

gds	an object of class gds.class (a GDS file), the opened Population Reference GDS file.
pathGeno	a character string representing the path where the reference genotyping files for each sample are located. The name of the genotyping files must correspond to the individual identification (Individual.ID) in the pedigree file.
fileSNPsRDS	a character string representing the path and file name of the RDS file that contains the indexes of the retained SNPs. The file must exist. The file must be a RDS file.
listSamples	a vector of character string corresponding to profiles (must be the profile identifiers) that will be retained and added to the Reference GDS file.
verbose	a logical indicating if the function must print messages when running.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Path where the demo genotype CSV files are located
pathGeno <- file.path(dataDir, "demoProfileGenotypes")

## The RDS file containing the pedigree information
pedigreeFile <- file.path(dataDir, "PedigreeReferenceDemo.rds")

## The RDS file containing the indexes of the retained SNPs
snpIndexFile <- file.path(dataDir, "listSNPIndexes_Demo.rds")

## The RDS file containing the filtered SNP information
filterSNVFile <- file.path(dataDir, "mapSNVSelected_Demo.rds")

## Temporary Reference GDS file in temporary directory
tempRefGDS <- file.path(tempdir(), "Ref_TEMP01.gds")

## Create temporary Reference GDS file
newGDS <- createfn.gds(tempRefGDS)
put.attr.gdsn(newGDS$root, "FileFormat", "SNP_ARRAY")

## Read the pedigree file
ped1KG <- readRDS(pedigreeFile)
```

```
## Add information about samples to the Reference GDS file
listSampleGDS <- RAIDS:::generateGDSRefSample(gdsReference=newGDS,
                dfPedReference=ped1KG, listSamples=NULL)

## Add SNV information to the Reference GDS
RAIDS:::generateGDSSNPinfo(gdsReference=newGDS, fileFreq=filterSNVFile,
                verbose=FALSE)

## Add genotype information to the Reference GDS
RAIDS:::generateGDSgenotype(gds=newGDS, pathGeno=pathGeno,
        fileSNPsRDS=snpIndexFile, listSamples=listSampleGDS, verbose=FALSE)

## Close file
closefn.gds(newGDS)

## Remove temporary files
unlink(tempRefGDS, force=TRUE)
```

`generateGDSRefSample` *Initialization of the section related to the profile information in the GDS file*

Description

This function initializes the section related to the profile information in the GDS file. The information is extracted from the `data.frame` passed to the function. The nodes "sample.id" and "sample.annot" are created in the GDS file.

Usage

```
generateGDSRefSample(gdsReference, dfPedReference, listSamples = NULL)
```

Arguments

`gdsReference`	an object of class gds.class (a GDS file), the opened GDS file.
`dfPedReference`	a `data.frame` containing the information related to the samples. It must have those columns: "sample.id", "Name.ID", "sex", "pop.group", "superPop" and "batch". All columns, except "sex" and batch", are `character` strings. The "batch" and "sex" columns are `integer`. The unique identifier of this `data.frame` is the "Name.ID" column. The row names of the `data.frame` must correspond to the identifiers present in the "Name.ID" column.
`listSamples`	a `vector` of `character` string representing the identifiers of the selected profiles. If `NULL`, all profiles are selected. Default: `NULL`.

Value

a `vector` of `character` string with the identifiers of the profiles saved in the GDS file.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Temporary GDS file in current directory
gdsFilePath <- file.path(tempdir(), "GDS_TEMP_10.gds")

## Create and open the GDS file
tmpGDS  <- createfn.gds(filename=gdsFilePath)

## Create "sample.annot" node (the node must be present)
pedInformation <- data.frame(sample.id=c("sample_01", "sample_02"),
        Name.ID=c("sample_01", "sample_02"),
        sex=c(1,1),  # 1:Male  2: Female
        pop.group=c("ACB", "ACB"),
        superPop=c("AFR", "AFR"),
        batch=c(1, 1),
        stringsAsFactors=FALSE)

## The row names must be the sample identifiers
rownames(pedInformation) <- pedInformation$Name.ID

## Add information about 2 samples to the GDS file
RAIDS:::generateGDSRefSample(gdsReference=tmpGDS,
        dfPedReference=pedInformation, listSamples=NULL)

## Read sample identifier list
read.gdsn(index.gdsn(node=tmpGDS, path="sample.id"))

## Read sample information from GDS file
read.gdsn(index.gdsn(node=tmpGDS, path="sample.annot"))

## Close GDS file
closefn.gds(gdsfile=tmpGDS)

## Delete the temporary GDS file
unlink(x=gdsFilePath, force=TRUE)
```

---

generateGDSSNPinfo *Add information related to SNVs into a Population Reference GDS file*

Description

The function adds the SNV information into a Population Reference GDS file.

Usage

```
generateGDSSNPinfo(gdsReference, fileFreq, verbose)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened Reference GDS file.
fileFreq	a character string representing the path and file name of the RDS file with the filtered SNP information.
verbose	a logical indicating if messages should be printed to show how the different steps in the function.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required package
library(gdsfmt)

## The RDS file containing the filtered SNP information
dataDir <- system.file("extdata", package="RAIDS")
fileFilerterSNVs <- file.path(dataDir, "mapSNVSelected_Demo.rds")

## Temporary Reference GDS file in temporary directory
file1KG <- file.path(tempdir(), "1KG_TEMP_002.gds")
filenewGDS <- createfn.gds(file1KG)

## Add SNV information to Reference GDS
RAIDS:::generateGDSSNPinfo(gdsReference=filenewGDS,
            fileFreq=fileFilerterSNVs, verbose=TRUE)

## Close GDS file (important)
closefn.gds(filenewGDS)

## Remove temporary 1KG_TEMP_002.gds file
unlink(file1KG, force=TRUE)
```

`generateGeneBlock` *Generate two indexes based on gene annotation for gdsAnnot1KG block*

Description

Generate two indexes based on gene annotation for gdsAnnot1KG block

Usage

```
generateGeneBlock(gdsReference, winSize = 10000, ensDb)
```

Arguments

`gdsReference`	an object of class gds.class (a GDS file), the opened 1KG GDS file (reference).
`winSize`	a single positive `integer` representing the size of the window to use to group the SNVs when the SNVs are in a non-coding region. Default: `10000`.
`ensDb`	An object of class `EnsDb` with the Ensembl genome annotation. By default, the `EnsDb.Hsapiens.v86` class has been used.

Value

a `data.frame` with those columns:

- chr a single `integer` representing the SNV chromosome.
- pos a single `integer` representing the SNV position.
- snp.allele a `character` string representing the reference allele and alternative allele for each of the SNV
- Exon a `character` with the ensembl GeneId(s) if the SNV is in one exon. If more than one GeneId they are separted by ':'
- GName a `character` with the ensembl GeneId(s) if the SNV is in the gene. If more than one GeneId they are separted by ':'
- Gene A single `integer` specific to the SNVs that share at least one genes
- GeneS A single `integer` specific to the SNVs that share a unique combination of genes

"chr", "pos", "snp.allele", "Exon", "GName", "Gene", "GeneS" Example for GName and the two indexes "Gene", "GeneS" GName Gene GeneS 470 ENSG00000230021 17 3820 471 ENSG00000230021 17 3820 472 ENSG00000230021:ENSG00000228794 17 3825 473 ENSG00000230021:ENSG00000228794 17 3825 481 ENSG00000230021:ENSG00000228794:ENSG00000225880 17 3826 482 ENSG00000230021:ENSG0000022 17 3826 483 ENSG00000230021:ENSG00000228794:ENSG00000225880 17 3826 492 ENSG00000230021:ENSG0000022 17 3825 493 ENSG00000230021:ENSG00000228794 17 3825

Author(s)

Pascal Belleau, Astrid Deschênes and Alex Krasnitz

Examples

```
## Required library
library(SNPRelate)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Required library
if (requireNamespace("EnsDb.Hsapiens.v86", quietly=TRUE)) {

    ## Making a "short cut" on the ensDb object
    edb <- EnsDb.Hsapiens.v86::EnsDb.Hsapiens.v86

    path1KG <- file.path(dataDir, "tests")

    ## Reference GDS file
    fileReferenceGDS  <- file.path(path1KG, "ex1_good_small_1KG.gds")


        ## Open the reference GDS file (demo version)
        gds1KG <- snpgdsOpen(fileReferenceGDS)

        ## The function returns a data.frame containing
        ## gene block information
        matGeneBlock <- RAIDS:::generateGeneBlock(gdsReference=gds1KG,
                            ensDb=edb)
        print(head(matGeneBlock[grep("ENSG00000157152",
                            matGeneBlock$GName),]))

        closefn.gds(gds1KG)

}
```

`generateMapSnvSel` *Generate the filter SNP information file in RDS format*

Description

The function applies a cut-off filter to the SNP information file to retain only the SNP that have a frequency superior or equal to the specified cut-off in at least one super population. The information about the retained SNPs is saved in a RDS format file. A RDS file containing the indexes of the retained SNP is also created.

Usage

```
generateMapSnvSel(cutOff = 0.01, fileSNV, fileSNPsRDS, fileFREQ)
```

Arguments

cutOff	a single numeric value, the cut-off for the frequency in at least one super population. Default: 0.01.
fileSNV	a character string representing the path and file name of the bulk SNP information file from Reference. The file must be in text format. The file must exist.
fileSNPsRDS	a character string representing the path and file name of the RDS file that will contain the indexes of the retained SNPs. The file extension must be '.rds'.
fileFREQ	a character string representing the path and file name of the RDS file that will contain the filtered SNP information. The file extension must be '.rds'.

Details

The filtered SNP information RDS file (parameter fileFREQ), contains a data.frame with those columns:

- CHROM a character string representing the chromosome where the SNV is located.
- POS a character string representing the SNV position on the chromosome.
- REF a character string representing the reference DNA base for the SNV.
- ALT a character string representing the alternative DNA base for the SNV.\
- EAS_AF a character string representing the allele frequency of the EAS super population.
- AFR_AF a character string representing the allele frequency of the AFR super population.
- AMR_AF a character string representing the allele frequency of the AMR super population.
- SAS_AF a character string representing the allele frequency of the SAS super population.

Value

The integer 0 when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Demo SNV information file used as input
snvFile <- file.path(dataDir, "matFreqSNV_Demo.txt.bz2")

## Temporary output files
## The first file contains the indexes of the retained SNPs
## The second file contains the filtered SNP information
snpIndexFile <- file.path(tempdir(), "listSNP_TEMP.rds")
filterSNVFile <- file.path(tempdir(), "mapSNVSel_TEMP.rds")

## Create a data.frame containing the information of the retained
```

```
## samples (samples with existing genotyping files)
generateMapSnvSel(cutOff=0.01, fileSNV=snvFile,
        fileSNPsRDS=snpIndexFile, fileFREQ=filterSNVFile)

## Remove temporary files
unlink(snpIndexFile, force=TRUE)
unlink(filterSNVFile, force=TRUE)
```

`generatePhase1KG2GDS` *Adding the phase information into the Reference GDS file*

Description

The function is adding the phase information into the Reference Phase GDS file. The phase information is extracted from a Reference GDS file and is added into a Reference Phase GDS file. An entry called 'phase' is added to the Reference Phase GDS file.

Usage

```
generatePhase1KG2GDS(
  gdsReference,
  gdsReferencePhase,
  pathGeno,
  fileSNPsRDS,
  verbose = FALSE
)
```

Arguments

`gdsReference`	an object of class gds.class (GDS file), an opened Reference GDS file.
`gdsReferencePhase`	an object of class gds.class (GDS file), an opened Reference Phase GDS file.
`pathGeno`	a `character` string representing the path where the 1K genotyping files for each sample are located. The name of the genotyping files must correspond to the individual identification (Individual.ID) in the pedigree file. Default: `"./data/sampleGeno"`.
`fileSNPsRDS`	a `character` string representing the path and file name of the RDS file that contains the indexes of the retained SNPs. The file must exist. The file must be a RDS file.
`verbose`	a `logicial` indicating if the function should print messages when running. Default: `FALSE`.

Value

The function returns `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required package
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Path where the demo genotype CSV files are located
pathGeno <- file.path(dataDir, "demoProfileGenotypes")

## The RDS file containing the pedigree information
pedigreeFile <- file.path(dataDir, "PedigreeReferenceDemo.rds")

## The RDS file containing the indexes of the retained SNPs
snpIndexFile <- file.path(dataDir, "listSNPIndexes_Demo.rds")

## The RDS file containing the filtered SNP information
filterSNVFile <- file.path(dataDir, "mapSNVSelected_Demo.rds")

## Temporary Reference GDS file containing reference information
fileReferenceGDS <- file.path(tempdir(), "1KG_TEMP_02.gds")

## Create a temporary Reference GDS file containing information from 1KG
generateGDS1KG(pathGeno=pathGeno, filePedRDS=pedigreeFile,
        fileSNVIndex=snpIndexFile, fileSNVSelected=filterSNVFile,
        fileNameGDS=fileReferenceGDS, listSamples=NULL)

## Temporary Phase GDS file that will contain the 1KG Phase information
fileRefPhaseGDS <- file.path(tempdir(), "1KG_TEMP_Phase_02.gds")

## Create Reference Phase GDS file
gdsPhase <- createfn.gds(fileRefPhaseGDS)

## Open Reference GDS file
gdsRef <- openfn.gds(fileReferenceGDS)

## Fill temporary Reference Phase GDS file
if (FALSE) {
    generatePhase1KG2GDS(gdsReference=gdsRef,
            gdsReferencePhase=gdsPhase,
            pathGeno=pathGeno, fileSNPsRDS=filterSNVFile,
            verbose=FALSE)
}

## Close Reference Phase information file
closefn.gds(gdsPhase)

## Close Reference information file
```

```
closefn.gds(gdsRef)

## Remove temporary files
unlink(fileReferenceGDS, force=TRUE)
unlink(fileRefPhaseGDS, force=TRUE)
```

`getBlockIDs` *Extract the block identifiers for a list of SNVs*

Description

The function uses the GDS Reference Annotation file to extract the unique block identifiers for a list of SNVs. The block type that is going to be used to extract the information has to be provided by the user.

Usage

```
getBlockIDs(gdsRefAnnot, snpIndex, blockTypeID)
```

Arguments

`gdsRefAnnot`	an object of class `gds.class` (a GDS file), the opened Reference SNV Annotation GDS file.
`snpIndex`	a `vectcor` of `integer` representing the indexes of the SNVs of interest.
`blockTypeID`	a `character` string corresponding to the block type used to extract the block identifiers. The block type must be present in the GDS Reference Annotation file.

Value

a `vector` of `numeric` corresponding to the block identifiers for the SNVs of interest.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
# Required library
library(gdsfmt)

## Path to the demo 1KG Annotation GDS file located in this package
dataDir <- system.file("extdata", package="RAIDS")

path1KG <- file.path(dataDir, "tests")
fileAnnotGDS <- file.path(path1KG, "ex1_good_small_1KG_Annot.gds")
```

```
gdsRefAnnotation <- openfn.gds(fileAnnotGDS)

## The indexes for the SNVs of interest
snpIndex <- c(1,3,5,6,9)

## Extract the block identifiers for the SNVs represented by their indexes
## for the block created using the genes from Hsapiens Ensembl v86
RAIDS:::getBlockIDs(gdsRefAnnot=gdsRefAnnotation, snpIndex=snpIndex,
                        blockTypeID="GeneS.Ensembl.Hsapiens.v86")

closefn.gds(gdsRefAnnotation)
```

`getRef1KGPop` *Extract the specified column from the 1KG GDS 'sample.ref' node for the reference profiles (real ancestry assignation)*

Description

The function extract the specified column for the 'sample.ref' node present in the Reference GDS file. The column must be present in the `data.frame` saved in the 'sample.ref' node. Only the information for the reference profiles is returned. The values represent the known ancestry assignation.

Usage

```
getRef1KGPop(gdsReference, popName = "superPop")
```

Arguments

`gdsReference`	an object of class gds.class (a GDS file), the opened Reference GDS file.
`popName`	a `character` string representing the name of the column that will be fetched in the `data.frame` present in the Reference GDS "sample.ref" node. The column must be present in the `data.frame`. Default: `"superPop"`.

Value

`vector` of `character` strings representing the content of the extracted column for the 1KG GDS 'sample.ref' node. The values represent the known ancestry assignation. The profile identifiers are used as names for the `vector`.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Open existing demo 1K GDS file with "sample.ref" node
nameFileGDS <- file.path(dataDir, "PopulationReferenceDemo.gds")
fileGDS <- snpgdsOpen(nameFileGDS)

## Extract super population information for the 1KG profiles
getRef1KGPop(gdsReference=fileGDS, popName="superPop")

## Close 1K GDS file
closefn.gds(fileGDS)
```

`getTableSNV` *Extract the genotype information for a SNV dataset using the Profile GDS file and the Reference GDS file*

Description

The function generates a `data.frame` containing the genotype information from a initial list of SNVs associated to a specific profile. The function uses the information present in the Reference GDS file and the Profile GDS file.

Usage

```
getTableSNV(
  gdsReference,
  gdsSample,
  currentProfile,
  studyID,
  minCov = 10,
  minProb = 0.999,
  eProb = 0.001,
  verbose
)
```

Arguments

`gdsReference`	an object of class `gds.class` (a GDS file), the opened Reference GDS file.
`gdsSample`	an object of class `gds.class` (a GDS file), the opened Profile GDS file.
`currentProfile`	a `character` string corresponding to the sample identifier used in `pruningSample` function.

studyID	a character string corresponding to the study identifier used in pruningSample function.
minCov	a single positive integer representing the minimum required coverage. Default: 10L.
minProb	a single numeric between 0 and 1 representing the probability that the calculated genotype call is correct. Default: 0.999.
eProb	a single numeric between 0 and 1 representing the probability of sequencing error. Default: 0.001.
verbose	a logicial indicating if messages should be printed when the function is running.

Value

a data.frame containing:

- cnt.tot a single integer representing the total coverage for the SNV.
- cnt.ref a single integer representing the coverage for the reference allele.
- cnt.alt a single integer representing the coverage for the alternative allele.
- snpPos a single integer representing the SNV position.
- snp.chr a single integer representing the SNV chromosome.
- normal.geno a single numeric indicating the genotype of the SNV. The possibles are: 0 (wild-type homozygote), 1 (heterozygote), 2 (altenative homozygote), 3 indicating that the normal genotype is unknown.
- pruned a logical
- snp.index a vector of integer representing the position of the SNVs in the Reference GDS file.
- keep a logical
- hetero a logical
- homo a logical

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## Temporary Profile GDS file for one profile in temporary directory
fileProfile <- file.path(tempdir(), "ex1.gds")
```

```
## Copy the Profile GDS file demo that has been pruned and annotated
file.copy(file.path(dataDir, "ex1_demo_with_pruning_and_1KG_annot.gds"),
                 fileProfile)

## Open the reference GDS file (demo version)
gds1KG <- snpgdsOpen(fileGDS)

## Open Profile GDS file for one profile
profileGDS <- openfn.gds(fileProfile)

## The function returns a data frame containing the SNVs information
result <- RAIDS:::getTableSNV(gdsReference=gds1KG, gdsSample=profileGDS,
        currentProfile="ex1", studyID="MYDATA", minCov=10L, minProb=0.999,
        eProb=0.001, verbose=FALSE)
head(result)

## Close both GDS files (important)
closefn.gds(profileGDS)
closefn.gds(gds1KG)

## Remove Profile GDS file (created for demo purpose)
unlink(fileProfile, force=TRUE)
```

---

groupChr1KGSNV *Merge the genotyping files per chromosome into one file*

Description

This function merge all the genotyping files associated to one specific sample into one file. That merged VCF file will be saved in a specified directory and will have the name of the sample. It will also be compressed (bzip). The function will merge the files for all samples present in the input directory.

Usage

```
groupChr1KGSNV(pathGenoChr, pathOut)
```

Arguments

pathGenoChr	a character string representing the path where the genotyping files for each sample and chromosome are located. The path must contains sub-directories (one per chromosome) and the genotyping files must be present in those sub-directories. The path must exists.
pathOut	a character string representing the path where the merged genotyping files for each sample will be created. The path must exists.

Value

The integer `0L` when successful or `FALSE` if not.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo vcf files in this package
dataDir <- system.file("extdata", package="RAIDS")
pathGenoTar <- file.path(dataDir, "demoGenoChr", "demoGenoChr.tar")

## Path where the chromosomes files will be located
pathGeno <- file.path(tempdir(), "tempGeno")
dir.create(pathGeno, showWarnings=FALSE)

## Untar the file that contains the VCF files for 3 samples split by
## chromosome (one directory per chromosome)
untar(tarfile=pathGenoTar, exdir=pathGeno)

## Path where the output VCF file will be created is
## the same where the split VCF are (pathGeno)

## The files must not exist
if (!file.exists(file.path(pathGeno, "NA12003.csv.bz2")) &&
        !file.exists(file.path(pathGeno, "NA12004.csv.bz2")) &&
        !file.exists(file.path(pathGeno, "NA12005.csv.bz2"))) {

        ## Return 0 when successful
        ## The files "NA12003.csv.bz2", "NA12004.csv.bz2" and
        ## "NA12005.csv.bz2" should not be present in the current directory
        groupChr1KGSNV(pathGenoChr=pathGeno, pathOut=pathGeno)

        ## Validate that files have been created
        file.exists(file.path(pathGeno, "NA12003.csv.bz2"))
        file.exists(file.path(pathGeno, "NA12004.csv.bz2"))
        file.exists(file.path(pathGeno, "NA12005.csv.bz2"))

}

## Remove temporary directory
unlink(pathGeno, recursive=TRUE, force=TRUE)
```

`identifyRelative` *Identify genetically unrelated patients in GDS Reference file*

Description

The function identify patients that are genetically related in the Reference file. It generates a first RDS file with the list of unrelated patient. It also generates a second RDS file with the kinship coefficient between the patients.

Usage

```
identifyRelative(gds, maf = 0.05, thresh = 2^(-11/2), fileIBD, filePart)
```

Arguments

gds	an object of class SNPRelate::SNPGDSFileClass, the Reference GDS file.
maf	a single numeric representing the threshold for the minor allele frequency. Only the SNPs with ">= maf" will be used. Default: 0.05.
thresh	a single numeric representing the threshold value used to decide if a pair of individuals is ancestrally divergent. Default: 2^(-11/2).
fileIBD	a character string representing the path and file name of the RDS file that will be created. The RDS file will contain the kinship coefficient between the patients. The extension of the file must be '.rds'.
filePart	a character string representing the path and file name of the RDS file that will be created. The RDS file will contain the information about the Reference patients that are unrelated. The file will contains two lists: the list of related samples, called rels and the list of unrelated samples, called unrels. The extension of the file must be '.rds'.

Value

NULL invisibly.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required package
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Open existing demo Reference GDS file
fileGDS <- file.path(dataDir, "PopulationReferenceDemo.gds")
tmpGDS <- snpgdsOpen(fileGDS)

## Temporary output files
## The first RDS file will contain the list of unrelated patients
## The second RDS file will contain the kinship information between patients
patientTmpFile <-  "unrelatedPatients_TEMP.rds"
```

```
ibdTmpFile <- "ibd_TEMP.rds"

## Different code depending of the withr package availability
if (requireNamespace("withr", quietly=TRUE)) {

    ## Temporary output files
    ## The first RDS file will contain the list of unrelated patients
    ## The second RDS file will contain the kinship information
    ## between patients
    patientTmpFileLocal <- withr::local_file(patientTmpFile)
    ibdTmpFileLocal <- withr::local_file(ibdTmpFile)

    ## Identify unrelated patients in demo Reference GDS file
    identifyRelative(gds=tmpGDS, maf=0.05, thresh=2^(-11/2),
        fileIBD=ibdTmpFileLocal, filePart=patientTmpFileLocal)

    ## Close demo Reference GDS file
    closefn.gds(tmpGDS)

    ## Remove temporary files
    withr::deferred_run()

} else {

    ## Identify unrelated patients in demo Reference GDS file
    identifyRelative(gds=tmpGDS, maf=0.05, thresh=2^(-11/2),
        fileIBD=ibdTmpFile, filePart=patientTmpFile)

    ## Close demo Reference GDS file
    closefn.gds(tmpGDS)

    ## Remove temporary files
    unlink(patientTmpFile, force=TRUE)
    unlink(ibdTmpFile, force=TRUE)
}
```

`matKNNSynthetic` *A small* `data.frame` *containing the inferred ancestry on the synthetic profiles.*

Description

The object is a `data.frame` with 4 columns.

Usage

```
data(matKNNSynthetic)
```

Format

The `data.frame` containing the information about the synthetic profiles. The `data.frame` contains 4 columns:

- `sample.id` a `character` string representing the unique synthetic profile identifier.
- `D` a `numeric` representing the number of dimensions used to infer the ancestry of the synthetic profile.
- `K` a `numeric` representing the number of neighbors used to infer the ancestry of the synthetic profile.
- `SuperPop` a `character` string representing the inferred ancestry of the synthetic profile for the specific D and K values.

Details

This dataset can be used to test the `computeSyntheticROC` function.

Value

The `data.frame` containing the information about the synthetic profiles. The `data.frame` contains 4 columns:

- `sample.id` a `character` string representing the unique synthetic profile identifier.
- `D` a `numeric` representing the number of dimensions used to infer the ancestry of the synthetic profile.
- `K` a `numeric` representing the number of neighbors used to infer the ancestry of the synthetic profile.
- `SuperPop` a `character` string representing the inferred ancestry of the synthetic profile for the specific D and K values.

See Also

- `computeSyntheticROC` for calculating the AUROC of the inferences for specific values of D and K using the inferred ancestry results from the synthetic profiles

Examples

```
## Loading demo dataset containing pedigree information for synthetic
## profiles
data(pedSynthetic)

## Loading demo dataset containing the inferred ancestry results
## for the synthetic data
data(matKNNSynthetic)

## Retain one K and one D value
matKNN <- matKNNSynthetic[matKNNSynthetic$D == 5 & matKNNSynthetic$K == 4, ]

## Compile statistics from the
## synthetic profiles for fixed values of D and K
```

```
results <- RAIDS:::computeSyntheticROC(matKNN=matKNN,
    matKNNAncestryColumn="SuperPop",
    pedCall=pedSynthetic, pedCallAncestryColumn="superPop",
    listCall=c("EAS", "EUR", "AFR", "AMR", "SAS"))

results$matAUROC.All
results$matAUROC.Call
results$listROC.Call
```

`pedSynthetic` *A small* `data.frame` *containing the information related to synthetic profiles. The ancestry of the profiles used to generate the synthetic profiles must be present.*

Description

The object is a `data.frame` with 7 columns. The row names of the `data.frame` must be the profile unique identifiers.

Usage

```
data(pedSynthetic)
```

Format

The `data.frame` containing the information about the synthetic profiles. The row names of the `data.frame` correspond to the profile unique identifiers. The `data.frame` contains 7 columns:

- `data.id` a `character` string representing the unique synthetic profile identifier.
- `case.id` a `character` string representing the unique profile identifier that was used to generate the synthetic profile.
- `sample.type` a `character` string representing the type of profile.
- `diagnosis` a `character` string representing the diagnosis of profile that was used to generate the synthetic profile.
- `source` a `character` string representing the source of the synthetic profile.
- `study.id` a `character` string representing the name of the study to which the synthetic profile is associated.
- `superPop` a `character` string representing the super population of the profile that was used to generate the synthetic profile.

Details

This dataset can be used to test the `computeSyntheticROC` function.

Value

The `data.frame` containing the information about the synthetic profiles. The row names of the `data.frame` correspond to the profile unique identifiers. The `data.frame` contains 7 columns:

- `data.id` a `character` string representing the unique synthetic profile identifier.
- `case.id` a `character` string representing the unique profile identifier that was used to generate the synthetic profile.
- `sample.type` a `character` string representing the type of profile.
- `diagnosis` a `character` string representing the diagnosis of profile that was used to generate the synthetic profile.
- `source` a `character` string representing the source of the synthetic profile.
- `study.id` a `character` string representing the name of the study to which the synthetic profile is associated.
- `superPop` a `character` string representing the super population of the profile that was used to generate the synthetic profile.

See Also

- `computeSyntheticROC` for calculating the AUROC of the inferences for specific values of D and K using the inferred ancestry results from the synthetic profiles

Examples

```
## Loading demo dataset containing pedigree information for synthetic
## profiles
data(pedSynthetic)

## Loading demo dataset containing the inferred ancestry results
## for the synthetic data
data(matKNNSynthetic)

## Retain one K and one D value
matKNN <- matKNNSynthetic[matKNNSynthetic$D == 5 & matKNNSynthetic$K == 4, ]

## Compile statistics from the
## synthetic profiles for fixed values of D and K
results <- RAIDS:::computeSyntheticROC(matKNN=matKNN,
    matKNNAncestryColumn="SuperPop",
    pedCall=pedSynthetic, pedCallAncestryColumn="superPop",
    listCall=c("EAS", "EUR", "AFR", "AMR", "SAS"))

results$matAUROC.All
results$matAUROC.Call
results$listROC.Call
```

prepPed1KG *Prepare the pedigree file using pedigree information from Reference*

Description

Using the pedigree file from Reference, this function extracts needed information and formats it into a data.frame so in can be used in following steps of the ancestry inference process. The function also requires that the genotyping files associated to each sample be available in a specified directory.

Usage

```
prepPed1KG(filePed, pathGeno = file.path("data", "sampleGeno"), batch = 0L)
```

Arguments

filePed	a character string representing the path and file name of the pedigree file (PED file) that contains the information related to the profiles present in the Reference GDS file. The PED file must exist.
pathGeno	a character string representing the path where the Reference genotyping files for each profile are located. Only the profiles with associated genotyping files are retained in the creation of the final data.frame. The name of the genotyping files must correspond to the individual identification (Individual.ID) in the pedigree file (PED file). Default: "./data/sampleGeno".
batch	ainteger that uniquely identifies the source of the pedigree information. The Reference is usually 0L. Default: 0L.

Value

a data.frame containing the needed pedigree information from Reference. The data.frame contains those columns:

- sample.ida character string representing the profile unique ID.
- Name.IDa character string representing the profile name.
- sexa character string representing the sex of the profile.
- pop.groupa character string representing the sub-continental ancestry of the profile.
- superPop a character string representing the continental ancestry of the profile.
- superPop a integer representing the batch of the profile.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Path where the demo genotype CSV files are located
pathGeno <- file.path(dataDir, "demoProfileGenotypes")

## Demo pedigree file
pedDemoFile <- file.path(dataDir, "PedigreeDemo.ped")

## Create a data.frame containing the information of the retained
## samples (samples with existing genotyping files)
prepPed1KG(filePed=pedDemoFile, pathGeno=pathGeno, batch=0L)
```

`prepPedSynthetic1KG` *Extract the sample information from the 1KG GDS file for a list of profiles associated to a specific study in the Profile GDS file*

Description

The function extracts the information for the profiles associated to a specific study in the GDS Sample file. The information is extracted from the 'study.annot' node as a 'data.frame'.

Then, the function used the 1KG GDS file to extract specific information about each sample and add it, as an extra column, to the 'data.frame'.

As example, this function can extract the synthetic profiles for a GDS Sample and the super-population of the 1KG samples used to generate each synthetic profile would be added as an extra column to the final 'data.frame'.

Usage

```
prepPedSynthetic1KG(gdsReference, gdsSample, studyID, popName)
```

Arguments

`gdsReference`	an object of class `gdsfmt::gds.class`, the opened 1 KG GDS file.
`gdsSample`	an object of class `gdsfmt::gds.class`, the opened Profile GDS file.
`studyID`	a `character` string representing the name of the study that will be extracted from the GDS Sample 'study.annot' node.
`popName`	a `character` string representing the name of the column from the `data.frame` stored in the 'sample.annot' node of the 1KG GDS file. The column must be present in the `data.frame`.

Details

As example, this function can extract the synthetic profiles for a Profile GDS and the super-population of the 1KG samples used to generate each synthetic profile would be added as an extra column to the final 'data.frame'. In that situation, the 'popName' parameter would correspond to the super-population column and the 'studyID' parameter would be the name given to the synthetic dataset.

Value

`data.frame` containing the columns extracted from the GDS Sample 'study.annot' node with a extra column named as the 'popName' parameter that has been extracted from the 1KG GDS 'sample.annot' node. Only the rows corresponding to the specified study ('studyID' parameter) are returned.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## The open 1KG GDS file is required (this is a demo file)
dataDir <- system.file("extdata", package="RAIDS")
gds_1KG_file <- file.path(dataDir, "PopulationReferenceDemo.gds")
gds1KG <- openfn.gds(gds_1KG_file)

fileSampleGDS <- file.path(dataDir, "GDS_Sample_with_study_demo.gds")
gdsSample <- openfn.gds(fileSampleGDS)

## Extract the study information for "TCGA.Synthetic" study present in the
## Profile GDS file and merge column "superPop" from 1KG GDS to the
## returned data.frame
## This function enables to extract the super-population associated to the
## 1KG samples that has been used to create the synthetic profiles
RAIDS:::prepPedSynthetic1KG(gdsReference=gds1KG, gdsSample=gdsSample,
    studyID="TCGA.Synthetic", popName="superPop")

## The GDS files must be closed
gdsfmt::closefn.gds(gds1KG)
gdsfmt::closefn.gds(gdsSample)
```

`prepSynthetic` *Add information related to the synthetic profiles (study and synthetic reference profiles information) into a Profile GDS file*

Description

This function add entries related to synthetic profiles into a Profile GDS file. The entries are related to two types of information: the synthetic study and the synthetic profiles.

The study information is appended to the Profile GDS file "study.list" node. The "study.platform" entry is always set to 'Synthetic'.

The profile information, for all selected synthetic profiles, is appended to the Profile GDS file "study.annot" node. Both the "Source" and the "Sample.Type" entries are always set to 'Synthetic'.

The synthetic profiles are assigned unique names by combining: `prefix.data.id.profile.listSampleRef.simulation number(1 to nbSim)`

Usage

```
prepSynthetic(
  fileProfileGDS,
  listSampleRef,
  profileID,
  studyDF,
  nbSim = 1L,
  prefix = "",
  verbose = FALSE
)
```

Arguments

`fileProfileGDS`	a `character` string representing the file name of the Profile GDS file containing the information about the reference profiles used to generate the synthetic profiles.
`listSampleRef`	a `vector` of `character` string representing the identifiers of the selected 1KG profiles that will be used as reference to generate the synthetic profiles.
`profileID`	a `character` string representing the profile identifier present in the `fileProfileGDS` that will be used to generate synthetic profiles.
`studyDF`	a `data.frame` containing the information about the study associated to the analysed sample(s). The `data.frame` must have those 2 columns: "study.id" and "study.desc". Those 2 columns must be in `character` strings (no factor). Other columns can be present, such as "study.platform", but won't be used.
`nbSim`	a single positive `integer` representing the number of simulations per combination of sample and 1KG reference. Default: `1L`.
`prefix`	a single `character` string representing the prefix that is going to be added to the name of the synthetic profile. The prefix enables the creation of multiple synthetic profile using the same combination of sample and 1KG reference. Default: `""`.
`verbose`	a `logical` indicating if messages should be printed to show how the different steps in the function. Default: `FALSE`.

Value

`0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")

## Temporary Profile GDS file
fileNameGDS <- file.path(tempdir(), "ex1.gds")

## Copy the Profile GDS file demo that has been pruned and annotated
file.copy(file.path(dataDir, "ex1_demo_with_pruning_and_1KG_annot.gds"),
                 fileNameGDS)

## Information about the synthetic data set
syntheticStudyDF <- data.frame(study.id="MYDATA.Synthetic",
        study.desc="MYDATA synthetic data", study.platform="PLATFORM",
        stringsAsFactors=FALSE)

## Add information related to the synthetic profiles into the Profile GDS
prepSynthetic(fileProfileGDS=fileNameGDS,
        listSampleRef=c("HG00243", "HG00150"), profileID="ex1",
        studyDF=syntheticStudyDF, nbSim=1L, prefix="synthetic",
        verbose=FALSE)

## Open Profile GDS file
profileGDS <- openfn.gds(fileNameGDS)

## The synthetic profiles should be added in the 'study.annot' entry
tail(read.gdsn(index.gdsn(profileGDS, "study.annot")))

## The synthetic study information should be added to
## the 'study.list' entry
tail(read.gdsn(index.gdsn(profileGDS, "study.list")))

## Close GDS file (important)
closefn.gds(profileGDS)

## Remove Profile GDS file (created for demo purpose)
unlink(fileNameGDS, force=TRUE)
```

---

pruning1KGbyChr *Extract the pruned SNVs in a population reference data set (ex:1KG) by chromosome and/or allelic frequency*

Description

The function extracts the pruned SNVs in a population reference data set (ex: 1KG) by chromosome and/or allelic frequency. The pruning is done through the linkage disequilibrium analysis. The pruned SNVs are saved in a RDS file.

Usage

```
pruning1KGbyChr(
  gdsReference,
  method = "corr",
  listSamples = NULL,
  slideWindowMaxBP = 5e+05,
  thresholdLD = sqrt(0.1),
  np = 1L,
  verbose = FALSE,
  chr = NULL,
  minAF = NULL,
  outPrefix = "pruned_1KG",
  keepObj = FALSE
)
```

Arguments

gdsReference	an object of class SNPRelate::SNPGDSFileClass, an opened SNP GDS file.
method	a character string that represents the method that will be used to calculate the linkage disequilibrium in the snpgdsLDpruning() function. The 4 possible values are: "corr", "r", "dprime" and "composite". Default: "corr".
listSamples	a character string that represents the method that will be used to calculate the linkage disequilibrium in the snpgdsLDpruning() function. The 4 possible values are: "corr", "r", "dprime" and "composite". Default: "corr".
slideWindowMaxBP	a single positive integer that represents the maximum basepairs (bp) in the sliding window. This parameter is used for the LD pruning done in the runLDPruning function. Default: 5e5.
thresholdLD	a single numeric value that represents the LD threshold used in the runLDPruning function. Default: sqrt(0.1).
np	a single positive integer specifying the number of threads to be used. Default: 1L.
verbose	a logical specifying if the function must provide more information about the process. Default: FALSE.
chr	a character string representing the chromosome where the selected SNVs should belong. Only one chromosome can be handled. If NULL, the chromosome is not used as a filtering criterion. Default: NULL.
minAF	a single positive numeric representing the minimum allelic frequency used to select the SNVs. If NULL, the allelic frequency is not used as a filtering criterion. Default: NULL.

outPrefix	a character string that represents the prefix of the RDS file(s) that will be generated. Default: "pruned_1KG".
keepObj	a logical specifying if the function must save the the processed information into a second RDS file. Default: FALSE.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required libraries
library(SNPRelate)
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## The 1KG Population Reference GDS demo file (opened)
gds1KG <- snpgdsOpen(file.path(dataDir, "PopulationReferenceDemo.gds"))

## The prefix of the RDS file to be created and containing the pruned SNVs
outPrefix <- file.path(tempdir(), "Pruned_Demo_Reference")

## Create a RDS file with the pruned SNVs
RAIDS:::pruning1KGbyChr(gdsReference=gds1KG, outPrefix=outPrefix)

prunedSNVs <- readRDS(file.path(paste0(outPrefix, ".rds")))
prunedSNVs

## Close 1K GDS file
closefn.gds(gds1KG)

## Delete temporary file
unlink(paste0(outPrefix, ".rds"), force=TRUE)
```

---

pruningSample *Compute the list of pruned SNVs for a specific profile using the information from the Reference GDS file and a linkage disequilibrium analysis*

Description

This function computes the list of pruned SNVs for a specific profile. When a group of SNVs are in linkage disequilibrium, only one SNV from that group is retained. The linkage disequilibrium is calculated with the snpgdsLDpruning() function. The initial list of SNVs that are passed to the snpgdsLDpruning() function can be specified by the user.

Usage

```
pruningSample(
  gdsReference,
  method = c("corr", "r", "dprime", "composite"),
  currentProfile,
  studyID,
  listSNP = NULL,
  slideWindowMaxBP = 500000L,
  thresholdLD = sqrt(0.1),
  np = 1L,
  verbose = FALSE,
  chr = NULL,
  superPopMinAF = NULL,
  keepPrunedGDS = TRUE,
  pathProfileGDS = NULL,
  keepFile = FALSE,
  pathPrunedGDS = ".",
  outPrefix = "pruned"
)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the 1 KG GDS file (reference data set).
method	a character string that represents the method that will be used to calculate the linkage disequilibrium in the snpgdsLDpruning() function. The 4 possible values are: "corr", "r", "dprime" and "composite". Default: "corr".
currentProfile	a character string corresponding to the profile identifier used in LD pruning done by the snpgdsLDpruning() function. A Profile GDS file corresponding to the profile identifier must exist and be located in the pathProfileGDS directory.
studyID	a character string corresponding to the study identifier used in the snpgdsLDpruning function. The study identifier must be present in the Profile GDS file.
listSNP	a vector of SNVs identifiers specifying selected to be passed the the pruning function; if NULL, all SNVs are used in the snpgdsLDpruning function. Default: NULL.
slideWindowMaxBP	a single positive integer that represents the maximum basepairs (bp) in the sliding window. This parameter is used for the LD pruning done in the snpgdsLDpruning function. Default: 500000L.
thresholdLD	a single numeric value that represents the LD threshold used in the snpgdsLDpruning function. Default: sqrt(0.1).

np	a single positive integer specifying the number of threads to be used. Default: 1L.
verbose	a logicial indicating if information is shown during the process in the snpgdsLDpruning function. Default: FALSE.
chr	a character string representing the chromosome where the selected SNVs should belong. Only one chromosome can be handled. If NULL, the chromosome is not used as a filtering criterion. Default: NULL.
superPopMinAF	a single positive numeric representing the minimum allelic frequency used to select the SNVs. If NULL, the allelic frequency is not used as a filtering criterion. Default: NULL.
keepPrunedGDS	a logicial indicating if the information about the pruned SNVs should be added to the GDS Sample file. Default: TRUE.
pathProfileGDS	a character string representing the directory where the Profile GDS files will be created. The directory must exist.
keepFile	a logical indicating if RDS files containing the information about the pruned SNVs must be created. Default: FALSE.
pathPrunedGDS	a character string representing an existing directory. The directory must exist. Default: ".".
outPrefix	a character string that represents the prefix of the RDS files that will be generated. The RDS files are only generated when the parameter keepFile=TRUE. Default: "pruned".

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library for GDS
library(gdsfmt)

## Path to the demo Reference GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")
fileGDS <- file.path(dataDir, "ex1_good_small_1KG.gds")

## The data.frame containing the information about the study
## The 3 mandatory columns: "study.id", "study.desc", "study.platform"
## The entries should be strings, not factors (stringsAsFactors=FALSE)
studyDF <- data.frame(study.id = "MYDATA",
                      study.desc = "Description",
                      study.platform = "PLATFORM",
                      stringsAsFactors = FALSE)

## The data.frame containing the information about the samples
```

```
## The entries should be strings, not factors (stringsAsFactors=FALSE)
samplePED <- data.frame(Name.ID = c("ex1", "ex2"),
                    Case.ID = c("Patient_h11", "Patient_h12"),
                    Diagnosis = rep("Cancer", 2),
                    Sample.Type = rep("Primary Tumor", 2),
                    Source = rep("Databank B", 2), stringsAsFactors = FALSE)
rownames(samplePED) <- samplePED$Name.ID

## Temporary Profile GDS file
profileFile <- file.path(tempdir(), "ex1.gds")

## Copy the Profile GDS file demo that has not been pruned yet
file.copy(file.path(dataDir, "ex1_demo.gds"), profileFile)

## Open 1KG file
gds1KG <- snpgdsOpen(fileGDS)

## Compute the list of pruned SNVs for a specific profile 'ex1'
## and save it in the Profile GDS file 'ex1.gds'
pruningSample(gdsReference=gds1KG, currentProfile=c("ex1"),
              studyID = studyDF$study.id, pathProfileGDS=tempdir())

## Close the Reference GDS file (important)
closefn.gds(gds1KG)

## Check content of Profile GDS file
## The 'pruned.study' entry should be present
content <- openfn.gds(profileFile)
content

## Close the Profile GDS file (important)
closefn.gds(content)

## Remove Profile GDS file (created for demo purpose)
unlink(profileFile, force=TRUE)
```

`readSNVFileGeneric` *Read a generic SNP pileup file*

Description

The function reads a generic SNP pileup file and returns a data frame containing the information about the read counts for the SNVs present in the file.

Usage

```
readSNVFileGeneric(fileName, offset = 0L)
```

Arguments

fileName	a character string representing the name, including the path, of a text file containing the SNV read counts. The text file must be comma separated. The text file must contain those columns: Chromosome, Position, Ref, Alt, Count, File1R and File1A.
offset	a integer representing the offset to be added to the position of the SNVs. The value of offset is added to the position present in the file. Default: 0L.

Value

a data.frame containing at least:

- Chromosome a numeric representing the name of the chromosome
- Position a numeric representing the position on the chromosome
- Ref a character string representing the reference nucleotide
- Alt a character string representing the alternative nucleotide
- File1R a numeric representing the count for the reference nucleotide
- File1A a numeric representing the count for the alternative nucleotide
- count a numeric representing the total count

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Directory where demo SNP-pileup file
dataDir <- system.file("extdata/example/snpPileup", package="RAIDS")

## The SNP-pileup file
snpPileupFile <- file.path(dataDir, "ex1.generic.txt.gz")

info <- RAIDS:::readSNVFileGeneric(fileName=snpPileupFile)
head(info)
```

---

readSNVPileupFile *Read a SNP-pileup file*

Description

The function reads a generic SNP pileup file and returns a data frame containing the information about the read counts for the SNVs present in the file.

Usage

```
readSNVPileupFile(fileName, offset = 0L)
```

Arguments

fileName	a character string representing the name, including the path, of a text file containing the SNV read counts as generated by snp-pileup software. The text file must be comma separated. The text file must contain those columns: Chromosome, Position, Ref, Alt, File1R, File1A, File1E and File1D.
offset	a integer representing the offset to be added to the position of the SNVs. The value of offset is added to the position present in the file. Default: 0L.

Value

the a data.frame containing at least:

- Chromosome a numeric representing the name of the chromosome
- Position a numeric representing the position on the chromosome
- Ref a character string representing the reference nucleotide
- Alt a character string representing the alternative nucleotide
- File1R a numeric representing the count for the reference nucleotide
- File1A a numeric representing the count for the alternative nucleotide
- File1E a numeric representing the count for the errors
- File1D a numeric representing the count for the deletions
- count a numeric representing the total count

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Directory where demo SNP-pileup file
dataDir <- system.file("extdata/example/snpPileup", package="RAIDS")

## The SNP-pileup file
snpPileupFile <- file.path(dataDir, "ex1.txt.gz")

info <- RAIDS:::readSNVPileupFile(fileName=snpPileupFile)
head(info)
```

readSNVVCF *Read a VCF file with the genotypes use for the ancestry call*

Description

The function reads VCF file and returns a data frame containing the information about the read counts for the SNVs present in the file.

Usage

```
readSNVVCF(fileName, profileName = NULL, offset = 0L)
```

Arguments

fileName	a character string representing the name, including the path, of a VCF file containing the SNV read counts. The VCF must contain those genotype fields: GT, AD, DP.
profileName	a character with Name.ID for the genotype name
offset	a integer representing the offset to be added to the position of the SNVs. The value of offset is added to the position present in the file. Default: 0L.

Value

a data.frame containing at least:

- Chromosome a numeric representing the name of the chromosome
- Position a numeric representing the position on the chromosome
- Ref a character string representing the reference nucleotide
- Alt a character string representing the alternative nucleotide
- File1R a numeric representing the count for the reference nucleotide
- File1A a numeric representing the count for the alternative nucleotide
- count a numeric representing the total count

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Directory where demo SNP-pileup file
dataDir <- system.file("extdata/example/snpPileup", package="RAIDS")

## The SNP-pileup file
snpPileupFile <- file.path(dataDir, "ex1.vcf.gz")
```

```
info <- RAIDS:::readSNVVCF(fileName=snpPileupFile)
head(info)
```

`runExomeAncestry` *Run most steps leading to the ancestry inference call on a specific exome profile*

Description

This function runs most steps leading to the ancestry inference call on a specific exome profile. First, the function creates the Profile GDS file for the specific profile using the information from a RDS Sample description file and the Population reference GDS file.

Usage

```
runExomeAncestry(
  pedStudy,
  studyDF,
  pathProfileGDS,
  pathGeno,
  pathOut,
  fileReferenceGDS,
  fileReferenceAnnotGDS,
  chrInfo,
  syntheticRefDF,
  genoSource = c("snp-pileup", "generic", "VCF"),
  np = 1L,
  verbose = FALSE
)
```

Arguments

`pedStudy`	a `data.frame` with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in `character` strings (no factor). The `data.frame` must contain the information for all the samples passed in the `listSamples` parameter. Only `filePedRDS` or `pedStudy` can be defined.
`studyDF`	a `data.frame` containing the information about the study associated to the analysed sample(s). The `data.frame` must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in `character` strings (no factor).
`pathProfileGDS`	a `character` string representing the path to the directory where the GDS Profile files will be created. Default: `NULL`.
`pathGeno`	a `character` string representing the path to the directory containing the VCF output of SNP-pileup for each sample. The SNP-pileup files must be compressed (gz files) and have the name identifiers of the samples. A sample with

"Name.ID" identifier would have an associated file called if genoSource is "VCF", then "Name.ID.vcf.gz", if genoSource is "generic", then "Name.ID.generic.txt.gz" if genoSource is "snp-pileup", then "Name.ID.txt.gz".

`pathOut` a `character` string representing the path to the directory where the output files are created.

`fileReferenceGDS`
a `character` string representing the file name of the Reference GDS file. The file must exist.

`fileReferenceAnnotGDS`
a `character` string representing the file name of the Population Reference GDS Annotation file. The file must exist.

`chrInfo` a `vector` of positive `integer` values representing the length of the chromosomes. See 'details' section.

`syntheticRefDF` a `data.frame` containing a subset of reference profiles for each sub-population present in the Reference GDS file. The `data.frame` must have those columns:

- sample.id a `character` string representing the sample identifier.
- pop.group a `character` string representing the subcontinental population assigned to the sample.
- superPop a `character` string representing the super-population assigned to the sample.

`genoSource` a `character` string with two possible values: 'snp-pileup', 'generic' or 'VCF'. It specifies if the genotype files are generated by snp-pileup (Facets) or are a generic format CSV file with at least those columns: 'Chromosome', 'Position', 'Ref', 'Alt', 'Count', 'File1R' and 'File1A'. The 'Count' is the depth at the specified position; 'FileR' is the depth of the reference allele and 'File1A' is the depth of the specific alternative allele. Finally the file can be a VCF file with at least those genotype fields: GT, AD, DP.

`np` a single positive `integer` specifying the number of threads to be used. Default: `1L`.

`verbose` a `logical` indicating if messages should be printed to show how the different steps in the function. Default: `FALSE`.

Details

The runExomeAncestry() function generates 3 types of files in the OUTPUT directory.

- Ancestry InferenceThe ancestry inference CSV file (".Ancestry.csv" file)
- Inference InformatonThe inference information RDS file (".infoCall.rds" file)
- Synthetic InformationThe parameter information RDS files from the synthetic inference ("KNN.synt.*.rds" files in a sub-directory)

In addition, a sub-directory (named using the profile ID) is also created.

Value

The integer `0L` when successful. See details section for more information about the generated output files.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

#####################################################################
## Load the information about the profile
#####################################################################
data(demoPedigreeEx1)
head(demoPedigreeEx1)

#####################################################################
## The 1KG GDS file and the 1KG SNV Annotation GDS file
## need to be located in the same directory
## Note that the 1KG GDS file used for this example is a
## simplified version and CANNOT be used for any real analysis
#####################################################################
path1KG <- file.path(dataDir, "tests")

fileReferenceGDS  <- file.path(path1KG, "ex1_good_small_1KG.gds")
fileAnnotGDS <- file.path(path1KG, "ex1_good_small_1KG_Annot.gds")

#####################################################################
## The Sample SNP pileup files (one per sample) need
## to be located in the same directory.
#####################################################################
pathGeno <- file.path(dataDir, "example", "snpPileup")

#####################################################################
## The path where the Profile GDS Files (one per sample)
## will be created need to be specified.
#####################################################################
pathProfileGDS <- file.path(tempdir(), "out.tmp")

pathOut <- file.path(tempdir(), "res.out")

#####################################################################
## A data frame containing general information about the study
## is also required. The data frame must have
## those 3 columns: "studyID", "study.desc", "study.platform"
```

```
#####################################################################
studyDF <- data.frame(study.id="MYDATA",
                      study.desc="Description",
                      study.platform="PLATFORM",
                      stringsAsFactors=FALSE)

#####################################################################
## Fix seed to ensure reproducible results
#####################################################################
set.seed(3043)

gds1KG <- snpgdsOpen(fileReferenceGDS)
dataRef <- select1KGPop(gds1KG, nbProfiles=2L)
closefn.gds(gds1KG)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]



        runExomeAncestry(pedStudy=demoPedigreeEx1, studyDF=studyDF,
            pathProfileGDS=pathProfileGDS,
            pathGeno=pathGeno,
            pathOut=pathOut,
            fileReferenceGDS=fileReferenceGDS,
            fileReferenceAnnotGDS=fileAnnotGDS,
            chrInfo=chrInfo,
            syntheticRefDF=dataRef,
            genoSource="snp-pileup")

        unlink(pathProfileGDS, recursive=TRUE, force=TRUE)
        unlink(pathOut, recursive=TRUE, force=TRUE)

}
```

---

runIBDKING *Identity-by-descent (IBD) analysis*

Description

This function calculates the IDB coefficients by KING method of moment using the SNPRelate::snpgdsIBDKING function.

Usage

```
runIBDKING(gds, profileID = NULL, snpID = NULL, maf = 0.05, verbose)
```

Arguments

gds	an object of class SNPRelate::SNPGDSFileClass, an opened SNP GDS file.
profileID	a vector of character strings representing the samples to keep for the analysis. If NULL, all samples are used. Default: NULL.
snpID	a vector of character strings representing the SNPs to keep for the analysis. If NULL, all SNPs are used. Default: NULL.
maf	a single numeric representing the threshold for the minor allele frequency. Only the SNPs with ">= maf" are retained. Default: 0.05.
verbose	a logical indicating if information is shown during the process in the snpgdsIBDKING() function.

Value

a list containing:

- sample.ida character string representing the sample ids used in the analysis
- snp.ida character string representing the SNP ids used in the analysis
- k0a numeric, the IBD coefficient, the probability of sharing zero IBD
- k1a numeric, the IBD coefficient, the probability of sharing one IBD
- IBS0a numeric, the proportion of SNPs with zero IBS
- kinshipa numeric, the proportion of SNPs with zero IBS, if the parameter kinship=TRUE

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required
library(SNPRelate)

## Open an example dataset (HapMap)
genoFile <- snpgdsOpen(snpgdsExampleFileName())

## Extract CEU population
samples <- read.gdsn(index.gdsn(genoFile, "sample.id"))
CEU <- samples[
    read.gdsn(index.gdsn(genoFile, "sample.annot/pop.group"))=="CEU"]

## Infer the presence of population stratification
ibd.robust <- RAIDS:::runIBDKING(gds=genoFile, profileID=CEU, snpID=NULL,
     maf=0.05, verbose=FALSE)

## close the genotype file
```

```
snpgdsClose(genoFile)
```

`runLDPruning` *SNP pruning based on linkage disequilibrium (LD)*

Description

This function is a wrapper for the snpgdsLDpruning() function that generates a pruned subset of SNPs that are in approximate linkage equilibrium.

Usage

```
runLDPruning(
  gds,
  method,
  listSamples = NULL,
  listKeep = NULL,
  slideWindowMaxBP = 500000L,
  thresholdLD = sqrt(0.1),
  np = 1L,
  verbose
)
```

Arguments

gds	an object of class SNPGDSFileClass, a SNP GDS file.
method	a character string that represents the method that will be used to calculate the linkage disequilibrium in the snpgdsLDpruning() function. The 4 possible values are: "corr", "r", "dprime" and "composite".
listSamples	a vector of character strings corresponding to the sample identifiers used in LD pruning done by the snpgdsLDpruning() function. If NULL, all samples are used. Default: NULL.
listKeep	a vector of SNVs identifiers specifying selected; if NULL, all SNVs are used in the snpgdsLDpruning function. Default: NULL.
slideWindowMaxBP	a single positive integer that represents the maximum basepairs (bp) in the sliding window. This parameter is used for the LD pruning done in the snpgdsLDpruning() function. Default: 500000L.
thresholdLD	a single numeric value that represents the LD threshold used in the snpgdsLDpruning function. Default: sqrt(0.1).
np	a single positive integer specifying the number of threads to be used. Default: 1L.
verbose	a logical indicating if information is shown during the process in the snpgdsLDpruning() function.

Details

The SNP pruning is based on linkage disequilibrium (LD) and is done by the snpgdsLDpruning() function in the SNPRelate package (https://bioconductor.org/packages/SNPRelate/).

Value

a list of SNP identifiers stratified by chromosomes as generated by snpgdsLDpruning function.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required
library(SNPRelate)

## Open an example dataset (HapMap)
genoFile <- snpgdsOpen(snpgdsExampleFileName())

## Fix seed to get reproducible results
set.seed(1000)

## Get linkage Disequilibrium (LD) based SNP pruning
snpSet <- RAIDS:::runLDPruning(gds=genoFile, verbose=FALSE)
names(snpSet)

## Get SNP ids
snp.id <- unlist(unname(snpSet))

## Close the genotype file
snpgdsClose(genoFile)
```

---

runProfileAncestry *Run most steps leading to the ancestry inference call on a specific profile (RNA or DNA)*

Description

This function runs most steps leading to the ancestry inference call on a specific profile. First, the function creates the Profile GDS file for the specific profile using the information from a RDS Sample description file and the Population reference GDS file.

Usage

```
runProfileAncestry(
  gdsReference,
  gdsRefAnnot,
  studyDF,
  currentProfile,
  pathProfileGDS,
  pathOut,
  chrInfo,
  syntheticRefDF,
  studyDFSyn,
  listProfileRef,
  studyType = c("DNA", "RNA"),
  np = 1L,
  blockTypeID = NULL,
  verbose = FALSE
)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened Population Reference GDS file.
gdsRefAnnot	an object of class gds.class (a GDS file), the opened Population Reference SNV Annotation GDS file. This parameter is RNA specific.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).
currentProfile	a character string representing the profile identifier.
pathProfileGDS	a character string representing the path to the directory where the GDS Profile files will be created. Default: NULL.
pathOut	a character string representing the path to the directory where the output files are created.
chrInfo	a vector of positive integer values representing the length of the chromosomes. See 'details' section.
syntheticRefDF	a data.frame containing a subset of reference profiles for each sub-population present in the Reference GDS file. The data.frame must have those columns: • sample.id a character string representing the sample identifier. • pop.group a character string representing the subcontinental population assigned to the sample. • superPop a character string representing the super-population assigned to the sample.
studyDFSyn	a data.frame containing the information about the synthetic data to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).

listProfileRef	a vector of character string representing the identifiers of the selected 1KG profiles that will be used as reference to generate the synthetic profiles.
studyType	a character string representing the type of study. The possible choices are: "DNA" and "RNA". The type of study affects the way the estimation of the allelic fraction is done. Default: "DNA".
np	a single positive integer specifying the number of threads to be used. Default: 1L.
blockTypeID	a character string corresponding to the block type used to extract the block identifiers. The block type must be present in the GDS Reference Annotation file.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Details

The runWrapperAncestry() function generates 3 types of files in the pathOut directory:

- Ancestry InferenceThe ancestry inference CSV file (".Ancestry.csv" file)
- Inference InformatonThe inference information RDS file (".infoCall.rds" file)
- Synthetic InformationThe parameter information RDS files from the synthetic inference ("KNN.synt.*.rds" files in a sub-directory)

In addition, a sub-directory (named using the profile ID) is also created.

Value

The integer 0L when successful. See details section for more information about the generated output files.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

#####################################################################
## Load the information about the profile
```

```
######################################################################
data(demoPedigreeEx1)
head(demoPedigreeEx1)

######################################################################
## The 1KG GDS file and the 1KG SNV Annotation GDS file
## need to be located in the same directory
## Note that the 1KG GDS file used for this example is a
## simplified version and CANNOT be used for any real analysis
######################################################################
path1KG <- file.path(dataDir, "tests")

fileReferenceGDS  <- file.path(path1KG, "ex1_good_small_1KG.gds")
fileAnnotGDS <- file.path(path1KG, "ex1_good_small_1KG_Annot.gds")

######################################################################
## The Sample SNP pileup files (one per sample) need
## to be located in the same directory.
######################################################################
pathGeno <- file.path(dataDir, "example", "snpPileup")

######################################################################
## The path where the Profile GDS Files (one per sample)
## will be created need to be specified.
######################################################################

pathProfileGDS <- file.path(tempdir(), "outTest.tmp")

pathOut <- file.path(tempdir(), "resTest.out")

######################################################################
## A data frame containing general information about the study
## is also required. The data frame must have
## those 3 columns: "studyID", "study.desc", "study.platform"
######################################################################
studyDF <- data.frame(study.id="MYDATA",
                      study.desc="Description",
                      study.platform="PLATFORM",
                      stringsAsFactors=FALSE)

######################################################################
## Fix seed to ensure reproducible results
######################################################################
set.seed(3043)

gdsReference <- snpgdsOpen(fileReferenceGDS)
dataRef <- select1KGPop(gdsReference, nbProfiles=2L)
closefn.gds(gdsReference)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {
```

```
## Chromosome length information
## chr23 is chrX, chr24 is chrY and chrM is 25
chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

studyDFSyn <- data.frame(study.id=paste0(studyDF$study.id, ".Synthetic"),
    study.desc=paste0(studyDF$study.id, " synthetic data"),
    study.platform=studyDF$study.platform, stringsAsFactors=FALSE)

listProfileRef <- dataRef$sample.id
profileFile <- file.path(pathProfileGDS, "ex1.gds")

## Not run:

    dir.create(pathProfileGDS)
    dir.create(pathOut)
    file.copy(file.path(dataDir, "tests", "ex1_demo.gds"), profileFile)

    gdsReference <- snpgdsOpen(fileReferenceGDS)
    gdsRefAnnot <- openfn.gds(fileAnnotGDS)

    RAIDS:::runProfileAncestry(gdsReference=gdsReference,
         gdsRefAnnot=gdsRefAnnot,
         studyDF=studyDF, currentProfile=ped[1,"Name.ID"],
         pathProfileGDS=pathProfileGDS,
         pathOut=pathOut,
         chrInfo=chrInfo,
         syntheticRefDF=dataRef,
         studyDFSyn=studyDFSyn,
         listProfileRef=listProfileRef,
         studyType="DNA")

    closefn.gds(gdsReference)
    closefn.gds(gdsRefAnnot)

    unlink(pathProfileGDS, recursive=TRUE, force=TRUE)
    unlink(pathOut, recursive=TRUE, force=TRUE)


## End(Not run)
}
```

`runRNAAncestry` *Run most steps leading to the ancestry inference call on a specific RNA profile*

Description

This function runs most steps leading to the ancestry inference call on a specific RNA profile. First, the function creates the Profile GDS file for the specific profile using the information from a RDS Sample description file and the Population Reference GDS file.

Usage

```
runRNAAncestry(
  pedStudy,
  studyDF,
  pathProfileGDS,
  pathGeno,
  pathOut,
  fileReferenceGDS,
  fileReferenceAnnotGDS,
  chrInfo,
  syntheticRefDF,
  genoSource = c("snp-pileup", "generic", "VCF"),
  np = 1L,
  blockTypeID,
  verbose = FALSE
)
```

Arguments

pedStudy	a data.frame with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in character strings (no factor). The data.frame must contain the information for all the samples passed in the listSamples parameter. Only filePedRDS or pedStudy can be defined.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).
pathProfileGDS	a character string representing the path to the directory where the GDS Profile files will be created. Default: NULL.
pathGeno	a character string representing the path to the directory containing the VCF output of SNP-pileup for each sample. The SNP-pileup files must be compressed (gz files) and have the name identifiers of the samples. A sample with "Name.ID" identifier would have an associated file called if genoSource is "VCF", then "Name.ID.vcf.gz", if genoSource is "generic", then "Name.ID.generic.txt.gz" if genoSource is "snp-pileup", then "Name.ID.txt.gz".
pathOut	a character string representing the path to the directory where the output files are created.
fileReferenceGDS	a character string representing the file name of the Population Reference GDS file. The file must exist.
fileReferenceAnnotGDS	a character string representing the file name of the Population Reference GDS Annotation file. The file must exist.
chrInfo	a vector of positive integer values representing the length of the chromosomes. See 'details' section.
syntheticRefDF	a data.frame containing a subset of reference profiles for each sub-population present in the Reference GDS file. The data.frame must have those columns:

- sample.id a character string representing the sample identifier.
- pop.group a character string representing the subcontinental population assigned to the sample.
- superPop a character string representing the super-population assigned to the sample.

genoSource — a character string with two possible values: 'snp-pileup', 'generic' or 'VCF'. It specifies if the genotype files are generated by snp-pileup (Facets) or are a generic format CSV file with at least those columns: 'Chromosome', 'Position', 'Ref', 'Alt', 'Count', 'File1R' and 'File1A'. The 'Count' is the depth at the specified position; 'FileR' is the depth of the reference allele and 'File1A' is the depth of the specific alternative allele. Finally the file can be a VCF file with at least those genotype fields: GT, AD, DP.

np — a single positive integer specifying the number of threads to be used. Default: 1L.

blockTypeID — a character string corresponding to the block type used to extract the block identifiers. The block type must be present in the GDS Reference Annotation file.

verbose — a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Details

The runExomeAncestry() function generates 3 types of files in the OUTPUT directory.

- Ancestry InferenceThe ancestry inference CSV file (".Ancestry.csv" file)
- Inference InformatonThe inference information RDS file (".infoCall.rds" file)
- Synthetic InformationThe parameter information RDS files from the synthetic inference ("KNN.synt.*.rds" files in a sub-directory)

In addition, a sub-directory (named using the profile ID) is also created.

Value

The integer 0L when successful. See details section for more information about the generated output files.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

#####################################################################
## Load the information about the profile
#####################################################################
data(demoPedigreeEx1)
head(demoPedigreeEx1)

#####################################################################
## The 1KG GDS file and the 1KG SNV Annotation GDS file
## need to be located in the same directory
## Note that the 1KG GDS file used for this example is a
## simplified version and CANNOT be used for any real analysis
#####################################################################
path1KG <- file.path(dataDir, "tests")

fileReferenceGDS <- file.path(path1KG, "ex1_good_small_1KG.gds")
fileAnnotGDS <- file.path(path1KG, "ex1_good_small_1KG_Annot.gds")

#####################################################################
## The Sample SNP pileup files (one per sample) need
## to be located in the same directory.
#####################################################################
pathGeno <- file.path(dataDir, "example", "snpPileup")

#####################################################################
## The path where the Profile GDS Files (one per sample)
## will be created need to be specified.
#####################################################################
pathProfileGDS <- file.path(tempdir(), "out.tmp")

pathOut <- file.path(tempdir(), "res.out")

#####################################################################
## A data frame containing general information about the study
## is also required. The data frame must have
## those 3 columns: "studyID", "study.desc", "study.platform"
#####################################################################
studyDF <- data.frame(study.id="MYDATA",
                      study.desc="Description",
                      study.platform="PLATFORM",
                      stringsAsFactors=FALSE)

#####################################################################
## Fix seed to ensure reproducible results
#####################################################################
set.seed(3043)
```

```
gds1KG <- snpgdsOpen(fileReferenceGDS)
dataRef <- select1KGPop(gds1KG, nbProfiles=2L)
closefn.gds(gds1KG)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]



        runRNAAncestry(pedStudy=demoPedigreeEx1, studyDF=studyDF,
            pathProfileGDS=pathProfileGDS,
            pathGeno=pathGeno,
            pathOut=pathOut,
            fileReferenceGDS=fileReferenceGDS,
            fileReferenceAnnotGDS=fileAnnotGDS,
            chrInfo=chrInfo,
            syntheticRefDF=dataRef,
            blockTypeID="GeneS.Ensembl.Hsapiens.v86",
            genoSource="snp-pileup")

        unlink(pathProfileGDS, recursive=TRUE, force=TRUE)
        unlink(pathOut, recursive=TRUE, force=TRUE)


}
```

`runWrapperAncestry` *Run most steps leading to the ancestry inference call on a specific profile (RNA or DNA)*

Description

This function runs most steps leading to the ancestry inference call on a specific profile. First, the function creates the Profile GDS file for the specific profile using the information from a RDS Sample description file and the Population reference GDS file.

Usage

```
runWrapperAncestry(
  pedStudy,
  studyDF,
  pathProfileGDS,
  pathGeno,
```

```
  pathOut,
  fileReferenceGDS,
  fileReferenceAnnotGDS,
  chrInfo,
  syntheticRefDF,
  genoSource = c("snp-pileup", "generic", "VCF"),
  studyType = c("DNA", "RNA"),
  np = 1L,
  blockTypeID = NULL,
  verbose = FALSE
)
```

Arguments

pedStudy	a data.frame with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in character strings (no factor). The data.frame must contain the information for all the samples passed in the listSamples parameter. Only filePedRDS or pedStudy can be defined.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).
pathProfileGDS	a character string representing the path to the directory where the GDS Profile files will be created. Default: NULL.
pathGeno	a character string representing the path to the directory containing the VCF output of SNP-pileup for each sample. The SNP-pileup files must be compressed (gz files) and have the name identifiers of the samples. A sample with "Name.ID" identifier would have an associated file called if genoSource is "VCF", then "Name.ID.vcf.gz", if genoSource is "generic", then "Name.ID.generic.txt.gz" if genoSource is "snp-pileup", then "Name.ID.txt.gz".
pathOut	a character string representing the path to the directory where the output files are created.
fileReferenceGDS	a character string representing the file name of the Reference GDS file. The file must exist.
fileReferenceAnnotGDS	a character string representing the file name of the Reference GDS Annotation file. The file must exist.
chrInfo	a vector of positive integer values representing the length of the chromosomes. See 'details' section.
syntheticRefDF	a data.frame containing a subset of reference profiles for each sub-population present in the Reference GDS file. The data.frame must have those columns: • sample.id a character string representing the sample identifier. • pop.group a character string representing the subcontinental population assigned to the sample. • superPop a character string representing the super-population assigned to the sample.

genoSource	a character string with two possible values: 'snp-pileup', 'generic' or 'VCF'. It specifies if the genotype files are generated by snp-pileup (Facets) or are a generic format CSV file with at least those columns: 'Chromosome', 'Position', 'Ref', 'Alt', 'Count', 'File1R' and 'File1A'. The 'Count' is the depth at the specified position; 'FileR' is the depth of the reference allele and 'File1A' is the depth of the specific alternative allele. Finally the file can be a VCF file with at least those genotype fields: GT, AD, DP.
studyType	a character string representing the type of study. The possible choices are: "DNA" and "RNA". The type of study affects the way the estimation of the allelic fraction is done. Default: "DNA".
np	a single positive integer specifying the number of threads to be used. Default: 1L.
blockTypeID	a character string corresponding to the block type used to extract the block identifiers. The block type must be present in the GDS Reference Annotation file.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Details

The runWrapperAncestry() function generates 3 types of files in the pathOut directory.

- Ancestry InferenceThe ancestry inference CSV file (".Ancestry.csv" file)
- Inference InformatonThe inference information RDS file (".infoCall.rds" file)
- Synthetic InformationThe parameter information RDS files from the synthetic inference ("KNN.synt.*.rds" files in a sub-directory)

In addition, a sub-directory (named using the profile ID) is also created.

Value

The integer 0L when successful. See details section for more information about the generated output files.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library for GDS
library(SNPRelate)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

######################################################################
## Load the information about the profile
######################################################################
data(demoPedigreeEx1)
head(demoPedigreeEx1)

######################################################################
## The 1KG GDS file and the 1KG SNV Annotation GDS file
## need to be located in the same directory
## Note that the 1KG GDS file used for this example is a
## simplified version and CANNOT be used for any real analysis
######################################################################
path1KG <- file.path(dataDir, "tests")

fileReferenceGDS  <- file.path(path1KG, "ex1_good_small_1KG.gds")
fileAnnotGDS <- file.path(path1KG, "ex1_good_small_1KG_Annot.gds")

######################################################################
## The Sample SNP pileup files (one per sample) need
## to be located in the same directory.
######################################################################
pathGeno <- file.path(dataDir, "example", "snpPileup")

######################################################################
## The path where the Profile GDS Files (one per sample)
## will be created need to be specified.
######################################################################
pathProfileGDS <- file.path(tempdir(), "out.tmp")

pathOut <- file.path(tempdir(), "res.out")

######################################################################
## A data frame containing general information about the study
## is also required. The data frame must have
## those 3 columns: "studyID", "study.desc", "study.platform"
######################################################################
studyDF <- data.frame(study.id="MYDATA",
                        study.desc="Description",
                        study.platform="PLATFORM",
                        stringsAsFactors=FALSE)

######################################################################
## Fix seed to ensure reproducible results
######################################################################
set.seed(3043)
```

```
gds1KG <- snpgdsOpen(fileReferenceGDS)
dataRef <- select1KGPop(gds1KG, nbProfiles=2L)
closefn.gds(gds1KG)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

    ## Not run:

        RAIDS:::runWrapperAncestry(pedStudy=demoPedigreeEx1, studyDF=studyDF,
            pathProfileGDS=pathProfileGDS,
            pathGeno=pathGeno, pathOut=pathOut,
            fileReferenceGDS=fileReferenceGDS,
            fileReferenceAnnotGDS=fileAnnotGDS,
            chrInfo=chrInfo, syntheticRefDF=dataRef,
            studyType="DNA", genoSource="snp-pileup")

        unlink(pathProfileGDS, recursive=TRUE, force=TRUE)
        unlink(pathOut, recursive=TRUE, force=TRUE)


## End(Not run)
}
```

`select1KGPop` *Random selection of a specific number of reference profiles in each subcontinental population present in the 1KG GDS file*

Description

The function randomly selects a fixed number of reference for each subcontinental population present in the 1KG GDS file. When a subcontinental population has less samples than the fixed number, all samples from the subcontinental population are selected.

Usage

```
select1KGPop(gdsReference, nbProfiles)
```

Arguments

`gdsReference`	an object of class gds.class (a GDS file), the opened 1KG GDS file.

nbProfiles — a single positive integer representing the number of samples that will be selected for each subcontinental population present in the 1KG GDS file. If the number of samples in a specific subcontinental population is smaller than the nbProfiles, the number of samples selected in this subcontinental population will correspond to the size of this population.

Value

a data.frame containing those columns:

- sample.id a character string representing the sample identifier.
- pop.group a character string representing the subcontinental population assigned to the sample.
- superPop a character string representing the super-population assigned to the sample.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## The number of samples needed by subcontinental population
## The number is small for demonstration purpose
nbProfiles <- 5L

## Open 1KG GDS Demo file
## This file only one superpopulation (for demonstration purpose)
dataDir <- system.file("extdata", package="RAIDS")
fileGDS <- file.path(dataDir, "PopulationReferenceDemo.gds")
gdsFileOpen <- openfn.gds(fileGDS, readonly=TRUE)

## Extract a selected number of random samples
## for each subcontinental population
## In the 1KG GDS Demo file, there is one subcontinental population
dataR <- select1KGPop(gdsReference=gdsFileOpen, nbProfiles=nbProfiles)

## Close the 1KG GDS Demo file (important)
closefn.gds(gdsFileOpen)
```

---

selParaPCAUpQuartile *Compile all the inferred ancestry results done on the synthetic profiles for different D and K values in the objective of selecting the optimal D and K values for a specific profile*

Description

The function calculates the accuracy of the inferred ancestry called done on the synthetic profiles for different D and K values. The accuracy is also calculated for each super-population used to generate the synthetic profiles. The known ancestry from the reference profiles used to generate the synthetic profiles is required to calculate the accuracy.

Usage

```
selParaPCAUpQuartile(
  matKNN,
  pedCall,
  refCall,
  predCall,
  listCall,
  kList = seq(3, 15, 1),
  pcaList = seq(2, 15, 1)
)
```

Arguments

`matKNN`	a `data.frame` containing the inferred ancestry for the synthetic profiles for different *K* and *D* values. The `data.frame` must contained those columns: "sample.id", "D", "K" and the fourth column name must correspond to the `predCall` argument.
`pedCall`	a `data.frame` containing the information about the super-population information from the 1KG GDS file for profiles used to generate the synthetic profiles. The `data.frame` must contained a column named as the `refCall` argument.
`refCall`	a `character` string representing the name of the column that contains the known ancestry for the reference profiles in the Reference GDS file.
`predCall`	a `character` string representing the name of the column that contains the inferred ancestry for the specified profiles. The column must be present in the `matKNN` `data.frame` argument.
`listCall`	a `vector` of `character` strings representing the list of possible ancestry assignations.
`kList`	a `vector` of `integer` representing the list of values tested for the *K* parameter. The *K* parameter represents the number of neighbors used in the K-nearest neighbor analysis. Default: `seq(3,15,1)`.
`pcaList`	a `vector` of `integer` representing the list of values tested for the *D* parameter. The *D* parameter represents the number of dimensions used in the PCA analysis. Default: `seq(2,15,1)`.

Value

a `list` containing 5 entries:

- `dfPCA` a `data.frame` containing statistical results on all combined synthetic results done with a fixed value of `D` (the number of dimensions). The `data.frame` contains those columns:

  - D a numeric representing the value of D (the number of dimensions).
  - median a numeric representing the median of the minimum AUROC obtained (within super populations) for all combination of the fixed D value and all tested K values.
  - mad a numeric representing the MAD of the minimum AUROC obtained (within super populations) for all combination of the fixed D value and all tested K values.
  - upQuartile a numeric representing the upper quartile of the minimum AUROC obtained (within super populations) for all combination of the fixed D value and all tested K values.
  - k a numeric representing the optimal K value (the number of neighbors) for a fixed D value.
- dfPop a data.frame containing statistical results on all combined synthetic results done with different values of D (the number of dimensions) and K (the number of neighbors). The data.frame contains those columns:
  - D a numeric representing the value of D (the number of dimensions).
  - K a numeric representing the value of K (the number of neighbors).
  - AUROC.min a numeric representing the minimum accuracy obtained by grouping all the synthetic results by super-populations, for the specified values of D and K.
  - AUROC a numeric representing the accuracy obtained by grouping all the synthetic results for the specified values of D and K.
  - Accu.CM a numeric representing the value of accuracy of the confusion matrix obtained by grouping all the synthetic results for the specified values of D and K.
- D a numeric representing the optimal D value (the number of dimensions) for the specific profile.
- K a numeric representing the optimal K value (the number of neighbors) for the specific profile.
- listD a numeric representing the optimal D values (the number of dimensions) for the specific profile. More than one D is possible.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Loading demo dataset containing pedigree information for synthetic
## profiles and known ancestry of the profiles used to generate the
## synthetic profiles
data(pedSynthetic)

## Loading demo dataset containing the inferred ancestry results
## for the synthetic data
data(matKNNSynthetic)

## Compile all the results for ancestry inference done on the
## synthetic profiles for different D and K values
## Select the optimal D and K values
results <- RAIDS:::selParaPCAUpQuartile(matKNN=matKNNSynthetic,
    pedCall=pedSynthetic, refCall="superPop", predCall="SuperPop",
    listCall=c("EAS", "EUR", "AFR", "AMR", "SAS"), kList=seq(3,15,1),
    pcaList=seq(2,15,1))
```

```
results$D
results$K
```

---

snpPositionDemo *A small* data.frame *containing the SNV information.*

Description

The object is a data.frame with 17 columns.

Usage

```
data(snpPositionDemo)
```

Format

The data.frame containing the information about the synthetic profiles. The data.frame contains 4 columns:

- cnt.tot a integer representing the number of reads at the SNV position.
- cnt.ref a integer representing the number of reads corresponding to the reference at the SNV position.
- cnt.alt a integer representing the number of reads different than the reference at the SNV position.
- snp.pos a integer representing the position of the SNV on the chromosome.
- snp.chr a integer representing the chromosome on which the SNV is located.
- normal.geno a integer representing the genotype (0=wild-type reference; 1=heterozygote; 2=homozygote alternative; 3=unkown).
- pruned a logical indicated if the SNV is pruned.
- snp.index a integer representing the index of the SNV in the reference SNV GDS file.
- keep a logical indicated if the genotype exists for the SNV.
- hetero a logical indicated if the SNV is heterozygote.
- homo a logical indicated if the SNV is homozygote.
- block.id a integer representing the block identifier associated to the current SNV.
- phase a integer representing the block identifier associated to the current SNV.
- lap a numeric representing the lower allelic fraction.
- LOH a integer indicating if the SNV is in an LOH region (0=not LOH, 1=in LOH).
- imbAR a integer indicating if the SNV is in an imbalanced region (-1=not classified as imbalanced or LOH, 0=in LOH; 1=tested positive for imbalance in at least 1 window).
- freq a numeric representing the frequency of the variant in the the reference.

Details

This dataset can be used to test the calcAFMLRNA and tableBlockAF internal functions.

Value

The data.frame containing the information about the synthetic profiles. The data.frame contains 4 columns:

- cnt.tot a integer representing the number of reads at the SNV position.
- cnt.ref a integer representing the number of reads corresponding to the reference at the SNV position.
- cnt.alt a integer representing the number of reads different than the reference at the SNV position.
- snp.pos a integer representing the position of the SNV on the chromosome.
- snp.chr a integer representing the chromosome on which the SNV is located.
- normal.geno a integer representing the genotype (0=wild-type reference; 1=heterozygote; 2=homozygote alternative; 3=unkown).
- pruned a logical indicated if the SNV is pruned.
- snp.index a integer representing the index of the SNV in the reference SNV GDS file.
- keep a logical indicated if the genotype exists for the SNV.
- hetero a logical indicated if the SNV is heterozygote.
- homo a logical indicated if the SNV is homozygote.
- block.id a integer representing the block identifier associated to the current SNV.
- phase a integer representing the block identifier associated to the current SNV.
- lap a numeric representing the lower allelic fraction.
- LOH a integer indicating if the SNV is in an LOH region (0=not LOH, 1=in LOH).
- imbAR a integer indicating if the SNV is in an imbalanced region (-1=not classified as imbalanced or LOH, 0=in LOH; 1=tested positive for imbalance in at least 1 window).
- freq a numeric representing the frequency of the variant in the the reference.

Examples

```
## Loading demo dataset containing SNV information
data(snpPositionDemo)

## Only use a subset of heterozygote SNVs related to one block
subset <- snpPositionDemo[which(snpPositionDemo$block.id == 2750 &
                  snpPositionDemo$hetero), c("cnt.ref", "cnt.alt", "phase")]

## Compute the log likelihood ratio based on the coverage of
## each allele in a specific block
result <- RAIDS:::calcAFMLRNA(subset)
head(result)
```

`snvListVCF` *Generate a VCF with the information from the SNPs that pass a cut-off threshold*

Description

This function extract the SNPs that pass a frequency cut-off in at least one super population from a GDS SNP information file and save the retained SNP information into a VCF file.

Usage

```
snvListVCF(gdsReference, fileOut, offset = 0L, freqCutoff = NULL)
```

Arguments

`gdsReference`	an object of class `gds.class` (a GDS file), the 1KG GDS file.
`fileOut`	a `character` string representing the path and file name of the VCF file that will be created wit the retained SNP information. The file should have the ".vcf" extension.
`offset`	a single `integer` that is added to the SNP position to switch from 0-based to 1-based coordinate when needed (or reverse). Default: `0L`.
`freqCutoff`	a single positive `numeric` specifying the cut-off to keep a SNP. If `NULL`, all SNPs are retained. Default: `NULL`.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Demo 1KG Reference GDS file
fileGDS <- openfn.gds(file.path(dataDir,
                   "PopulationReferenceDemo.gds"))

## Output VCF file that will be created (temporary)
vcfFile <- file.path(tempdir(), "Demo_TMP_01.vcf")

## Create a VCF file with the SNV dataset present in the GDS file
```

```
## No cutoff on frequency, so all SNVs are saved
snvListVCF(gdsReference=fileGDS, fileOut=vcfFile, offset=0L,
                    freqCutoff=NULL)

## Close GDS file (IMPORTANT)
closefn.gds(fileGDS)

## Remove temporary VCF file
unlink(vcfFile, force=TRUE)
```

---

splitSelectByPop *Group samples per subcontinental population*

Description

The function groups the samples per subcontinental population and generates a matrix containing the sample identifiers and where each column is a subcontinental population.

Usage

```
splitSelectByPop(dataRef)
```

Arguments

dataRef	a data.frame containing those columns:

- sample.id a character string representing the sample identifier.
- pop.group a character string representing the subcontinental population assigned to the sample.
- superPop a character string representing the super-population assigned to the sample.

Value

a matrix containing the sample identifiers and where each column is the name of a subcontinental population. The number of row corresponds to the number of samples for each subcontinental population.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## A data.frame containing samples from 2 subcontinental populations
demo <- data.frame(sample.id=c("SampleA", "SampleB", "SampleC", "SampleD"),
    pop.group=c("TSI", "TSI", "YRI", "YRI"),
    superPop=c("EUR", "EUR", "AFR", "AFR"))

## Generate a matrix populated with the sample identifiers and where
## each row is a subcontinental population
splitSelectByPop(dataRef=demo)
```

`syntheticGeno` *Generate synthetic profiles for each cancer profile and 1KG reference profile combination and add them to the Profile GDS file*

Description

The functions uses one cancer profile in combination with one 1KG reference profile to generate an synthetic profile that is saved in the Profile GDS file.

When more than one 1KG reference profiles are specified, the function recursively generates synthetic profiles for each cancer profile + 1KG reference profile combination.

The number of synthetic profiles generated by combination is specified by the number of simulation requested.

Usage

```
syntheticGeno(
  gdsReference,
  gdsRefAnnot,
  fileProfileGDS,
  profileID,
  listSampleRef,
  nbSim = 1L,
  prefix = "",
  pRecomb = 0.01,
  minProb = 0.999,
  seqError = 0.001
)
```

Arguments

`gdsReference`	an object of class `gds.class` (a GDS file), the opened 1KG GDS file.
`gdsRefAnnot`	an object of class `gds.class` (a GDS file), the opened 1KG SNV Annotation GDS file.
`fileProfileGDS`	a `character` string representing the file name of Profile GDS file containing the information about the sample. The file must exist.

profileID	a character string representing the unique identifier of the cancer profile.
listSampleRef	a vector of character strings representing the sample identifiers of the 1KG selected reference samples.
nbSim	a single positive integer representing the number of simulations that will be generated per sample + 1KG reference combination. Default: 1L.
prefix	a character string that represent the prefix that will be added to the name of the synthetic profiles generated by the function. Default: "".
pRecomb	a single positive numeric between 0 and 1 that represents the frequency of phase switching in the synthetic profiles, Default: 0.01.
minProb	a single positive numeric between 0 and 1 that represents the probability that the genotype is correct. Default: 0.999.
seqError	a single positive numeric between 0 and 1 representing the sequencing error rate. Default: 0.001.

Value

The integer 0L when the function is successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")

## Profile GDS file (temporary)
fileNameGDS <- file.path(tempdir(), "ex1.gds")

## Copy the Profile GDS file demo that has been pruned and annotated
file.copy(file.path(dataDir, "ex1_demo_with_pruning_and_1KG_annot.gds"),
                fileNameGDS)

## Information about the synthetic data set
syntheticStudyDF <- data.frame(study.id="MYDATA.Synthetic",
        study.desc="MYDATA synthetic data", study.platform="PLATFORM",
        stringsAsFactors=FALSE)

## Add information related to the synthetic profiles into the Profile GDS
prepSynthetic(fileProfileGDS=fileNameGDS,
        listSampleRef=c("HG00243", "HG00150"), profileID="ex1",
        studyDF=syntheticStudyDF, nbSim=1L, prefix="synthTest",
        verbose=FALSE)

## The 1KG files
```

```
gds1KG <- snpgdsOpen(file.path(dataDir,
                           "ex1_good_small_1KG.gds"))
gds1KGAnnot <- openfn.gds(file.path(dataDir,
                           "ex1_good_small_1KG_Annot.gds"))

## Generate the synthetic profiles and add them into the Profile GDS
syntheticGeno(gdsReference=gds1KG, gdsRefAnnot=gds1KGAnnot,
        fileProfileGDS=fileNameGDS, profileID="ex1",
        listSampleRef=c("HG00243", "HG00150"), nbSim=1,
        prefix="synthTest",
        pRecomb=0.01, minProb=0.999, seqError=0.001)

## Open Profile GDS file
profileGDS <- openfn.gds(fileNameGDS)

tail(read.gdsn(index.gdsn(profileGDS, "sample.id")))

## Close GDS files (important)
closefn.gds(profileGDS)
closefn.gds(gds1KG)
closefn.gds(gds1KGAnnot)

## Remove Profile GDS file (created for demo purpose)
unlink(fileNameGDS, force=TRUE)
```

`tableBlockAF` *Compile the information about the SNVs for each block*

Description

The function evaluates a score about loss of heterozygosity and allelic fraction for each block. It generates specific information about the variants in the block, like the number of homozygotes or heterozygotes. In the case of RNA-seq, the blocks are genes.

Usage

```
tableBlockAF(snpPos)
```

Arguments

`snpPos`	a `data.frame` with lower allelic fraction (lap) for the SNVs with coverage > `minCov`, for a specific chromosome.

Value

a `data.frame` containing only heterozygote SNV information. The `data.frame` contain those columns:

- block a single `integer` representing the unique identifier of the block.
- aRF a single `numeric` representing the final allelic fraction; not computed yet, `-1` value assigned to all entries.
- aFraction a single `integer` representing the possible allelic fraction in absence of loss of heterozygosity (LOH).
- lR a single `integer` representing the coverage for the alternative allele.
- nPhase a single `integer` representing the number of SNV phases.
- sumAlleleLow a single `integer` representing the sum of the alleles with the less coverage.
- sumAlleleHigh a single `integer` representing the sum of the alleles with more coverage.
- lH a single `numeric` for the homozygotes log10 of the product frequencies of the allele not found in the profile (not a probability).
- lM a single `numeric` log10 product frequency allele in population.
- lRhomo a single `numeric` representing the score `lH - lM`.
- nbHomo a single `integer` representing the number of homozygote SNVs per block.
- nbKeep a single `integer` representing the number of SNVs retained per block.
- nbHetero a single `integer` representing the number of heterozygote SNVs per block.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Loading demo dataset containing SNV information
data(snpPositionDemo)

## Retain SNVs on chromosome 1
subset <- snpPositionDemo[which(snpPositionDemo$snp.chr == 1),]

##Compile the information about the SNVs for each block
result <- RAIDS:::tableBlockAF(subset)
head(result)
```

`testAlleleFractionChange` *Calculate presence of allelic fraction change using specified SNVs separately and together*

Description

The function tests allelic fraction change using all specified SNVs separately and together. The function reports the associated results, including statistic for the region represented by all the SNVs.

Usage

```
testAlleleFractionChange(matCov, pCutOff = -3, vMean)
```

Arguments

matCov	a data.frame containing only heterozygote SNVs. The data.frame must contain those columns: • cnt.ref a single integer representing the coverage for the reference allele. • cnt.alt a single integer representing the coverage for the alternative allele.
pCutOff	a numeric representing the cut-off for considering a region imbalanced when comparing likelihood to gave allelic fraction change and likelihood not to have allelic fraction change. Default: -3.
vMean	a positive numeric representing the current ratio (minor allele/(minor allele + second allele)) that is going to be used as reference to see if there is a allelic fraction change.

Value

a list containing 4 entries:

- pWin a vector of numeric representing the probability (x2) of obtaining the current alternative/(alternative+reference) ratio from a reference distribution specified by user.
- pa integer indicating if all SNVs tested positive (1=TRUE, 0=FALSE). The cut-off is 0.5.
- pCuta integer indicating if all SNVs tested positive (1=TRUE, 0-FALSE).
- pCut1a integer indicating if the region tested positive (1=TRUE, 0=FALSE) for allelic ratio change.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Data frame with SNV information
snpInfo <- data.frame(
    cnt.ref=c(40, 17, 27, 15, 4, 14, 16, 32),
    cnt.alt=c(2, 4, 5, 10, 7, 23, 0, 0))

RAIDS:::testAlleleFractionChange(matCov=snpInfo, pCutOff=-3, vMean=0.5)
```

`testEmptyBox` *Calculate imbalance region using specified heterozygote SNVs separately and together*

Description

The function tests imbalance region using all specified SNVs separately and together. The function reports the associated results, including statistic for the region.

Usage

```
testEmptyBox(matCov, pCutOff = -3)
```

Arguments

`matCov`	a `data.frame` containing only heterozygote SNVs. The `data.frame` must contain those columns: • cnt.ref a single `integer` representing the coverage for the reference allele. • cnt.alt a single `integer` representing the coverage for the alternative allele.
`pCutOff`	a `numeric` representing the cut-off for considering a region imbalanced when comparing likelihood to be imbalanced and likelihood not to be imbalanced. Default: `-3`.

Value

a `list` containing 4 entries:

- pWin a `vector` of `numeric` representing the probability (x2) of obtaining the current alternative/(alternative+reference) ratio from a 0.5 distribution.
- p a `numeric` representing the likelihood for the region
- pCut a `integer` indicating if all SNVs tested positive (1=TRUE, 0=FALSE). The cut-off is 0.5.
- pCut1 a `integer` indicating if the window tested positive (1=TRUE, 0=FALSE) for imbalance.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Data frame with SNV information for a list of heterozygote SNVs
snpInfo <- data.frame(
    cnt.ref=c(40, 17, 27, 15, 4, 14, 16, 32),
    cnt.alt=c(2, 4, 5, 10, 7, 23, 0, 0))

## Calculate imbalance for the region represented by the SNVs
RAIDS:::testEmptyBox(matCov=snpInfo, pCutOff=-3)
```

validateAdd1KG2SampleGDS *Validate input parameters for add1KG2SampleGDS() function*

Description

This function validates the input parameters for the add1KG2SampleGDS function.

Usage

```
validateAdd1KG2SampleGDS(gdsReference, gdsProfileFile, currentProfile, studyID)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened Population Reference GDS file.
gdsProfileFile	a character string representing the path and file name of the Profile GDS file. The Profile GDS file must exist.
currentProfile	a character string corresponding to the profile identifier associated to the current list of pruned SNVs.
studyID	a character string corresponding to the study identifier associated to the current list of pruned SNVs.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## The 1KG Population Reference GDS demo file (opened)
gds1KG <- openfn.gds(file.path(dataDir,
                "PopulationReferenceDemo.gds"), readonly=TRUE)

## The validatiion should be successful
RAIDS:::validateAdd1KG2SampleGDS(gdsReference=gds1KG,
```

```
    gdsProfileFile=file.path(dataDir, "GDS_Sample_with_study_demo.gds"),
    currentProfile="Sample01", studyID="Synthetic")

## All GDS file must be closed
closefn.gds(gdsfile=gds1KG)
```

---

validateAddStudy1Kg *Validate the parameters for the addStudy1Kg() function*

Description

The function validates the input parameters for the addStudy1Kg function. When a parameter is not as expected, an error message is generated.

Usage

```
validateAddStudy1Kg(gdsReference, fileProfileGDS, verbose)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened 1KG GDS file.
fileProfileGDS	a character string representing the path and file name of the GDS Sample file. The GDS Sample file must exist.
verbose	a logical indicating if messages should be printed to show how the different steps in the function.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata", package="RAIDS")
fileReferenceGDS <- file.path(dataDir, "PopulationReferenceDemo.gds")
gds1KG <- snpgdsOpen(fileReferenceGDS)

## Path to demo Profile GDS file
fileProfileGDS  <- file.path(dataDir, "GDS_Sample_with_study_demo.gds")

## Returns 0L when all parameters are valid
RAIDS:::validateAddStudy1Kg(gdsReference=gds1KG,
    fileProfileGDS=fileProfileGDS, verbose=TRUE)
```

```
## All GDS file must be closed
closefn.gds(gdsfile=gds1KG)
```

---

validateCharacterString *Validate that the input parameter is a character sting*

Description

This function validates that the input parameter is a character string (vector of 1 entry). If the parameter is not a character string, the function generates an error with a specific message.

Usage

```
validateCharacterString(value, name)
```

Arguments

value	a character string that will be validated.
name	a character string that represents the name of the parameter that is tested.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## The validation should be successful
RAIDS:::validateCharacterString(value="hi", name="test")
```

validateComputeAncestryFromSyntheticFile *Validate input parameters for computeAncestryFromSyntheticFile() function*

Description

This function validates the input parameters for the computeAncestryFromSyntheticFile function.

Usage

```
validateComputeAncestryFromSyntheticFile(
  gdsReference,
  gdsProfile,
  listFiles,
  currentProfile,
  spRef,
  studyIDSyn,
  np,
  listCatPop,
  fieldPopIn1KG,
  fieldPopInfAnc,
  kList,
  pcaList,
  algorithm,
  eigenCount,
  missingRate,
  verbose
)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the opened Population Reference GDS file.
gdsProfile	an object of class gds.class (a GDS file), the opened Profile GDS file.
listFiles	a vector of character strings representing the name of files that contain the results of ancestry inference done on the synthetic profiles for multiple values of D and K. The files must exist.
currentProfile	a character string representing the profile identifier of the current profile on which ancestry will be inferred.
spRef	a vector of character strings representing the known super population ancestry for the 1KG profiles. The Population Reference profile identifiers are used as names for the vector.
studyIDSyn	a character string corresponding to the study identifier. The study identifier must be present in the GDS Sample file.

np	a single positive integer representing the number of threads.
listCatPop	a vector of character string representing the list of possible ancestry assignations.
fieldPopIn1KG	a character string representing the name of the column that contains the known ancestry for the reference profiles in the Reference GDS file.
fieldPopInfAnc	a character string representing the name of the column that will contain the inferred ancestry for the specified dataset. Default: "SuperPop".
kList	a vector of integer representing the list of values tested for the *K* parameter. The *K* parameter represents the number of neighbors used in the K-nearest neighbor analysis.
pcaList	a vector of integer representing the list of values tested for the *D* parameter. The *D* parameter represents the number of dimensions used in the PCA analysis.
algorithm	a character string representing the algorithm used to calculate the PCA.
eigenCount	a single integer indicating the number of eigenvectors that will be in the output of the snpgdsPCA function; if 'eigenCount' <= 0, then all eigenvectors are returned.
missingRate	a numeric value representing the threshold missing rate at with the SNVs are discarded; the SNVs are retained in the snpgdsPCA with "<= missingRate" only; if NaN, no missing threshold.
verbose	a logical indicating if messages should be printed to show how the different steps in the function.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Required library
library(gdsfmt)

## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The 1KG Population Reference GDS demo file (opened)
gds1KG <- openfn.gds(file.path(dataDir,
            "PopulationReferenceDemo.gds"), readonly=TRUE)
```

```
## The Profile GDS (opened)
gdsSample <- openfn.gds(file.path(dataDir,
                    "GDS_Sample_with_study_demo.gds"), readonly=TRUE)

listFiles <- file.path(dataDir,  "listSNPIndexes_Demo.rds")

## The validation should be successful
RAIDS:::validateComputeAncestryFromSyntheticFile(gdsReference=gds1KG,
    gdsProfile=gdsSample, listFiles=listFiles, currentProfile="sample01",
    spRef=c("EUR", "AFR"), studyIDSyn="Synthetic", np=1L,
    listCatPop=c("EAS", "EUR", "AFR", "AMR", "SAS"),
    fieldPopIn1KG="superpop", fieldPopInfAnc="Superpop", kList=c(2, 3, 4),
    pcaList=c(3, 4, 5), algorithm="exact", eigenCount=32L, missingRate=0.2,
    verbose=FALSE)

## All GDS file must be closed
closefn.gds(gdsfile=gds1KG)
closefn.gds(gdsfile=gdsSample)
```

`validateComputeKNNRefSample` *Validate the input parameters for computeKNNRefSample() function*

Description

The function validates the input parameters for the `computeKNNRefSample` function. When a parameter is not as expected, an error message is generated.

Usage

```
validateComputeKNNRefSample(
  listEigenvector,
  listCatPop,
  spRef,
  fieldPopInfAnc,
  kList,
  pcaList
)
```

Arguments

`listEigenvector`
: a `list` with 3 entries: 'sample.id', 'eigenvector.ref' and 'eigenvector'. The `list` represents the PCA done on the 1KG reference profiles and the specific profile projected onto it. The 'sample.id' entry must contain only one identifier (one profile).

listCatPop	a vector of character string representing the list of possible ancestry assignations.
spRef	vector of character strings representing the known super population ancestry for the 1KG profiles. The 1KG profile identifiers are used as names for the vector.
fieldPopInfAnc	a character string representing the name of the column that will contain the inferred ancestry for the specified dataset.
kList	a vector of integer representing the list of values tested for the K parameter. The K parameter represents the number of neighbors used in the K-nearest neighbors analysis.
pcaList	a vector of integer representing the list of values tested for the D parameter. The D parameter represents the number of dimensions used in the PCA analysis.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Load the demo PCA on the synthetic profiles projected on the
## demo 1KG reference PCA
data(demoPCASyntheticProfiles)

## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

pcaSynthetic <- demoPCASyntheticProfiles
pcaSynthetic$sample.id <- pcaSynthetic$sample.id[1]

## The function returns 0L when all parameters are valid
RAIDS:::validateComputeKNNRefSample(listEigenvector=pcaSynthetic,
    listCatPop=c("EAS", "EUR", "AFR", "AMR", "SAS"),
    spRef=demoKnownSuperPop1KG, fieldPopInfAnc="Superpop",
    kList=c(10, 11, 12), pcaList=c(13, 14, 15))
```

---

validateComputeKNNRefSynthetic

Validate the input parameters for computeKNNRefSynthetic() function

---

Description

The function validates the input parameters for the computeKNNRefSynthetic function. When a parameter is not as expected, an error message is generated.

Usage

```
validateComputeKNNRefSynthetic(
  gdsProfile,
  listEigenvector,
  listCatPop,
  studyIDSyn,
  spRef,
  fieldPopInfAnc,
  kList,
  pcaList
)
```

Arguments

gdsProfile	an object of class SNPRelate::SNPGDSFileClass, the opened Profile GDS file.
listEigenvector	a list with 3 entries: 'sample.id', 'eigenvector.ref' and 'eigenvector'. The list represents the PCA done on the 1KG reference profiles and the synthetic profiles projected onto it.
listCatPop	a vector of character string representing the list of possible ancestry assignations.
studyIDSyn	a character string corresponding to the study identifier. The study identifier must be present in the Profile GDS file.
spRef	vector of character strings representing the known super population ancestry for the 1KG profiles. The 1KG profile identifiers are used as names for the vector.
fieldPopInfAnc	a character string representing the name of the column that will contain the inferred ancestry for the specified dataset.
kList	a vector of integer representing the list of values tested for the K parameter. The K parameter represents the number of neighbors used in the K-nearest neighbors analysis.
pcaList	a vector of integer representing the list of values tested for the D parameter. The D parameter represents the number of dimensions used in the PCA analysis.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Load the demo PCA on the synthetic profiles projected on the
## demo 1KG reference PCA
data(demoPCASyntheticProfiles)
```

```
## Load the known ancestry for the demo 1KG reference profiles
data(demoKnownSuperPop1KG)

## Path to the demo GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")
fileProfileGDS <- file.path(dataDir, "ex1.gds")

## Open GDS files
gdsProfile <- openfn.gds(fileProfileGDS)

## The function returns 0L when all parameters are valid
RAIDS:::validateComputeKNNRefSynthetic(gdsProfile=gdsProfile,
    listEigenvector=demoPCASyntheticProfiles,
    listCatPop=c("EAS", "EUR", "AFR", "AMR", "SAS"),
    studyIDSyn="MyStudy", spRef=demoKnownSuperPop1KG,
    fieldPopInfAnc="Superpop", kList=c(10, 11, 12),
    pcaList=c(13, 14, 15))

## Close GDS file (it is important to always close the GDS files)
closefn.gds(gdsProfile)
```

`validateComputePCAMultiSynthetic` *Validate the input parameters for computePCAMultiSynthetic() function*

Description

The function validates the input parameters for the `computePCAMultiSynthetic` function. When a parameter is not as expected, an error message is generated.

Usage

```
validateComputePCAMultiSynthetic(
  gdsProfile,
  listPCA,
  sampleRef,
  studyIDSyn,
  verbose
)
```

Arguments

`gdsProfile`	an object of class gds.class (a GDS file), an opened Profile GDS file.
`listPCA`	a `list` containing the PCA `object` generated with the 1KG reference profiles (excluding the ones used to generate the synthetic data set) in an entry called `"pca.unrel"`.

sampleRef	a vector of character strings representing the identifiers of the 1KG reference profiles that should not be used to create the reference PCA.
studyIDSyn	a character string corresponding to the study identifier. The study identifier must be present in the Profile GDS file.
verbose	a logical indicating if messages should be printed to show how the different steps in the function.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Loading demo PCA on subset of 1KG reference dataset
data(demoPCA1KG)

## Path to the demo GDS file is located in this package
dataDir <- system.file("extdata/demoKNNSynthetic", package="RAIDS")
fileProfileGDS <- file.path(dataDir, "ex1.gds")

## Open GDS files
gdsProfile <- openfn.gds(fileProfileGDS)

## The function returns 0L when all parameters are valid
RAIDS:::validateComputePCAMultiSynthetic(gdsProfile=gdsProfile,
    listPCA=demoPCA1KG, sampleRef=c("HG00246", "HG00325"),
    studyIDSyn="MyStudy", verbose=FALSE)

## Close GDS file (it is important to always close the GDS files)
closefn.gds(gdsProfile)
```

---

validateComputePCARefSample

Validate input parameters for computePCARefSample() function

Description

This function validates the input parameters for the computePCARefSample function.

Usage

```
validateComputePCARefSample(
  gdsProfile,
  currentProfile,
  studyIDRef,
  np,
  algorithm,
  eigenCount,
  missingRate,
  verbose
)
```

Arguments

gdsProfile	an object of class gds.class, an opened Profile GDS file.
currentProfile	a single character string representing the profile identifier.
studyIDRef	a single character string representing the study identifier.
np	a single positive integer representing the number of CPU that will be used.
algorithm	a character string representing the algorithm used to calculate the PCA.
eigenCount	a single integer indicating the number of eigenvectors that will be in the output of the snpgdsPCA function; if 'eigen.cnt' <= 0, then all eigenvectors are returned.
missingRate	a numeric value representing the threshold missing rate at with the SNVs are discarded; if NaN, no missing threshold.
verbose	a logical indicating if messages should be printed to show how the different steps in the function.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The GDS Sample (opened)
gdsSample <- openfn.gds(file.path(dataDir,
                    "GDS_Sample_with_study_demo.gds"), readonly=TRUE)

## The validation should be successful
```

```
RAIDS:::validateComputePCARefSample(gdsProfile=gdsSample,
    currentProfile="HCC01", studyIDRef="1KG", np=1L, algorithm="exact",
    eigenCount=32L, missingRate=0.02, verbose=FALSE)

## All GDS file must be closed
closefn.gds(gdsfile=gdsSample)
```

`validateComputePoolSyntheticAncestryGr` *Validate input parameters for computePoolSyntheticAncestryGr() function*

Description

This function validates the input parameters for the `computePoolSyntheticAncestryGr` function.

Usage

```
validateComputePoolSyntheticAncestryGr(
  gdsProfile,
  sampleRM,
  spRef,
  studyIDSyn,
  np,
  listCatPop,
  fieldPopInfAnc,
  kList,
  pcaList,
  algorithm,
  eigenCount,
  missingRate,
  verbose
)
```

Arguments

`gdsProfile`	an object of class `SNPRelate::SNPGDSFileClass`, the opened Profile GDS file.
`sampleRM`	a `vector` of `character` strings representing the identifiers of the population reference samples that should not be used to create the reference PCA.
`spRef`	a `vector` of `character` strings representing the known super population ancestry for the population reference profiles. The population reference profile identifiers are used as names for the `vector`.
`studyIDSyn`	a `character` string corresponding to the study identifier. The study identifier must be present in the GDS Sample file.
`np`	a single positive `integer` representing the number of threads.

listCatPop	a vector of character string representing the list of possible ancestry assignations.
fieldPopInfAnc	a character string representing the name of the column that will contain the inferred ancestry for the specified dataset.
kList	a vector of integer representing the list of values tested for the *K* parameter. The *K* parameter represents the number of neighbors used in the K-nearest neighbor analysis.
pcaList	a vector of integer representing the list of values tested for the *D* parameter. The *D* parameter represents the number of dimensions used in the PCA analysis.
algorithm	a character string representing the algorithm used to calculate the PCA. The 2 choices are "exact" (traditional exact calculation) and "randomized" (fast PCA with randomized algorithm introduced in Galinsky et al. 2016).
eigenCount	a single integer indicating the number of eigenvectors that will be in the output of the snpgdsPCA function; if 'eigenCount' <= 0, then all eigenvectors are returned.
missingRate	a numeric value representing the threshold missing rate at with the SNVs are discarded; the SNVs are retained in the snpgdsPCA only with "<= missingRate" only; if NaN, no missing threshold.
verbose	a logical indicating if message information should be printed.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

References

Galinsky KJ, Bhatia G, Loh PR, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. Am J Hum Genet. 2016 Mar 3;98(3):456-72. doi: 10.1016/j.ajhg.2015.12.022. Epub 2016 Feb 25.

Examples

```
## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The Profile GDS (opened)
gdsSample <- openfn.gds(file.path(dataDir,
                    "GDS_Sample_with_study_demo.gds"), readonly=TRUE)

## The known super population ancestry for the population reference profiles
spRef <- c("EUR", "SAS", "EAS", "EUR", "AFR")
names(spRef) <- c("HG00100", "HG00101", "HG00102", "HG00103", "HG00104")

## The validation should be successful
```

```
RAIDS:::validateComputePoolSyntheticAncestryGr(gdsProfile=gdsSample,
     sampleRM="TGCA_01", spRef=spRef,
     studyIDSyn="TCGA", np=1L, listCatPop=c("AFR", "EAS", "SAS"),
     fieldPopInfAnc="Pop", kList=seq_len(3),
     pcaList=seq_len(10), algorithm="exact", eigenCount=12L,
     missingRate=0.02, verbose=FALSE)

## All GDS file must be closed
closefn.gds(gdsfile=gdsSample)
```

`validateComputeSyntheticRoc` *Validate input parameters for computeSyntheticROC() function*

Description

This function validates the input parameters for the `computeSyntheticROC()` function.

Usage

```
validateComputeSyntheticRoc(
  matKNN,
  matKNNAncestryColumn,
  pedCall,
  pedCallAncestryColumn,
  listCall
)
```

Arguments

`matKNN`	a `data.frame` containing the inferred ancestry results for fixed values of D and K. On of the column names of the `data.frame` must correspond to the `matKNNAncestryColumn` argument.
`matKNNAncestryColumn`	a `character` string representing the name of the column that contains the inferred ancestry for the specified synthetic profiles. The column must be present in the `matKNN` argument.
`pedCall`	a `data.frame` containing the information about the super-population information from the 1KG GDS file for profiles used to generate the synthetic profiles. The `data.frame` must contained a column named as the `pedCallAncestryColumn` argument.
`pedCallAncestryColumn`	a `character` string representing the name of the column that contains the known ancestry for the reference profiles in the Reference GDS file. The column must be present in the `pedCall` argument.
`listCall`	a `vector` of `character` strings representing the list of all possible ancestry assignations.

Value

`0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Loading demo dataset containing pedigree information for synthetic
## profiles and known ancestry of the profiles used to generate the
## synthetic profiles
data(pedSynthetic)

## Loading demo dataset containing the inferred ancestry results
## for the synthetic data
data(matKNNSynthetic)

## The inferred ancestry results for the synthetic data using
## values of D=6 and K=5
matKNN <- matKNNSynthetic[matKNNSynthetic$K == 6 & matKNNSynthetic$D == 5, ]

## The validation should be successful
RAIDS:::validateComputeSyntheticRoc(matKNN=matKNN,
    matKNNAncestryColumn="SuperPop",
    pedCall=pedSynthetic, pedCallAncestryColumn="superPop",
    listCall=c("EAS", "EUR", "AFR", "AMR", "SAS"))
```

---

validateCreateStudy2GDS1KG

Validate input parameters for createStudy2GDS1KG() function

Description

This function validates the input parameters for the `createStudy2GDS1KG` function.

Usage

```
validateCreateStudy2GDS1KG(
  pathGeno,
  pedStudy,
  fileNameGDS,
  batch,
  studyDF,
  listProfiles,
  pathProfileGDS,
```

```
  genoSource,
  verbose
)
```

Arguments

pathGeno	a character string representing the path to the directory containing the VCF output of SNP-pileup for each sample. The SNP-pileup files must be compressed (gz files) and have the name identifiers of the samples. A sample with "Name.ID" identifier would have an associated SNP-pileup file called "Name.ID.txt.gz". The directory must exist.
pedStudy	a data.frame with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in character strings (no factor). The data.frame must contain the information for all the samples passed in the listSamples parameter. Only filePedRDS or pedStudy can be defined.
fileNameGDS	a character string representing the file name of the Population Reference GDS file. The file must exist.
batch	a single positive integer representing the current identifier for the batch. Beware, this field is not stored anymore.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).
listProfiles	a vector of character string corresponding to the profile identifiers that will have a GDS Sample file created. The profile identifiers must be present in the "Name.ID" column of the RDS file passed to the filePedRDS parameter. If NULL, all profiles in the filePedRDS are selected.
pathProfileGDS	a character string representing the path to the directory where the Profile GDS files will be created.
verbose	a logical indicating if message information should be printed.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Demo 1KG Population Reference GDS file
gds1KG <- file.path(dataDir, "PopulationReferenceDemo.gds")

## The data.frame containing the information about the study
```

```
## The 3 mandatory columns: "study.id", "study.desc", "study.platform"
## The entries should be strings, not factors (stringsAsFactors=FALSE)
studyInfo <- data.frame(study.id="Pancreatic.WES",
                study.desc="Pancreatic study",
                study.platform="WES",
                stringsAsFactors=FALSE)

## PED Study
ped <- data.frame(Name.ID=c("Sample_01", "Sample_02"),
            Case.ID=c("TCGA-H01", "TCGA-H02"),
            Sample.Type=c("DNA", "DNA"),
            Diagnosis=c("Cancer", "Cancer"), Source=c("TCGA", "TCGA"))

## The validation should be successful
RAIDS:::validateCreateStudy2GDS1KG(pathGeno=dataDir, pedStudy=ped,
            fileNameGDS=gds1KG, batch=1, studyDF=studyInfo,
            listProfiles=c("Sample_01", "Sample_02"),
            pathProfileGDS=dataDir,
            genoSource="snp-pileup", verbose=TRUE)
```

`validateDataRefSynParameter`

Validate that the reference profile data set has the mandatory columns

---

Description

The function validates the input reference profile data set. The reference profile data set must be a `data.frame` with those mandatory columns: "sample.id", "pop.group", "superPop". All columns must be in `character` strings (no factor).

Usage

```
validateDataRefSynParameter(syntheticRefDF)
```

Arguments

`syntheticRefDF`	a `data.frame` containing a subset of reference profiles for each sub-population present in the Reference GDS file. The mandatory columns are: "sample.id", "pop.group", "superPop". All columns must be in `character` strings (no factor).

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Profiles used for synthetic data set
syntheticRefDF <- data.frame(sample.id=c("HG00150", "HG00138", "HG00330",
    "HG00275"), pop.group=c("GBR", "GBR","FIN", "FIN"),
    superPop=c("EUR", "EUR", "EUR", "EUR"), stringsAsFactors=FALSE)

## Return 0L when the reference profile data set is valid
RAIDS:::validateDataRefSynParameter(syntheticRefDF=syntheticRefDF)
```

`validateEstimateAllelicFraction` *Validate input parameters for estimateAllelicFraction() function*

Description

This function validates the input parameters for the `estimateAllelicFraction` function.

Usage

```
validateEstimateAllelicFraction(
  gdsReference,
  gdsProfile,
  currentProfile,
  studyID,
  chrInfo,
  studyType,
  minCov,
  minProb,
  eProb,
  cutOffLOH,
  cutOffHomoScore,
  wAR,
  cutOffAR,
  gdsRefAnnot,
  blockID,
  verbose
)
```

Arguments

`gdsReference`	an object of class `gds.class` (a GDS file), the Population Reference GDS file.
`gdsProfile`	an object of class `gds.class` (a GDS file), the Profile GDS file.
`currentProfile`	a `character` string corresponding to the sample identifier as used in `pruningSample` function.
`studyID`	a `character` string corresponding to the name of the study as used in `pruningSample` function.

chrInfo	a vector of integer values representing the length of the chromosomes.
studyType	a character string representing the type of study. The possible choices are: "DNA" and "RNA". The type of study affects the way the estimation of the allelic fraction is done. Default: "DNA".
minCov	a single positive integer representing the minimum required coverage.
minProb	a single positive numeric between 0 and 1 that represents the probability that the genotype is correct.
eProb	a single numeric between 0 and 1 representing the probability of sequencing error.
cutOffLOH	a single numeric representing the cutoff, in log, for the homozygote score to assign a region as LOH.
cutOffHomoScore	a single numeric representing the cutoff, in log, that the SNVs in a block are called homozygote by error.
wAR	a single positive integer representing the size-1 of the window used to compute an empty box.
cutOffAR	a single numeric representing the cutoff, in log score, that the SNVs in a gene are allelic fraction different from 0.5.
gdsRefAnnot	an object of class gds.class (a GDS file), the1 Population Reference SNV Annotation GDS file. This parameter is RNA specific.
blockID	a character string corresponding to the block identifier in gdsRefAnnot. This parameter is RNA specific.
verbose	a logicial indicating if the function should print message when running.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The 1KG Population Reference GDS Demo file (opened)
gds1KG <- openfn.gds(file.path(dataDir,
                "PopulationReferenceDemo.gds"), readonly=TRUE)

## The GDS Sample (opened)
gdsSample <- openfn.gds(file.path(dataDir,
```

```
                    "GDS_Sample_with_study_demo.gds"), readonly=TRUE)

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

    ## The validation should be successful
    RAIDS:::validateEstimateAllelicFraction(gdsReference=gds1KG,
        gdsProfile=gdsSample,
        currentProfile="Sample01", studyID="Synthetic", chrInfo=chrInfo,
        studyType="DNA", minCov=10L, minProb=0.03, eProb=0.002, cutOffLOH=10,
        cutOffHomoScore=11, wAR=2, cutOffAR=10, gdsRefAnnot=gds1KG,
        blockID="1", verbose=FALSE)

    ## All GDS file must be closed
    closefn.gds(gdsfile=gds1KG)
    closefn.gds(gdsfile=gdsSample)

}
```

`validateGDSClass` *Validate that the input parameter is a GDS object*

Description

This function validates that the input parameter inherits the gds.class class.

Usage

```
validateGDSClass(gds, name)
```

Arguments

`gds`	an object of class gds.class (a GDS file), the 1 KG GDS file.
`name`	a `character` string that represents the name of the parameter that is tested.

Value

The function returns `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The 1KG GDS file (opened)
gds1KG <- openfn.gds(file.path(dataDir,
                "PopulationReferenceDemo.gds"), readonly=TRUE)

## The validation should be successful
RAIDS:::validateGDSClass(gds=gds1KG, name="gds")

## All GDS file must be closed
closefn.gds(gdsfile=gds1KG)
```

`validateGenerateGDS1KG` *Validate input parameters for generateGDS1KG() function*

Description

This function validates the input parameters for the `generateGDS1KG` function.

Usage

```
validateGenerateGDS1KG(
  pathGeno,
  filePedRDS,
  fileSNVIndex,
  fileSNVSelected,
  fileNameGDS,
  listSamples,
  verbose
)
```

Arguments

`pathGeno`	a `character` string representing the path where the 1K genotyping files for each sample are located. The name of the genotyping files must correspond to the individual identification (Individual.ID) in the pedigree file.
`filePedRDS`	a `character` string representing the path and file name of the RDS file that contains the pedigree information. The file must exist. The file must be a RDS file.
`fileSNVIndex`	a `character` string representing the path and file name of the RDS file that contains the indexes of the retained SNPs. The file must exist. The file must be a RDS file.

`fileSNVSelected`
: a `character` string representing the path and file name of the RDS file that contains the filtered SNP information. The file must exist. The file must be a RDS file.

`fileNameGDS`
: a `character` string representing the path and file name of the GDS file that will be created. The GDS file will contain the SNP information, the genotyping information and the pedigree information from 1000 Genomes. The extension of the file must be '.gds'.

`listSamples`
: a `vector` of `character` string corresponding to samples (must be the sample.ids) that will be retained and added to the GDS file. When `NULL`, all the samples are retained.

`verbose`
: a `logical` indicating if the function must print messages when running.

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## The RDS file containing the pedigree information
pedigreeFile <- file.path(dataDir, "PedigreeReferenceDemo.rds")

## The RDS file containing the indexes of the retained SNPs
snpIndexFile <- file.path(dataDir, "listSNPIndexes_Demo.rds")

## The RDS file containing the filtered SNP information
filterSNVFile <- file.path(dataDir, "mapSNVSelected_Demo.rds")

## Temporary GDS file containing 1KG information
gdsFile <- file.path(dataDir, "1KG_TEMP.gds")

## The validation should be successful
RAIDS:::validateGenerateGDS1KG(pathGeno=dataDir, filePedRDS=pedigreeFile,
    fileSNVIndex=snpIndexFile, fileSNVSelected=filterSNVFile,
    fileNameGDS=gdsFile, listSamples=NULL, verbose=FALSE)
```

`validateLogical` *Validate that the input parameter is a logical*

Description

This function validates that the input parameter is a logical. If the parameter is not a logicial, the function generates an error with a specific message.

Usage

```
validateLogical(logical, name)
```

Arguments

`logical`	a `logical` that will be validated.
`name`	a `character` string that represents the name of the parameter that is tested.

Value

The function returns `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## The validation should be successful
RAIDS:::validateLogical(logical=TRUE, name="test")
```

`validatePEDStudyParameter`
Validate that the PED study has the mandatory columns

Description

The function validates the input PED study. The PED study must be a `data.frame` with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in `character` strings (no factor).

Usage

```
validatePEDStudyParameter(pedStudy)
```

Arguments

pedStudy	a data.frame with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in character strings (no factor).

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Pedigree Study data frame
ped <- data.frame(Name.ID=c("Sample_01", "Sample_02"),
            Case.ID=c("TCGA-H01", "TCGA-H02"),
            Sample.Type=c("DNA", "DNA"),
            Diagnosis=c("Cancer", "Cancer"), Source=c("TCGA", "TCGA"))

## Return 0L when PED is valid
RAIDS:::validatePEDStudyParameter(pedStudy=ped)
```

---

validatePepSynthetic *Validate input parameters for prepSynthetic() function*

Description

This function validates the input parameters for the prepSynthetic() function.

Usage

```
validatePepSynthetic(
  fileProfileGDS,
  listSampleRef,
  profileID,
  studyDF,
  nbSim,
  prefix,
  verbose
)
```

Arguments

fileProfileGDS	a character string representing the file name of the GDS Sample file containing the information about the sample used to generate the synthetic profiles.
listSampleRef	a vector of character string representing the identifiers of the selected 1KG samples that will be used as reference to generate the synthetic profiles.
profileID	a character string representing the profile identifier present in the fileProfileGDS that will be used to generate synthetic profiles.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 2 columns: "study.id" and "study.desc". Those 2 columns must be in character strings (no factor). Other columns can be present, such as "study.platform", but won't be used.
nbSim	a single positive integer representing the number of simulations per combination of sample and 1KG reference.
prefix	a single character string representing the prefix that is going to be added to the name of the synthetic profile. The prefix enables the creation of multiple synthetic profile using the same combination of sample and 1KG reference.
verbose	a logical indicating if messages should be printed to show how the different steps in the function.

Value

0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The Profile GDS Sample
gdsSample <- file.path(dataDir, "GDS_Sample_with_study_demo.gds")

## The study data frame
studyDF <- data.frame(study.id="MYDATA.Synthetic",
    study.desc="MYDATA synthetic data", study.platform="PLATFORM",
    stringsAsFactors=FALSE)

## The validation should be successful
RAIDS:::validatePepSynthetic(fileProfileGDS=gdsSample,
     listSampleRef=c("Sample01", "Sample02"), profileID="A101TCGA",
     studyDF=studyDF, nbSim=1L, prefix="TCGA", verbose=TRUE)
```

validatePositiveIntegerVector *Validate that the input parameter is a vector of positive* numeric

Description

This function validates that the input parameter is a `vector` of positive `numeric` values (vector of 1 entry or more). All values have to be positive (`>0`). If the parameter is not respecting the validation, the function generates an error with a specific message.

Usage

```
validatePositiveIntegerVector(value, name)
```

Arguments

`value`	a `vector` of `numeric` that will be validated.
`name`	a `character` string that represents the name of the parameter that is tested.

Value

The function returns `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## The validation should be successful
RAIDS:::validatePositiveIntegerVector(value=c(1, 2 ,3, 5), name="test")
```

validatePrepPed1KG *Validate input parameters for prepPed1KG() function*

Description

This function validates the input parameters for the `prepPed1KG` function.

Usage

```
validatePrepPed1KG(filePed, pathGeno, batch)
```

Arguments

filePed	a character string representing the path and file name of the pedigree file (PED file) that contains the information related to the profiles present in the 1KG GDS file. The PED file must exist.
pathGeno	a character string representing the path where the Reference genotyping files for each profile are located. Only the profiles with associated genotyping files are retained in the creation of the final data.frame. The name of the genotyping files must correspond to the individual identification (Individual.ID) in the pedigree file (PED file).
batch	ainteger that uniquely identifies the source of the pedigree information. The Reference is usually 0L.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## Demo pedigree file
pedDemoFile <- file.path(dataDir, "PedigreeDemo.ped")

## The validation should be successful
RAIDS:::validatePrepPed1KG(filePed=pedDemoFile,
     pathGeno=dataDir, batch=1)
```

---

validateProfileGDSExist

Validate that the Profile GDS file exists for the specified profile

---

Description

The function validates that the Profile GDS file associated to a profile identifier exists in the specified directory.

Usage

```
validateProfileGDSExist(pathProfile, profile)
```

Arguments

pathProfile	a character string representing the directory where the Profile GDS files will be created. The directory must exist.
currentProfile	a character string corresponding to the profile identifier. A Profile GDS file corresponding to the profile identifier must exist and be located in the pathProfile directory.

Value

a character string representing the path to the existing Profile GDS file.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo 1KG GDS file is located in this package
dataDir <- system.file("extdata/tests", package="RAIDS")

## The function returns the path to the existing Profile GDS file
RAIDS:::validateProfileGDSExist(pathProfile=dataDir,
    profile="ex1_demo")
```

---

validatePruningSample *Validate input parameters for pruningSample() function*

Description

This function validates the input parameters for the pruningSample function.

Usage

```
validatePruningSample(
  gdsReference,
  method,
  currentProfile,
  studyID,
  listSNP,
  slideWindowMaxBP,
  thresholdLD,
  np,
  verbose,
  chr,
  superPopMinAF,
  keepPrunedGDS,
```

```
  pathProfileGDS,
  keepFile,
  pathPrunedGDS,
  outPrefix
)
```

Arguments

gdsReference	an object of class gds.class (a GDS file), the Population Reference GDS file.
method	a character string that represents the method that will be used to calculate the linkage disequilibrium in the snpgdsLDpruning() function.
currentProfile	a character string corresponding to the profile identifier used in LD pruning done by the snpgdsLDpruning() function. A Profile GDS file corresponding to the profile identifier must exist and be located in the pathProfileGDS directory.
studyID	a character string corresponding to the study identifier used in the snpgdsLDpruning function. The study identifier must be present in the Profile GDS file.
listSNP	a vector of SNVs identifiers specifying selected to be passed the the pruning function; if NULL, all SNVs are used in the snpgdsLDpruning function.
slideWindowMaxBP	a single positive integer that represents the maximum basepairs (bp) in the sliding window. This parameter is used for the LD pruning done in the snpgdsLDpruning function.
thresholdLD	a single numeric value that represents the LD threshold used in the snpgdsLDpruning function.
np	a single positive integer specifying the number of threads to be used.
verbose	a logicial indicating if information is shown during the process in the snpgdsLDpruning function.
chr	a character string representing the chromosome where the selected SNVs should belong. Only one chromosome can be handled. If NULL, the chromosome is not used as a filtering criterion.
superPopMinAF	a single positive numeric representing the minimum allelic frequency used to select the SNVs. If NULL, the allelic frequency is not used as a filtering criterion.
keepPrunedGDS	a logicial indicating if the information about the pruned SNVs should be added to the GDS Sample file.
pathProfileGDS	a character string representing the directory where the Profile GDS files will be created. The directory must exist.
keepFile	a logical indicating if RDS files containing the information about the pruned SNVs must be created.
pathPrunedGDS	a character string representing an existing directory. The directory must exist.
outPrefix	a character string that represents the prefix of the RDS files that will be generated. The RDS files are only generated when the parameter keepFile=TRUE.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Required library
library(gdsfmt)

## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The 1KG GDS file (opened)
gds1KG <- openfn.gds(file.path(dataDir,
            "PopulationReferenceDemo.gds"), readonly=TRUE)

## The validation should be successful
RAIDS:::validatePruningSample(gdsReference=gds1KG, method="corr",
     currentProfile="TGCA_01", studyID="TCGA",
     listSNP=c("sr10103", "sr10202"), slideWindowMaxBP=1000L,
     thresholdLD=0.008, np=1L, verbose=TRUE, chr=1,
     superPopMinAF=0.002, keepPrunedGDS=TRUE, pathProfileGDS=dataDir,
     keepFile=FALSE, pathPrunedGDS=dataDir, outPrefix="test")

## All GDS file must be closed
closefn.gds(gdsfile=gds1KG)
```

---

validateRunExomeOrRNAAncestry

Validate the parameters of the runExomeAncestry() function

Description

The function validates the input parameters for the `runExomeAncestry` function. When a parameter is not as expected, an error message is generated.

Usage

```
validateRunExomeOrRNAAncestry(
  pedStudy,
  studyDF,
  pathProfileGDS,
  pathGeno,
  pathOut,
  fileReferenceGDS,
  fileReferenceAnnotGDS,
  chrInfo,
  syntheticRefDF,
```

```
  genoSource,
  verbose
)
```

Arguments

pedStudy	a data.frame with those mandatory columns: "Name.ID", "Case.ID", "Sample.Type", "Diagnosis", "Source". All columns must be in character strings (no factor). The data.frame must contain the information for all the samples passed in the listSamples parameter. Only filePedRDS or pedStudy can be defined.
studyDF	a data.frame containing the information about the study associated to the analysed sample(s). The data.frame must have those 3 columns: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).
pathProfileGDS	a character string representing the path to the directory where the GDS Profile files will be created. Default: NULL.
pathGeno	a character string representing the path to the directory containing the VCF output of SNP-pileup for each sample. The SNP-pileup files must be compressed (gz files) and have the name identifiers of the samples. A sample with "Name.ID" identifier would have an associated SNP-pileup file called "Name.ID.txt.gz". The directory must exist.
pathOut	a character string representing the path to the directory where the output files are created.
fileReferenceGDS	a character string representing the file name of the Population Reference GDS file. The file must exist.
fileReferenceAnnotGDS	a character string representing the file name of the Population Reference GDS annotation file. The file must exist.
chrInfo	a vector of positive integer values representing the length of the chromosomes. See 'details' section.
syntheticRefDF	a data.frame containing those columns: • sample.id a character string representing the sample identifier. • pop.group a character string representing the subcontinental population assigned to the sample. • superPop a character string representing the super-population assigned to the sample.
verbose	a logical indicating if messages should be printed to show how the different steps in the function. Default: FALSE.

Value

The integer 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Path to the demo pedigree file is located in this package
dataDir <- system.file("extdata", package="RAIDS")

## Path where the output file will be generated
pathOut <- file.path(dataDir, "example", "res.out")

## Study data frame
study <- data.frame(study.id = "MYDATA",
                    study.desc = "Description",
                    study.platform = "PLATFORM",
                    stringsAsFactors = FALSE)

## Population Reference GDS demo file
gdsRef <- file.path(dataDir, "PopulationReferenceDemo.gds")

gdsAnnotRef <- file.path(dataDir, "PopulationReferenceSNVAnnotationDemo.gds")

## Pedigree Study data frame
ped <- data.frame(Name.ID=c("Sample_01", "Sample_02"),
            Case.ID=c("TCGA-H01", "TCGA-H02"),
            Sample.Type=c("DNA", "DNA"),
            Diagnosis=c("Cancer", "Cancer"), Source=c("TCGA", "TCGA"))

## Required library for this example to run correctly
if (requireNamespace("GenomeInfoDb", quietly=TRUE) &&
     requireNamespace("BSgenome.Hsapiens.UCSC.hg38", quietly=TRUE)) {

    ## Chromosome length information
    ## chr23 is chrX, chr24 is chrY and chrM is 25
    chrInfo <- GenomeInfoDb::seqlengths(BSgenome.Hsapiens.UCSC.hg38::Hsapiens)[1:25]

    ## Profiles used for synthetic data set
    syntheticRefDF <- data.frame(sample.id=c("HG00150", "HG00138", "HG00330",
        "HG00275"), pop.group=c("GBR", "GBR","FIN", "FIN"),
        superPop=c("EUR", "EUR", "EUR", "EUR"), stringsAsFactors=FALSE)

    ## Returns OL when all parameters are valid
    RAIDS:::validateRunExomeOrRNAAncestry(pedStudy=ped, studyDF=study,
        pathProfileGDS=dataDir, pathGeno=dataDir, pathOut=pathOut,
        fileReferenceGDS=gdsRef, fileReferenceAnnotGDS=gdsAnnotRef,
        chrInfo=chrInfo, syntheticRefDF=syntheticRefDF,
        genoSource="snp-pileup", verbose=FALSE)

}
```

`validateSingleRatio` *Validate that the input parameter is a single positive numeric between zero and one (included)*

Description

This function validates that the input parameter is a single numeric between zero and one (included). If the parameter is not, the function generates an error with a specific message.

Usage

```
validateSingleRatio(value, name)
```

Arguments

value	a single positive numeric that will be validated.
name	a character string that represents the name of the parameter that is tested.

Value

The function returns 0L when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## The validation should be successful
RAIDS:::validateSingleRatio(value=0.02, name="test")
```

---

validateStudyDataFrameParameter

Validate that the study data set has the mandatory columns

Description

The function validates the input dtudy data set. The study data set must be a data.frame with those mandatory columns: "studyID", "study.desc", "study.platform". All columns must be in character strings (no factor).

Usage

```
validateStudyDataFrameParameter(studyDF)
```

Arguments

studyDF	a data.frame containing the study information. The mandatory columns are: "study.id", "study.desc", "study.platform". All columns must be in character strings (no factor).

Value

The integer `0L` when successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Study data frame
study <- data.frame(study.id = "MYDATA",
                    study.desc = "Description",
                    study.platform = "PLATFORM",
                    stringsAsFactors = FALSE)

## Return 0L when the study data set is valid
RAIDS:::validateStudyDataFrameParameter(studyDF=study)
```

`validateSyntheticGeno` *Validate input parameters for syntheticGeno() function*

Description

This function validates the input parameters for the `syntheticGeno()` function.

Usage

```
validateSyntheticGeno(
  gdsReference,
  gdsRefAnnot,
  fileProfileGDS,
  profileID,
  listSampleRef,
  nbSim,
  prefix,
  pRecomb,
  minProb,
  seqError
)
```

Arguments

`gdsReference`	an object of class `gds.class` (a GDS file), the 1KG GDS file.
`gdsRefAnnot`	an object of class `gds.class` (a GDS file), the1 1KG SNV Annotation GDS file.
`fileProfileGDS`	a `character` string representing the file name of the GDS Sample file containing the information about the sample. The file must exist.

profileID	a character string representing the unique identifier of the cancer sample.
listSampleRef	a vector of character strings representing the sample identifiers of the 1KG selected reference samples.
nbSim	a single positive integer representing the number of simulations that will be generated per sample + 1KG reference combination.
prefix	a character string that represent the prefix that will be added to the name of the synthetic profiles generated by the function.
pRecomb	a single positive numeric between 0 and 1 that represents the frequency of phase switching in the synthetic profiles.
minProb	a single positive numeric between 0 and 1 that represents the probability that the genotype is correct.
seqError	a single positive numeric between 0 and 1 representing the sequencing error rate.

Value

The integer 0L when the function is successful.

Author(s)

Pascal Belleau, Astrid Deschênes and Alexander Krasnitz

Examples

```
## Directory where demo GDS files are located
dataDir <- system.file("extdata", package="RAIDS")

## The 1KG GDS file (opened)
gdsRef <- openfn.gds(file.path(dataDir,
                "PopulationReferenceDemo.gds"), readonly=TRUE)

## The 1KG GDS Annotation file (opened)
gdsRefAnnot <- openfn.gds(file.path(dataDir,
    "PopulationReferenceSNVAnnotationDemo.gds"), readonly=TRUE)

## The GDS Sample file
gdsSample <- file.path(dataDir, "GDS_Sample_with_study_demo.gds")

## The validation should be successful
RAIDS:::validateSyntheticGeno(gdsReference=gdsRef, gdsRefAnnot=gdsRefAnnot,
     fileProfileGDS=gdsSample, profileID="A101TCGA",
     listSampleRef="A101TCGA", nbSim=1L, prefix="TCGA", pRecomb=0.02,
     minProb=0.999, seqError=0.002)

## All GDS file must be closed
closefn.gds(gdsfile=gdsRef)
closefn.gds(gdsfile=gdsRefAnnot)
```

Index