Package 'sesame'

January 16, 2022

Type Package

Title SEnsible Step-wise Analysis of DNA MEthylation BeadChips

Description
Tools For analyzing Illumina Infinium DNA methylation arrays.SeSAMe provides utilities to support analyses of multiple generations of Infinium DNA methylation BeadChips, including preprocessing, quality control, visualization and inference. SeSAMe features more accurate detection calling, intelligenet inference of ethnicity, sex and advanced quality control routines.

Version 1.12.7

Depends R (>= 4.1), sesameData, methods

License MIT + file LICENSE

RoxygenNote 7.1.2

Imports BiocParallel, grDevices, utils, stringr, tibble, illuminaio, MASS, GenomicRanges, IRanges, grid, preprocessCore, S4Vectors, randomForest, wheatmap, ggplot2, graphics, KernSmooth, parallel, matrixStats, DNAcopy, stats, SummarizedExperiment, e1071, fgsea, ggrepel, reshape2

Suggests scales, knitr, rmarkdown, testthat, dplyr, tidyr, BiocStyle, IlluminaHumanMethylation450kmanifest, minfi, FlowSorted.CordBloodNorway.450k, FlowSorted.Blood.450k, HDF5Array

Encoding UTF-8

VignetteBuilder knitr

URL https://github.com/zwdzwd/sesame

BugReports https://github.com/zwdzwd/sesame/issues

biocViews DNAMethylation, MethylationArray, Preprocessing, QualityControl

Collate 'sex.R' 'species.R' 'QC.R' 'GEO.R' 'SigDFMethods.R' 'sesame.R' 'age.R' 'background_correction.R' 'cell_composition.R' 'channel_inference.R' 'cnv.R' 'ethnicity.R' 'deidentify.R' 'detection.R' 'dm.R' 'dye_bias.R' 'feature_selection.R' 'fileSet.R' 'mask.R' 'open.R' 'sesamize.R' 'snp.R' 'strain.R'

'tissue.R' 'track.R' 'utils.R' 'vcf.R' 'visualize.R' 'zzz.R'
'kyCG.R'

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

git_branch RELEASE_3_14

git_last_commit 1e3ec3a

git_last_commit_date 2021-12-13

Date/Publication 2022-01-16

Author Wanding Zhou [aut, cre],
Hui Shen [aut],
Timothy Triche [ctb],
Bret Barnes [ctb]

Maintainer Wanding Zhou <zhouwanding@gmail.com>

R topics documented:

sesame-package . . . 5
addMask . . . 5
as.data.frame.sesameQC . . . 6
attachManifest . . . 7
BetaValueToMValue . . . 7
binSignals . . . 8
bisConversionControl . . . 8
bSubComplete . . . 9
bSubMostVariable . . . 9
bSubProbes . . . 10
calcDatabaseSetStatistics1 . . . 11
calcDatabaseSetStatisticsAll . . . 11
checkLevels . . . 12
chipAddressToSignal . . . 13
cnSegmentation . . . 13
compareDatbaseSetOverlap . . . 14
compareMouseStrainReference . . . 15
compareMouseTissueReference . . . 16
controls . . . 16
createDatabaseSetNetwork . . . 17
createUCSCtrack . . . 18
deIdentify . . . 18
detectionPnegEcdf . . . 19
detectionPoobEcdf . . . 20
detectionPoobEcdf2 . . . 21
diffRefSet . . . 22
dmContrasts . . . 22
DML . . . 23
DMR . . . 24
dyeBiasCorr . . . 25
dyeBiasCorrMostBalanced . . . 26

dyeBiasCorrTypeINorm . . . 26
dyeBiasDistortion . . . 27
estimateCellComposition . . . 28
estimateLeukocyte . . . 28
extractDesign . . . 29
formatVCF . . . 30
getAFTypeIbySumAlleles . . . 31
getAutosomeProbes . . . 31
getBetas . . . 32
getBinCoordinates . . . 33
getDatabaseSetOverlap . . . 33
getNormCtls . . . 34
getProbesByChromosome . . . 35
getProbesByGene . . . 35
getProbesByRegion . . . 36
getProbesByTSS . . . 37
getRefSet . . . 38
getSexInfo . . . 38
inferEthnicity . . . 39
inferInfiniumIChannel . . . 40
inferSex . . . 40
inferSexKaryotypes . . . 41
inferSpecies . . . 42
inferStrain . . . 43
inferTissue . . . 43
initFileSet . . . 44
isUniqProbeID . . . 45
mapFileSet . . . 46
meanIntensity . . . 46
medianTotalIntensity . . . 47
MValueToBetaValue . . . 48
neob . . . 48
noMasked . . . 49
noob . . . 49
openSesame . . . 50
openSesameToFile . . . 51
plotLollipop . . . 51
plotVolcano . . . 52
predictAgeHorvath353 . . . 53
predictAgeSkinBlood . . . 53
predictMouseAgeInMonth . . . 54
print.DMLSummary . . . 55
print.fileSet . . . 55
print.sesameQC . . . 56
print.SigDF . . . 57
probeID_designType . . . 57
probeSuccessRate . . . 58
qualityMask . . . 59

qualityRank . . . 59
readFileSet . . . 60
readIDATpair . . . 61
reIdentify . . . 62
reopenSesame . . . 63
resetMask . . . 63
RGChannelSetToSigDFs . . . 64
scrub . . . 64
scrubSoft . . . 65
sdfPlatform . . . 66
sdf_read_table . . . 66
sdf_write_table . . . 67
searchIDATprefixes . . . 67
segmentBins . . . 68
sesamePlotIntensVsBetas . . . 69
sesamePlotRedGrnQQ . . . 69
sesameQC . . . 70
sesamize . . . 71
setMask . . . 72
setMaskBySpecies . . . 72
SigDF . . . 73
SigDFsToRGChannelSet . . . 73
SigDFToRatioSet . . . 74
signalMU . . . 75
sliceFileSet . . . 75
SNPcheck . . . 76
summaryExtractTest . . . 77
testEnrichment . . . 77
testEnrichment1 . . . 78
testEnrichmentFGSEA . . . 79
testEnrichmentFisher . . . 80
testEnrichmentGene . . . 81
testEnrichmentSpearman . . . 82
totalIntensities . . . 82
twoCompsEst2 . . . 83
visualizeGene . . . 84
visualizeProbes . . . 85
visualizeRegion . . . 86
visualizeSegments . . . 87

Index **89**

sesame-package *Analyze DNA methylation data*

Description

SEnsible and step-wise analysis of DNA methylation data

Details

This package complements array functionalities that allow processing >10,000 samples in parallel on clusters.

Author(s)

Wanding Zhou <Wanding.Zhou@vai.org>, Hui Shen <Hui.Shen@vai.org> Timothy J Triche Jr <Tim.Triche@vai.org>

See Also

Useful links:

- https://github.com/zwdzwd/sesame
- Report bugs at https://github.com/zwdzwd/sesame/issues

Examples

```
sdf <- readIDATpair(sub('_Grn.idat','',system.file(
    'extdata','4207113116_A_Grn.idat',package='sesameData')))

## The OpenSesame pipeline
betas <- openSesame(sdf)
```

addMask *Add probes to mask*

Description

This function essentially merge existing probe masking with new prboes to mask

Usage

```
addMask(sdf, probes)
```

Arguments

sdf	a SigDF
probes	a vector of probe IDs or a logical vector with TRUE representing masked probes

Value

a SigDF with added mask

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sum(sdf$mask)
sum(addMask(sdf, c("cg14057072", "cg22344912"))$mask)
```

---

as.data.frame.sesameQC

Coerce a sesameQC into a dataframe

Description

Coerce a sesameQC into a dataframe

Usage

```
## S3 method for class 'sesameQC'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	a sesameQC object
row.names	see as.data.frame
optional	see as.data.frame
...	see as.data.frame

Value

a data.frame

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
qc <- sesameQC(sdf)
df <- as.data.frame(qc)
```

attachManifest *Annotate a data.frame using manifest*

Description

Annotate a data.frame using manifest

Usage

```
attachManifest(df, probe_id = "Probe_ID", pfm = NULL, genome = NULL)
```

Arguments

df	input data frame with Probe_ID as a column
probe_id	the Probe_ID column name, default to "Probe_ID" or rownames
pfm	which array platform, guess from probe ID if not given
genome	the genome build, use default if not given

Value

a new data.frame with manifest attached

Examples

```
df = data.frame(Probe_ID = c("cg00101675_BC21", "cg00116289_BC21"))
attachManifest(df)
```

BetaValueToMValue *Convert beta-value to M-value*

Description

Logit transform a beta value vector to M-value vector.

Usage

```
BetaValueToMValue(b)
```

Arguments

b	vector of beta values

Details

Convert beta-value to M-value (aka logit transform)

Value

a vector of M values

Examples

```
BetaValueToMValue(c(0.1, 0.5, 0.9))
```

---

binSignals *Bin signals from probe signals*

Description

require GenomicRanges

Usage

```
binSignals(probe.signals, bin.coords, probe.coords)
```

Arguments

probe.signals	probe signals
bin.coords	bin coordinates
probe.coords	probe coordinates

Value

bin signals

---

bisConversionControl *Compute internal bisulfite conversion control*

Description

Compute GCT score for internal bisulfite conversion control. The function takes a `SigSet` as input. The higher the GCT score, the more likely the incomplete conversion.

Usage

```
bisConversionControl(sdf)
```

Arguments

sdf	a SigDF

Value

GCT score (the higher, the more incomplete conversion)

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
bisConversionControl(sdf)
```

`bSubComplete` *subset beta value matrix by complete probes*

Description

subset beta value matrix by complete probes

Usage

```
bSubComplete(betas)
```

Arguments

`betas`	beta value matrix

Value

subsetted beta value matrix

Examples

```
betas <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
betas <- bSubComplete(betas)
```

`bSubMostVariable` *Get most variable probes*

Description

Get most variable probes

Usage

```
bSubMostVariable(betas, n = 2000)
```

Arguments

betas	beta value matrix (row: cpg; column: sample)
n	number of most variable probes

Value

beta value matrix for the most variable probes

Examples

```
## get most variable autosome probes
betas <- sesameDataGet('HM450.10.TCGA.PAAD.normal')
betas.most.variable <- bSubMostVariable(
    betas[getAutosomeProbes('HM450'),],2000)

## clear cache
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

---

bSubProbes *subset beta value matrix by probes*

Description

subset beta value matrix by probes

Usage

```
bSubProbes(betas, probes)
```

Arguments

betas	beta value matrix
probes	probe set

Value

subsetted beta value matrix

Examples

```
probes <- getAutosomeProbes('HM450')
betas <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
betas <- bSubProbes(betas, probes)
```

calcDatabaseSetStatistics1

calcDatabaseSetStatistics1 calculates features of x

Description

calcDatabaseSetStatistics1 calculates features of x

Usage

```
calcDatabaseSetStatistics1(x)
```

Arguments

x	Vector of numeric values

Value

Vector with ~20 different engineered features

calcDatabaseSetStatisticsAll

calcDatabaseSetStatisticsAll builds dataset for a given betas matrix composed of engineered features from the given database sets

Description

calcDatabaseSetStatisticsAll builds dataset for a given betas matrix composed of engineered features from the given database sets

Usage

```
calcDatabaseSetStatisticsAll(betas, databaseSets)
```

Arguments

betas	matrix of beta values where probes are on the rows and samples are on the columns
databaseSets	List of vectors corresponding to probe locations for which the features will be extracted

Value

Vector for a given sample columns are features across different databaseSets

Examples

```
library(SummarizedExperiment)
se = sesameDataGet('MM285.20Kx467.SE')
samplesheet = colData(se)[, c("Mouse_Age_Months", "Mouse_Age_Days", "Sex",
"Strain_Corrected","Tissue_Corrected", 'Genotype')]
betas = assay(se)
databaseSetNames = c('KYCG.MM285.seqContextN.20210630',
'KYCG.MM285.probeType.20210630')
databaseSets = do.call(c, lapply(databaseSetNames, sesameDataGet))
calcDatabaseSetStatisticsAll(betas, databaseSets=databaseSets)
```

---

checkLevels *filter data matrix by factor completeness only works for discrete factors*

Description

filter data matrix by factor completeness only works for discrete factors

Usage

```
checkLevels(betas, fc)
```

Arguments

betas	matrix data
fc	factors, or characters

Value

a boolean vector whether there is non-NA value for each tested group for each probe

Examples

```
se0 = sesameDataGet("MM285.10.tissues")[1:1000,]
se_ok = checkLevels(SummarizedExperiment::assay(se0),
    SummarizedExperiment::colData(se0)$tissue)
sum(se_ok) # number of good probes
se1 = se0[se_ok,]
```

chipAddressToSignal *Lookup address in one sample*

Description

Lookup address and transform address to probe

Usage

```
chipAddressToSignal(dm, mft)
```

Arguments

dm	data frame in chip address, 2 columns: cy3/Grn and cy5/Red
mft	a data frame with columns Probe_ID, M, U and col

Details

Translate data in chip address to probe address. Type I probes can be separated into Red and Grn channels. The methylated allele and unmethylated allele are at different addresses. For type II probes methylation allele and unmethylated allele are at the same address. Grn channel is for methylated allele and Red channel is for unmethylated allele. The out-of-band signals are type I probes measured using the other channel.

Value

a SigDF, indexed by probe ID address

cnSegmentation *Perform copy number segmentation*

Description

Perform copy number segmentation using the signals in the signal set. The function takes a SigDF for the target sample and a set of normal SigDF for the normal samples. An optional arguments specifies the version of genome build that the inference will operate on. The function outputs an object of class CNSegment with signals for the segments (seg.signals), the bin coordinates (bin.coords) and bin signals (bin.signals).

Usage

```
cnSegmentation(sdf, sdfs.normal = NULL, refversion = c("hg19", "hg38"))
```

Arguments

sdf	SigDF
sdfs.normal	a list of SigDFs for normalization, if not given, use the stored normal data from sesameData. However, we do recommend using a matched copy number normal dataset for normalization.
refversion	hg19 or hg38

Value

an object of CNSegment

Examples

```
sesameDataCache("EPIC") # in case not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdfs.normal <- sesameDataGet('EPIC.5.SigDF.normal')[1:3]
seg <- cnSegmentation(sdf, sdfs.normal)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

---

compareDatbaseSetOverlap

compareDatbaseSetOverlap calculates the pariwise overlap between given list of database sets using a distance metric.

Description

compareDatbaseSetOverlap calculates the pariwise overlap between given list of database sets using a distance metric.

Usage

```
compareDatbaseSetOverlap(
  databaseSets = NA,
  metric = "Jaccard",
  verbose = FALSE
)
```

Arguments

databaseSets	List of vectors corresponding to the database sets of interest with associated meta data as an attribute to each element. Optional. (Default: NA)
metric	String representing the similarity metric to use. Optional. (Default: "Jaccard").
verbose	Logical value indicating whether to display intermediate text output about the type of test. Optional. (Default: FALSE)

Value

An upper triangular matrix containing a metric (Jaccard) comparing the pairwise distances between database sets.

Examples

```
databaseSetNames = c('KYCG.MM285.seqContextN.20210630')
databaseSets = do.call(c, lapply(databaseSetNames, sesameDataGet))
compareDatbaseSetOverlap(databaseSets)
```

---

compareMouseStrainReference

Compare Strain SNPs with a reference panel

Description

Compare Strain SNPs with a reference panel

Usage

```
compareMouseStrainReference(betas = NULL, show_sample_names = FALSE)
```

Arguments

betas	beta value vector or matrix (for multiple samples)
show_sample_names	whether to show sample name

Value

grid object that contrast the target sample with pre-built mouse strain reference

Examples

```
sesameDataCache("MM285") # if not done yet
compareMouseStrainReference()
```

compareMouseTissueReference *Compare mouse array data with mouse tissue references*

Description

Compare mouse array data with mouse tissue references

Usage

```
compareMouseTissueReference(betas = NULL, color = "blueYellow")
```

Arguments

betas	matrix of betas for the target sample
color	either blueYellow or fullJet

Value

grid object that contrast the target sample with pre-built mouse tissue reference

Examples

```
sesameDataCache("MM285") # if not done yet
compareMouseTissueReference()

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

controls *get the controls attributes*

Description

get the controls attributes

Usage

```
controls(sdf)
```

Arguments

sdf	a SigDF

Value

the controls data frame

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf = sesameDataGet('EPIC.1.SigDF')
head(controls(sdf))
```

---

createDatabaseSetNetwork

createGeneNetwork creates databaseSet network using the Jaccard index.

Description

createGeneNetwork creates databaseSet network using the Jaccard index.

Usage

```
createDatabaseSetNetwork(databaseSets)
```

Arguments

databaseSets	Vector of probes corresponding to a single database set of interest.

Value

ggplot lollipop plot

Examples

```
databaseSetNames = c('KYCG.MM285.seqContextN.20210630')
databaseSets = do.call(c, lapply(databaseSetNames, sesameDataGet))
createDatabaseSetNetwork(databaseSets)
```

createUCSCtrack *Turn beta values into a UCSC browser track*

Description

Turn beta values into a UCSC browser track

Usage

```
createUCSCtrack(betas, output = NULL, platform = "HM450", refversion = "hg38")
```

Arguments

betas	a named numeric vector
output	output file name
platform	HM450, EPIC etc.
refversion	hg38, hg19 etc.

Value

when output is null, return a data.frame, otherwise NULL

Examples

```
betas.tissue <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
## add output to create an actual file
df <- createUCSCtrack(betas.tissue)

## to convert to bigBed
## sort -k1,1 -k2,2n output.bed >output_sorted.bed
## bedToBigBed output_sorted.bed hg38.chrom output.bb
```

deIdentify *De-identify IDATs by removing SNP probes*

Description

Mask SNP probe intensity mean by zero.

Usage

```
deIdentify(path, out_path = NULL, snps = NULL, mft = NULL, randomize = FALSE)
```

Arguments

path	input IDAT file
out_path	output IDAT file
snps	SNP definition, if not given, default to SNP probes
mft	sesame-compatible manifest if non-standard
randomize	whether to randomize the SNPs. if TRUE, randomize the signal intensities. one can use set.seed to reidentify the IDAT with the secret seed (see examples). If FALSE, this sets all SNP intensities to zero.

Value

NULL, changes made to the IDAT files

Examples

```
my_secret <- 13412084
set.seed(my_secret)
temp_out <- tempfile("test")
deIdentify(system.file(
    "extdata", "4207113116_A_Grn.idat", package = "sesameData"),
     temp_out, randomize = TRUE)
unlink(temp_out)
```

---

detectionPnegEcdf *Detection P-value based on ECDF of negative control*

Description

The function takes a SigDF as input, computes detection p-value using negative control probes' empirical distribution and returns a new SigDF with an updated mask slot.

Usage

```
detectionPnegEcdf(sdf, return.pval = FALSE, pval.threshold = 0.05)
```

Arguments

sdf	a SigDF
return.pval	whether to return p-values, instead of a masked SigDF
pval.threshold	minimum p-value to mask

Value

a SigDF, or a p-value vector if return.pval is TRUE

Examples

```
sdf <- sesameDataGet("EPIC.1.SigDF")
sum(sdf$mask)
sum(detectionPnegEcdf(sdf)$mask)
```

`detectionPoobEcdf` *Detection P-value based on ECDF of out-of-band signal*

Description

aka pOOBAH (p-vals by Out-Of-Band Array Hybridization)

Usage

```
detectionPoobEcdf(
  sdf,
  return.pval = FALSE,
  combine.neg = TRUE,
  pval.threshold = 0.05
)

pOOBAH(sdf, return.pval = FALSE, combine.neg = TRUE, pval.threshold = 0.05)
```

Arguments

`sdf`	a `SigDF`
`return.pval`	whether to return p-values, instead of a masked `SigDF`
`combine.neg`	whether to combine negative control probes with the out-of-band probes in simulating the signal background
`pval.threshold`	minimum p-value to mask

Details

The function takes a `SigDF` as input, computes detection p-value using out-of-band probes empirical distribution and returns a new `SigDF` with an updated mask slot.

Value

a `SigDF`, or a p-value vector if return.pval is TRUE

Examples

```
sdf <- sesameDataGet("EPIC.1.SigDF")
sum(sdf$mask)
sum(detectionPoobEcdf(sdf)$mask)

sdf <- sesameDataGet("EPIC.1.SigDF")
sum(sdf$mask)
sum(resetMask(sdf)$mask)
sum(pOOBAH(sdf, pval.threshold=0.2)$mask)
```

---

detectionPoobEcdf2 *Detection P-value based on ECDF of out-of-band signal*

Description

aka pOOBAH2 (p-vals by Out-Of-Band Array Hybridization)

Usage

```
detectionPoobEcdf2(
  sdf,
  return.pval = FALSE,
  combine.neg = TRUE,
  pval.threshold = 0.05
)

pOOBAH2(sdf, return.pval = FALSE, combine.neg = TRUE, pval.threshold = 0.05)
```

Arguments

sdf	a SigDF
return.pval	whether to return p-values, instead of a masked SigDF
combine.neg	whether to combine negative control probes with the out-of-band probes in simulating the signal background
pval.threshold	minimum p-value to mask

Details

The function takes a SigDF as input, computes detection p-value using out-of-band probes empirical distribution and returns a new SigDF with an updated mask slot.

The difference between this function and the original pOOBAH is that pOOBAH2 is based on background-subtracted and dyebias corrected signal and do not distinguish the color channel difference.

Value

a SigDF, or a p-value vector if return.pval is TRUE

Examples

```
sdf <- sesameDataGet("EPIC.1.SigDF")
sum(sdf$mask)
sdf <- detectionPoobEcdf2(sdf)
sum(sdf$mask)
sdf <- sesameDataGet("EPIC.1.SigDF")
sum(sdf$mask)
sdf <- pOOBAH2(sdf)
sum(sdf$mask)
```

---

diffRefSet *Restrict refset to differentially methylated probes use with care, might introduce bias*

Description

The function takes a matrix with probes on the rows and cell types on the columns and output a subset matrix and only probes that show discordant methylation levels among the cell types.

Usage

```
diffRefSet(g)
```

Arguments

g	a matrix with probes on the rows and cell types on the columns

Value

g a matrix with a subset of input probes (rows)

Examples

```
g <- diffRefSet(getRefSet(platform='HM450'))
```

---

dmContrasts *List all contrasts of a DMLSummary*

Description

List all contrasts of a DMLSummary

Usage

```
dmContrasts(smry)
```

Arguments

smry	a DMLSummary object

Value

a character vector of contrasts

Examples

```
data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:1000,], ~type, meta=data$sampleInfo)
dmContrasts(smry)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

DML *Test differential methylation on each locus*

Description

The function takes a beta value matrix with probes on the rows and samples on the columns. It also takes a sample information data frame (meta) and formula for testing. The function outputs a list of coefficient tables for each factor tested.

Usage

```
DML(betas, fm, meta = NULL, mc.cores = 1)
```

Arguments

betas	beta values, matrix or SummarizedExperiment rows are probes and columns are samples.
fm	formula
meta	data frame for sample information, column names are predictor variables (e.g., sex, age, treatment, tumor/normal etc) and are referenced in formula. Rows are samples. When the betas argument is a SummarizedExperiment object, this is ignored. colData(betas) will be used instead.
mc.cores	number of cores for parallel processing

Value

a list of test summaries, summary.lm objects

Examples

```
sesameDataCache("HM450") # in case not done yet
data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:1000,], ~type, meta=data$sampleInfo)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

DMR *Find Differentially Methylated Region (DMR)*

Description

This subroutine uses Euclidean distance to group CpGs and then combine p-values for each segment. The function performs DML test first if cf is NULL. It groups the probe testing results into differential methylated regions in a coefficient table with additional columns designating the segment ID and statistical significance (P-value) testing the segment.

Usage

```
DMR(
  betas,
  smry,
  contrast,
  platform = NULL,
  refversion = NULL,
  dist.cutoff = NULL,
  seg.per.locus = 0.5
)
```

Arguments

betas	beta values for distance calculation
smry	DML
contrast	the pair-wise comparison or contrast check colnames(attr(smry, "model.matrix")) if uncertain
platform	EPIC, HM450, MM285, ...
refversion	hg38, hg19, mm10, ...
dist.cutoff	distance cutoff (default to use dist.cutoff.quantile)
seg.per.locus	number of segments per locus higher value leads to more segments

Value

coefficient table with segment ID and segment P-value each row is a locus, multiple loci may share a segment ID if they are merged to the same segment. Records are ordered by Seg_Est.

Examples

```
sesameDataCache("HM450") # in case not done yet

data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:1000,], ~type, meta=data$sampleInfo)
colnames(attr(smry, "model.matrix")) # pick a contrast from here
merged_segs = DMR(data$betas[1:1000,], smry, "typeTumour")

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

`dyeBiasCorr` *Correct dye bias in by linear scaling.*

Description

The function takes a `SigDF` as input and scale both the Grn and Red signal to a reference (ref) level. If the reference level is not given, it is set to the mean intensity of all the in-band signals. The function returns a `SigDF` with dye bias corrected.

Usage

```
dyeBiasCorr(sdf, ref = NULL)
```

Arguments

`sdf`	a `SigDF`
`ref`	reference signal level

Value

a normalized `SigDF`

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.db <- dyeBiasCorr(sdf)
```

dyeBiasCorrMostBalanced *Correct dye bias using most balanced sample as the reference*

Description

The function chose the reference signal level from a list of SigDF. The chosen sample has the smallest difference in Grn and Red signal intensity as measured using the normalization control probes. In practice, it doesn't matter which sample is chosen as long as the reference level does not deviate much. The function returns a list of SigDFs with dye bias corrected.

Usage

```
dyeBiasCorrMostBalanced(sdfs)
```

Arguments

sdfs	a list of normalized SigDFs

Value

a list of normalized SigDFs

Examples

```
sesameDataCache("HM450") # if not done yet
sdfs <- sesameDataGet('HM450.10.SigDF')
sdfs.db <- dyeBiasCorrMostBalanced(sdfs)
```

dyeBiasCorrTypeINorm *Dye bias correction by matching green and red to mid point*

Description

This function compares the Type-I Red probes and Type-I Grn probes and generates and mapping to correct signal of the two channels to the middle. The function takes one single SigDF and returns a SigDF with dye bias corrected.

Usage

```
dyeBiasCorrTypeINorm(sdf)

dyeBiasNL(sdf)
```

Arguments

sdf	a SigDF

Value

a SigDF after dye bias correction.

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.db <- dyeBiasCorrTypeINorm(sdf)
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf <- dyeBiasNL(sdf)
```

---

dyeBiasDistortion *Quantify how much dye bias in high signal range deviates from the global median*

Description

Positive value indicates augmentation of high-end dye bias over low-end. negative value represents high-end dye bias contradicts that at low-end (a distorted dye bias). Negative distortion score (< -1) suggests low experiment quality. 0 suggests a consistent dye bias at high and low-end.

Usage

```
dyeBiasDistortion(sdf)
```

Arguments

sdf	a SigDF

Value

a numeric score

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
dyeBiasDistortion(sdf)
```

`estimateCellComposition` *Estimate cell composition using reference*

Description

This is a reference-based cell composition estimation. The function takes a reference methylation status matrix (rows for probes and columns for cell types, can be obtained by getRefSet function) and a query beta value measurement. The length of the target beta values should be the same as the number of rows of the reference matrix. The method assumes one unknown component. It outputs a list containing the estimated cell fraction, the error of optimization and methylation status of the unknown component.

Usage

```
estimateCellComposition(g, q, refine = TRUE, dichotomize = FALSE, ...)
```

Arguments

`g`	reference methylation
`q`	target measurement: length(q) == nrow(g)
`refine`	to refine estimate, takes longer
`dichotomize`	to dichotomize query beta value before estimate, this relieves unclean background subtraction
`...`	extra parameters for optimization, this includes temp - annealing temperature (0.5) maxIter - maximum iteration to stop after converge (1000) delta - delta score to reset counter (0.0001) verbose - output debug info (FALSE)

Value

a list of fraction, min error and unknown component methylation state

`estimateLeukocyte` *Estimate leukocyte fraction using a two-component model*

Description

The method assumes only two components in the mixture: the leukocyte component and the target tissue component. The function takes the beta values matrix of the target tissue and the beta value matrix of the leukocyte. Both matrices have probes on the row and samples on the column. Row names should have probe IDs from the platform. The function outputs a single numeric describing the fraction of leukocyte.

Usage

```
estimateLeukocyte(
  betas.tissue,
  betas.leuko = NULL,
  betas.tumor = NULL,
  platform = c("EPIC", "HM450", "HM27")
)
```

Arguments

betas.tissue	tissue beta value matrix (#probes X #samples)
betas.leuko	leukocyte beta value matrix, if missing, use the SeSAMe default by infinium platform
betas.tumor	optional, tumor beta value matrix
platform	"HM450", "HM27" or "EPIC"

Value

leukocyte estimate, a numeric vector

Examples

```
betas.tissue <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
estimateLeukocyte(betas.tissue)
```

---

extractDesign *Extract the first design category*

Description

Extract the first design category

Usage

```
extractDesign(design_str)
```

Arguments

design_str	Design string in e.g., the mouse array

Value

a character vector for the design category

formatVCF *Convert SNP from Infinium array to VCF file*

Description

Convert SNP from Infinium array to VCF file

Usage

```
formatVCF(sdf, vcf = NULL, refversion = "hg19", annoS = NULL, annoI = NULL)
```

Arguments

sdf	SigDF
vcf	output VCF file path, if NULL output to console
refversion	reference version, currently only support
annoS	SNP variant annotation, download if not given
annoI	Infinium-I variant annotation, download if not given hg19 and hg38 in human

Value

VCF file. If vcf is NULL, a data.frame is output to console. The data.frame does not contain VCF headers.

Note the vcf is not sorted. You can sort with awk '$1 ~ /^#/ print $0;next print $0 | "sort -k1,1 -k2,2n"'

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')

annoS <- sesameDataGetAnno("EPIC/EPIC.hg19.snp_overlap_b151.rds")
annoI <- sesameDataGetAnno("EPIC/EPIC.hg19.typeI_overlap_b151.rds")
## output to console
head(formatVCF(sdf, annoS=annoS, annoI=annoI))
```

getAFTypeIbySumAlleles *Get allele frequency treating type I by summing alleles*

Description

Takes a SigDF as input and returns a numeric vector containing extra allele frequencies based on Color-Channel-Switching (CCS) probes. If no CCS probes exist in the SigDF, then an numeric(0) is returned.

Usage

```
getAFTypeIbySumAlleles(sdf, known.ccs.only = TRUE)
```

Arguments

sdf	SigDF
known.ccs.only	consider only known CCS probes

Value

beta values

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
af <- getAFTypeIbySumAlleles(sdf)
```

getAutosomeProbes *Get autosome probes*

Description

Get autosome probes

Usage

```
getAutosomeProbes(platform = c("EPIC", "HM450", "MM285"), refversion = NULL)
```

Arguments

platform	'EPIC', 'HM450' etc.
refversion	hg19, hg38, or mm10, inference by default

Value

a vector of autosome probes

Examples

```
auto.probes <- getAutosomeProbes('MM285')
```

`getBetas` *Get beta Values*

Description

sum.typeI is used for rescuing beta values on Color-Channel-Switching CCS probes. The function takes a `SigDF` and returns beta value except that Type-I in-band signal and out-of-band signal are combined. This prevents color-channel switching due to SNPs.

Usage

```
getBetas(sdf, mask = TRUE, sum.TypeI = FALSE)
```

Arguments

`sdf`	`SigDF`
`mask`	whether to use mask
`sum.TypeI`	whether to sum type I channels

Value

a numeric vector, beta values

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
betas <- getBetas(sdf)
```

getBinCoordinates *Get bin coordinates*

Description

requires GenomicRanges, IRanges

Usage

```
getBinCoordinates(seqInfo, gapInfo, probe.coords)
```

Arguments

seqInfo	chromosome information object
gapInfo	chromosome gap information
probe.coords	probe coordinates

Value

bin.coords

getDatabaseSetOverlap *getDatabaseSetOverlap tests for the overlap of set of probes (querySet) in a single given feature (database set)*

Description

getDatabaseSetOverlap tests for the overlap of set of probes (querySet) in a single given feature (database set)

Usage

```
getDatabaseSetOverlap(querySet, databaseSets, platform = NA, verbose = TRUE)
```

Arguments

querySet	Vector of probes corresponding to a single database set of interest.
databaseSets	List of vectors corresponding to the database sets of interest with associated meta data as an attribute to each element.
platform	String corresponding to the type of platform to use. Either MM285, EPIC, HM450, or HM27. If it is not provided, it will be inferred from the query set probeIDs (Default: NA).
verbose	Logical value indicating whether to display intermediate text output about the type of test. Optional. (Default: FALSE)

Value

A sparse data.frame containing all of the meta data from all database sets.

Examples

```
library(SummarizedExperiment)
MM285.tissueSignature = sesameDataGet('MM285.141.SE.tissueSignature')
df = rowData(MM285.tissueSignature)
querySet = df$Probe_ID[df$branch == "E-Brain"]
databaseSetNames = c('KYCG.MM285.seqContextN.20210630',
'KYCG.MM285.designGroup.20210210')
databaseSets = do.call(c, lapply(databaseSetNames, sesameDataGet))
getDatabaseSetOverlap(querySet, databaseSets)
```

---

getNormCtls *get normalization control signal*

Description

get normalization control signal from SigDF. The function optionally takes mean for each channel.

Usage

```
getNormCtls(sdf, average = FALSE)
```

Arguments

sdf	a SigDF
average	whether to average

Value

a data frame of normalization control signals

Examples

```
sdf <- readIDATpair(file.path(system.file(
    'extdata','',package='sesameData'), '4207113116_B'))

df.ctl <- getNormCtls(sdf)
```

getProbesByChromosome *Get Probes by Chromosome*

Description

Get Probes by Chromosome

Usage

```
getProbesByChromosome(
  chrms,
  platform = c("EPIC", "HM450", "MM285"),
  refversion = NULL
)
```

Arguments

chrms	chromosomes to subset
platform	EPIC, HM450, Mouse
refversion	hg19, hg38, or mm10, inference by default

Value

a vector of probes on the selected chromosomes

Examples

```
sex.probes <- getProbesByChromosome(c('chrX','chrY'))
```

getProbesByGene *Get Probes by Gene*

Description

Get probes mapped to a gene. All transcripts for the gene are considered. The function takes a gene name as appears in UCSC RefGene database. The platform and reference genome build can be changed with ‘platform‘ and ‘refversion‘ options. The function returns a vector of probes that falls into the given gene.

Usage

```
getProbesByGene(
  geneName,
  platform = c("EPIC", "HM450", "MM285"),
  upstream = 0,
  dwstream = 0,
  refversion = c("hg38", "hg19", "mm10")
)
```

Arguments

geneName	gene name
platform	EPIC or HM450
upstream	number of bases to expand upstream of target gene
dwstream	number of bases to expand downstream of target gene
refversion	hg38 or hg19

Value

probes that fall into the given gene

Examples

```
probes <- getProbesByGene('CDKN2A', upstream=500, dwstream=500)
```

---

getProbesByRegion *Get probes by genomic region*

Description

The function takes a genomic coordinate and output the a vector of probes on the specified platform that falls in the given genomic region.

Usage

```
getProbesByRegion(
  chrm,
  beg = 1,
  end = -1,
  platform = c("EPIC", "HM450"),
  refversion = c("hg38", "hg19")
)
```

Arguments

chrm	chromosome
beg	begin, 1 if omitted
end	end, chromosome end if omitted
platform	EPIC or HM450
refversion	hg38 or hg19

Value

probes that fall into the given region

Examples

```
getProbesByRegion('chr5', 135413937, 135419936,
    refversion = 'hg19', platform = 'HM450')
```

`getProbesByTSS` *Get Probes by Gene Transcription Start Site (TSS)*

Description

Get probes mapped to a TSS. All transcripts for the gene are considered. The function takes a gene name as appears in UCSC RefGene database. The platform and reference genome build can be changed with ‘platform‘ and ‘refversion‘ options. The function returns a vector of probes that falls into the TSS region of the gene.

Usage

```
getProbesByTSS(
  geneName,
  upstream = 1500,
  dwstream = 1500,
  platform = c("EPIC", "HM450", "MM285"),
  refversion = c("hg38", "hg19", "mm10")
)
```

Arguments

`geneName`	gene name
`upstream`	the number of base pairs to expand upstream the TSS
`dwstream`	the number of base pairs to expand dwstream the TSS
`platform`	EPIC, HM450, or MM285
`refversion`	hg38, hg19 or mm10

Value

probes that fall into the given gene

Examples

```
probes <- getProbesByTSS('CDKN2A')
```

getRefSet *Retrieve reference set*

Description

The function retrieves the curated reference DNA methylation status for a set of cell type names under the Infinium platform. Supported cell types include "CD4T", "CD19B", "CD56NK", "CD14Monocytes", "granulocytes", "scFat", "skin" etc. See package sesameData for more details. The function output a matrix with probes on the rows and specified cell types on the columns. 0 suggests unmethylation and 1 suggests methylation. Intermediate methylation and nonclusive calls are left with NA.

Usage

```
getRefSet(cells = NULL, platform = c("EPIC", "HM450"))
```

Arguments

cells	reference cell types
platform	EPIC or HM450

Value

g, a 0/1 matrix with probes on the rows and specified cell types on the columns.

Examples

```
betas <- getRefSet('CD4T', platform='HM450')
```

getSexInfo *Get sex-related information*

Description

The function takes a SigDF and returns a vector of three numerics: the median intensity of chrY probes; the median intensity of chrX probes; and fraction of intermediate chrX probes. chrX and chrY probes excludes pseudo-autosomal probes.

Usage

```
getSexInfo(sdf)
```

Arguments

sdf	a SigDF

Value

medianY and medianX, fraction of XCI, methylated and unmethylated X probes, median intensities of auto-chromosomes.

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
getSexInfo(sdf)
```

`inferEthnicity` *Infer Ethnicity*

Description

This function uses both the built-in rsprobes as well as the type I Color-Channel-Switching probes to infer ethnicity.

Usage

```
inferEthnicity(sdf)
```

Arguments

`sdf`	a `SigDF`

Details

s better be background subtracted and dyebias corrected for best accuracy

Please note: the betas should come from SigDF *without* channel inference.

Value

string of ethnicity

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
inferEthnicity(sdf)
```

inferInfiniumIChannel *Infer and reset color channel for Type-I probes instead of using what is specified in manifest. The results are stored to sdf@extra$IGG and sdf@extra$IRR slot.*

Description

IGG => Type-I green that is inferred to be green IRR => Type-I red that is inferred to be red

Usage

```
inferInfiniumIChannel(
  sdf,
  switch_failed = FALSE,
  verbose = FALSE,
  summary = FALSE
)
```

Arguments

sdf	a SigDF
switch_failed	whether to switch failed probes (default to FALSE)
verbose	whether to print correction summary
summary	return summarized numbers only.

Value

a SigDF, or numerics if summary == TRUE

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
inferInfiniumIChannel(sdf)
```

inferSex *Infer Sex*

Description

Infer Sex

Usage

```
inferSex(x, pfm = NULL)
```

Arguments

x	either a raw SigDF or a beta value vector named by probe ID SigDF is preferred over beta values.
pfm	platform Only MM285, EPIC and HM450 are supported.

Value

'F' or 'M' We established our sex calling based on the CpGs hypermethylated in inactive X (XiH), CpGs hypomethylated in inactive X (XiL) and signal intensity ratio of Y-chromosome over autosomes. Currently human inference uses a random forest and mouse inference uses a support vector machine.

The function checks the sample quality. If the sample is of poor quality the inference return NA.

Note many factors such as Dnmt genotype, XXY male (Klinefelter's), 45,X female (Turner's) can confuse the model sometimes. This function works on a single sample.

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf = sesameDataGet('EPIC.1.SigDF')
inferSex(sdf)
```

---

inferSexKaryotypes *Infer Sex Karyotype*

Description

The function takes a SigDF and infers the sex chromosome Karyotype and presence/absence of X-chromosome inactivation (XCI). chrX, chrY and XCI are inferred relatively independently. This function gives a more detailed look of potential sex chromosome aberrations.

Usage

```
inferSexKaryotypes(sdf)
```

Arguments

sdf	a SigDF

Value

Karyotype string, with XCI

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
inferSexKaryotypes(sdf)
```

`inferSpecies` *Infer Species*

Description

We infer species based on probes pvalues and alignment score. AUC was calculated for each specie, y_true is 1 or 0 for pval < threshold.pos or pval > threshold.neg, respeceively,

Usage

```
inferSpecies(
  sdf,
  df_as = NULL,
  topN = 3000,
  threshold.pos = 0.01,
  threshold.neg = 0.1,
  ret.max = TRUE,
  balance = TRUE,
  threshold.sucess.rate = 0.8
)
```

Arguments

`sdf`	a `SigSet`
`df_as`	a data.frame of alignment score for each probe.
`topN`	Top n positive and negative probes used to infer species.
`threshold.pos`	pvalue < threshold.pos are considered positive (default: 0.01).
`threshold.neg`	pvalue > threshold.neg are considered negative (default: 0.2).
`ret.max`	whether to return the species with maximal AUC.
`balance`	whether to balance the postive and negative probe number (default: TRUE).
`threshold.sucess.rate`	threshold of success rate to determine mouse species.

Value

a list of auc, pvalue, species (NCBI official species names) and taxid.

Examples

```
if (FALSE) { ## remove this, testing doesn't allow large file caching
  sdf = sesameDataGet("MM285.1.SigDF")
  inferSpecies(sdf)
}
```

`inferStrain` *Infer strain information for mouse array*

Description

Infer strain information for mouse array

Usage

```
inferStrain(betas, strain_snp_table = NULL)
```

Arguments

`betas`	beta value vector from which VAFs are extracted
`strain_snp_table`	if not given download the default from sesameData

Value

a list of best guess, p-value of the best guess and the probabilities of all strains

Examples

```
sesameDataCache("MM285") # if not done yet
sdf = sesameDataGet('MM285.1.SigDF')
betas = getBetas(dyeBiasNL(noob(sdf)))
inferStrain(betas)
```

`inferTissue` *inferTissue1 infers the tissue of a single sample (as identified through the branchIDs in the row data of the reference) by reporting independent composition through cell type deconvolution.*

Description

inferTissue1 infers the tissue of a single sample (as identified through the branchIDs in the row data of the reference) by reporting independent composition through cell type deconvolution.

Usage

```
inferTissue(
  betas,
  reference = NULL,
  platform = NULL,
  abs_delta_beta_min = 0.3,
  auc_min = 0.99,
```

```
  coverage_min = 0.8,
  topN = 15
)
```

Arguments

betas	Named vector with probes and their corresponding beta value measurement
reference	Summarized Experiment with either hypomethylated or hypermethylated probe selection (row data), sample selection (column data), meta data, and the betas (assay)
platform	String representing the array type of the betas and reference
abs_delta_beta_min	Numerical value indicating the absolute minimum required delta beta for the probe selection criteria
auc_min	Numeric value corresponding to the minimum AUC value required for a probe to be considered
coverage_min	Numeric value corresponding to the minimum coverage requirement for a probe to be considered. Coverage is defined here as the proportion of samples without an NA value at a given probe.
topN	number of probes to at most use for each branch

Value

inferred tissue as a string

Examples

```
sesameDataCache("MM285") # if not done yet
sdf = sesameDataGet("MM285.1.SigDF")
inferTissue(getBetas(dyeBiasNL(noob(sdf))))
```

---

initFileSet *initialize a fileSet class by allocating appropriate storage*

Description

initialize a fileSet class by allocating appropriate storage

Usage

```
initFileSet(map_path, platform, samples, probes = NULL, inc = 4)
```

Arguments

map_path	path of file to map
platform	EPIC, HM450 or HM27, consistent with sdfPlatform(sdf)
samples	sample names
probes	probe names
inc	bytes per unit data storage

Value

a sesame::fileSet object

Examples

```
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))
```

---

isUniqProbeID — *Whether the probe ID is the uniq probe ID like in the mouse array, e.g., cg36609548*

Description

Whether the probe ID is the uniq probe ID like in the mouse array, e.g., cg36609548

Usage

```
isUniqProbeID(Probe_ID)
```

Arguments

Probe_ID	Probe ID

Value

a logical(1), whether the probe ID is based on the new ID system

mapFileSet *Deposit data of one sample to a fileSet (and hence to file)*

Description

Deposit data of one sample to a fileSet (and hence to file)

Usage

```
mapFileSet(fset, sample, named_values)
```

Arguments

fset	a sesame::fileSet, as obtained via readFileSet
sample	sample name as a string
named_values	value vector named by probes

Value

a sesame::fileSet

Examples

```
## create two samples
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))

## a hypothetical numeric array (can be beta values, intensities etc)
hypothetical <- setNames(runif(fset$n), fset$probes)

## map the numeric to file
mapFileSet(fset, 's1', hypothetical)

## get data
sliceFileSet(fset, 's1', 'cg00000292')
```

meanIntensity *Whole-dataset-wide Mean Intensity*

Description

The function takes one single SigDF and computes mean intensity of all the in-band measurements. This includes all Type-I in-band measurements and all Type-II probe measurements. Both methylated and unmethylated alleles are considered. This function outputs a single numeric for the mean.

Usage

```
meanIntensity(sdf, mask = TRUE)
```

Arguments

sdf	a SigDF
mask	whether to mask probes using mask column

Details

Note: mean in this case is more informative than median because methylation level is mostly bi-modal.

Value

mean of all intensities

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
meanIntensity(sdf)
```

---

medianTotalIntensity *Whole-dataset-wide Median Total Intensity (M+U)*

Description

The function takes one single SigDF and computes median intensity of M+U for each probe. This function outputs a single numeric for the median.

Usage

```
medianTotalIntensity(sdf, mask = TRUE)
```

Arguments

sdf	a SigDF
mask	whether to mask probes using mask column

Value

median of all intensities

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
medianTotalIntensity(sdf)
```

MValueToBetaValue *Convert M-value to beta-value*

Description

Convert M-value to beta-value (aka inverse logit transform)

Usage

```
MValueToBetaValue(m)
```

Arguments

m	a vector of M values

Value

a vector of beta values

Examples

```
MValueToBetaValue(c(-3, 0, 3))
```

neob *Negative control plus out-of-band background correction*

Description

The function takes a SigDF and returns a modified SigDF with background subtracted. Background was modelled in a normal distribution and true signal in an exponential distribution. The Norm-Exp deconvolution is parameterized using both negative control and Out-Of-Band (oob) probes

Usage

```
neob(sdf, offset = 15)
```

Arguments

sdf	a SigDF
offset	offset

Value

a new SigDF with neob background correction

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.nb <- neob(sdf)
```

noMasked *remove masked probes from SigDF*

Description

remove masked probes from SigDF

Usage

```
noMasked(sdf)
```

Arguments

sdf	input SigDF object

Value

a SigDF object without masked probes

Examples

```
sesameDataCache("EPIC")
sdf = sesameDataGet("EPIC.1.SigDF")
sdf = pOOBAH(sdf)

sdf_noMasked = noMasked(sdf)
```

noob *Noob background correction*

Description

The function takes a SigDF and returns a modified SigDF with background subtracted. Background was modelled in a normal distribution and true signal in an exponential distribution. The Norm-Exp deconvolution is parameterized using Out-Of-Band (oob) probes

Usage

```
noob(sdf, offset = 15)
```

Arguments

sdf	a SigDF
offset	offset

Value

a new SigDF with noob background correction

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.nb <- noob(sdf)
```

---

openSesame *The openSesame pipeline*

Description

This function is a simple wrapper of noob + nonlinear dye bias correction + pOOBAH masking.

Usage

```
openSesame(x, platform = "", manifest = NULL, BPPARAM = SerialParam(), ...)
```

Arguments

x	SigDF(s), IDAT prefix(es), minfi GenomicRatioSet(s), or RGChannelSet(s)
platform	optional platform string
manifest	optional dynamic manifest
BPPARAM	get parallel with MulticoreParam(n)
...	parameters to getBetas

Details

If the input is an IDAT prefix or a SigDF, the output is the beta value numerics. If the input is a minfi GenomicRatioSet or RGChannelSet, the output is the sesamized GenomicRatioSet.

Value

a numeric vector for processed beta values

Examples

```
sdf <- sesameDataGet('HM450.1.TCGA.PAAD')$sdf
IDATprefixes <- searchIDATprefixes(
    system.file("extdata", "", package = "sesameData"))
betas <- openSesame(IDATprefixes)
```

`openSesameToFile` *openSesame pipeline with file-backed storage*

Description

openSesame pipeline with file-backed storage

Usage

```
openSesameToFile(map_path, idat_dir, BPPARAM = SerialParam(), inc = 4)
```

Arguments

`map_path`	path of file to be mapped (beta values file)
`idat_dir`	source IDAT directory
`BPPARAM`	get parallel with MulticoreParam(2)
`inc`	bytes per item data storage. increase to 8 if precision is important. Most cases 32-bit representation is enough.

Value

a sesame::fileSet

Examples

```
openSesameToFile('mybetas',
    system.file('extdata',package='sesameData'))
```

`plotLollipop` *plotLollipop creates a lollipop plot of log(estimate) given data with fields estimate.*

Description

plotLollipop creates a lollipop plot of log(estimate) given data with fields estimate.

Usage

```
plotLollipop(data, n = 10, title = NA, subtitle = NA)
```

Arguments

data	DataFrame where each field is a database name with its estimate.
n	Integer representing the number of top enrichments to report. Optional. (Default: 10)
title	String representing the title label. Optional. (Default: NA)
subtitle	String representing the subtitle label. Optional. (Default: NA)

Value

ggplot lollipop plot

Examples

```
data = data.frame(estimate=c(runif(10, 0, 10)))
plotLollipop(data)
```

---

plotVolcano *plotVolcano creates a volcano plot of -log2(p.value) and log(estimate) given data with fields estimate and p.value.*

Description

plotVolcano creates a volcano plot of -log2(p.value) and log(estimate) given data with fields estimate and p.value.

Usage

```
plotVolcano(data, title = NA, subtitle = NA, n.fdr = FALSE, alpha = 0.05)
```

Arguments

data	DataFrame where each field is a database name with two fields for the estimate and p.value.
title	String representing the title label. Optional. (Default: NA)
subtitle	String representing the subtitle label. Optional. (Default: NA)
n.fdr	Integer corresponding to the number of comparisons made. Optional. (Default: NA).
alpha	Float representing the cut-off alpha value for the plot. Optional. (Default: 0.05)

Value

ggplot volcano plot

Examples

```
data=data.frame(estimate=c(runif(10)), p.value=c(runif(10)))
plotVolcano(data)
```

`predictAgeHorvath353` *Horvath 353 age predictor*

Description

The function takes a named numeric vector of beta values. The name attribute contains the probe ID (cg, ch or rs IDs). The function looks for overlapping probes and estimate age using Horvath aging model (Horvath 2013 Genome Biology). The function outputs a single numeric of age in years.

Usage

```
predictAgeHorvath353(betas)
```

Arguments

`betas`	a probeID-named vector of beta values

Value

age in years

Examples

```
betas <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
predictAgeHorvath353(betas)
```

`predictAgeSkinBlood` *Horvath Skin and Blood age predictor*

Description

The function takes a named numeric vector of beta values. The name attribute contains the probe ID (cg, ch or rs IDs). The function looks for overlapping probes and estimate age using Horvath aging model (Horvath et al. 2018 Aging, 391 probes). The function outputs a single numeric of age in years.

Usage

```
predictAgeSkinBlood(betas)
```

Arguments

betas	a probeID-named vector of beta values

Value

age in years

Examples

```
betas <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
predictAgeSkinBlood(betas)
```

---

predictMouseAgeInMonth

Mouse age predictor

Description

The function takes a named numeric vector of beta values. The name attribute contains the probe ID. The function looks for overlapping probes and estimate age using an aging model built from 321 MM285 probes. The function outputs a single numeric of age in months. The clock is most accurate with the sesame preprocessing.

Usage

```
predictMouseAgeInMonth(betas, na_fallback = TRUE)
```

Arguments

betas	a probeID-named vector of beta values
na_fallback	use the fallback default for NAs.

Value

age in month

Examples

```
betas = sesameDataGet('MM285.10.tissue')$betas
predictMouseAgeInMonth(betas[,1])
```

print.DMLSummary *Print DMLSummary object*

Description

Print DMLSummary object

Usage

```
## S3 method for class 'DMLSummary'
print(x, ...)
```

Arguments

x	a DMLSummary object
...	extra parameter for print

Value

print DMLSummary result on screen

Examples

```
sesameDataCache("HM450") # in case not done yet
data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:1000,], ~type, meta=data$sampleInfo)
smry
```

print.fileSet *Print a fileSet*

Description

Print a fileSet

Usage

```
## S3 method for class 'fileSet'
print(x, ...)
```

Arguments

x	a sesame::fileSet
...	stuff for print

Value

string representation

Examples

```
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))
fset
```

`print.sesameQC` *Print sesameQC object*

Description

Print sesameQC object

Usage

```
## S3 method for class 'sesameQC'
print(x, ...)
```

Arguments

`x`	a sesameQC object
`...`	extra parameter for print

Value

print sesameQC result on screen

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sesameQC(sdf)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

print.SigDF *Print SigDF object*

Description

Print SigDF object

Usage

```
## S3 method for class 'SigDF'
print(x, ...)
```

Arguments

x	a SigDF object
...	extra parameter for print

Value

print SigDF result on screen

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf = sesameDataGet('EPIC.1.SigDF')
sdf
```

probeID_designType *Extract the probe type field from probe ID This only works with the new probe ID system. See https://github.com/zhou-lab/InfiniumAnnotation for illustration*

Description

Extract the probe type field from probe ID This only works with the new probe ID system. See https://github.com/zhou-lab/InfiniumAnnotation for illustration

Usage

```
probeID_designType(Probe_ID)
```

Arguments

Probe_ID	Probe ID

Value

a vector of '1' and '2' suggesting Infinium-I and Infinium-II

Examples

```
probeID_designType("cg36609548_TC21")
```

`probeSuccessRate` *Whole-dataset-wide Probe Success Rate*

Description

This function calculates the probe success rate using pOOBAH detection p-values. Probes that has a detection p-value higher than a specific threshold are considered failed probes.

Usage

```
probeSuccessRate(sdf, mask = TRUE, max_pval = 0.05)
```

Arguments

`sdf`	a `SigDF`
`mask`	whether or not we count the masked probes in SigDF
`max_pval`	the maximum p-value to consider detection success

Value

a fraction number as probe success rate

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf = sesameDataGet('EPIC.1.SigDF')
probeSuccessRate(sdf)
```

qualityMask *Mask beta values by design quality*

Description

Currently quality masking only supports three platforms

Usage

```
qualityMask(
  sdf,
  mask.use.manifest = TRUE,
  manifest = NULL,
  mask.use.tcga = FALSE
)
```

Arguments

sdf	a SigDF object
mask.use.manifest	use manifest to mask probes
manifest	the manifest to use, for custom arrays
mask.use.tcga	whether to use TCGA masking, only applies to HM450

Value

a filtered SigDF

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sum(sdf$mask)
sum(qualityMask(sdf)$mask)
```

qualityRank *This function looks at public data of similar nature e.g., tissue, FFPE vs non-FFPE, etc to evaluate the quality of the target data quality*

Description

This function looks at public data of similar nature e.g., tissue, FFPE vs non-FFPE, etc to evaluate the quality of the target data quality

This function looks at public data of similar nature e.g., tissue, FFPE vs non-FFPE, etc to evaluate the quality of the target data quality

Usage

```
qualityRank(sdf, tissue = NULL, samplePrep = NULL, raw = FALSE)

qualityRank(sdf, tissue = NULL, samplePrep = NULL, raw = FALSE)
```

Arguments

sdf	a raw (unprocessed) SigDF
tissue	A string (blood,buccal and saliva)
samplePrep	FFPE, fresh, frozen
raw	to return the raw comparison table

Value

three numbers: 1. The number of public samples compared. 2. The fraction of public samples with more nondetection, and 3. The fraction of public samples with lower mean intensity 4. The higher the fraction, the better the sample.

three numbers: 1. The number of public samples compared. 2. The fraction of public samples with more nondetection, and 3. The fraction of public samples with lower mean intensity 4. The higher the fraction, the better the sample.

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
ranks <- qualityRank(sdf)


sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
ranks <- qualityRank(sdf)
```

---

readFileSet *Read an existing fileSet from storage*

Description

This function only reads the meta-data.

Usage

```
readFileSet(map_path)
```

Arguments

map_path	path of file to map (should contain valid _idx.rds index)

Value

a sesame::fileSet object

Examples

```
## create two samples
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))

## a hypothetical numeric array (can be beta values, intensities etc)
hypothetical <- setNames(runif(fset$n), fset$probes)

## map the numeric to file
mapFileSet(fset, 's1', hypothetical)

## read it from file
fset <- readFileSet('mybetas2')

## get data
sliceFileSet(fset, 's1', 'cg00000292')
```

---

readIDATpair *Import a pair of IDATs from one sample*

Description

The function takes a prefix string that are shared with _Grn.idat and _Red.idat. The function returns a SigDF.

Usage

```
readIDATpair(
  prefix.path,
  platform = "",
  manifest = NULL,
  controls = NULL,
  verbose = FALSE
)
```

Arguments

prefix.path	sample prefix without _Grn.idat and _Red.idat
platform	EPIC, HM450 and HM27 etc.
manifest	optional design manifest file
controls	optional control probe manifest file
verbose	be verbose? (FALSE)

Value

a SigDF

Examples

```
sdf <- readIDATpair(sub('_Grn.idat','',system.file(
    "extdata", "4207113116_A_Grn.idat", package = "sesameData")))
```

---

reIdentify *Re-identify IDATs by restoring scrambled SNP intensities*

Description

This requries setting a seed with a secret number that was used to de-identify the IDAT (see example). This requires a secret number that was used to de-idenitfy the IDAT

Usage

```
reIdentify(path, out_path = NULL, snps = NULL, mft = NULL)
```

Arguments

path	input IDAT file
out_path	output IDAT file
snps	SNP definition, if not given, default to SNP probes
mft	sesame-compatible manifest if non-standard

Value

NULL, changes made to the IDAT files

Examples

```
temp_out <- tempfile("test")

set.seed(123)
reIdentify(system.file(
    "extdata", "4207113116_A_Grn.idat", package = "sesameData"), temp_out)
unlink(temp_out)
```

reopenSesame *re-compute beta value for GenomicRatioSet*

Description

re-compute beta value for GenomicRatioSet

Usage

```
reopenSesame(x, naFrac = 0.2)
```

Arguments

x	GenomicRatioSet
naFrac	maximum NA fraction for a probe before it gets dropped (1)

Value

a GenomicRatioSet

resetMask *Reset Masking*

Description

Reset Masking

Usage

```
resetMask(sdf)
```

Arguments

sdf	a SigDF

Value

a new SigDF with mask reset to all FALSE

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sum(sdf$mask)
sdf <- addMask(sdf, c("cg14057072", "cg22344912"))
sum(sdf$mask)
sum(resetMask(sdf)$mask)
```

RGChannelSetToSigDFs *Convert RGChannelSet (minfi) to a list of SigDF (SeSAMe)*

Description

Notice the colData() and rowData() is lost. Most cases, rowData is empty anyway.

Usage

```
RGChannelSetToSigDFs(rgSet, manifest = NULL, BPPARAM = SerialParam())
```

Arguments

rgSet	a minfi::RGChannelSet
manifest	manifest file
BPPARAM	get parallel with MulticoreParam(n)

Value

a list of sesame::SigDF

Examples

```
if (FALSE) { # to avoid excessive memory usage in package builder
if (require(FlowSorted.Blood.450k)) {
    rgSet <- FlowSorted.Blood.450k[,1:2]
    sdfs <- RGChannelSetToSigDFs(rgSet)
}
}
```

scrub *SCRUB background correction*

Description

This function takes a SigDF and returns a modified SigDF with background subtracted. scrub subtracts residual background using background median

Usage

```
scrub(sdf)
```

Arguments

sdf	a SigDF

Details

This function is meant to be used after noob.

Value

a new SigDF with noob background correction

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.nb <- noob(sdf)
sdf.nb.scrub <- scrub(sdf.nb)
```

---

scrubSoft *SCRUB background correction*

Description

This function takes a SigDF and returns a modified SigDF with background subtracted. scrubSoft subtracts residual background using a noob-like procedure.

Usage

```
scrubSoft(sdf)
```

Arguments

sdf	a SigDF

Details

This function is meant to be used after noob.

Value

a new SigDF with noob background correction

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.nb <- noob(sdf)
sdf.nb.scrubSoft <- scrubSoft(sdf.nb)
```

sdfPlatform *Convenience function to output platform attribute of SigDF*

Description

Convenience function to output platform attribute of SigDF

Usage

```
sdfPlatform(sdf)
```

Arguments

sdf	a SigDF object

Value

the platform string for the SigDF object

Examples

```
sesameDataCache("EPIC")
sdf = sesameDataGet('EPIC.1.SigDF')
sdfPlatform(sdf)
```

sdf_read_table *read a table file to SigDF*

Description

read a table file to SigDF

Usage

```
sdf_read_table(fname, platform = NULL, ...)
```

Arguments

fname	file name
platform	array platform (will infer if not given)
...	additional argument to read.table

Value

read table file to SigDF

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf = sesameDataGet('EPIC.1.SigDF')
fname = sprintf("%s/sigdf.txt", tempdir())
sdf_write_table(sdf, file=fname)
sdf2 = sdf_read_table(fname)
```

---

sdf_write_table *write SigDF to table file*

Description

write SigDF to table file

Usage

```
sdf_write_table(sdf, ...)
```

Arguments

sdf	the SigDF to output
...	additional argument to write.table

Value

write SigDF to table file

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf = sesameDataGet('EPIC.1.SigDF')
sdf_write_table(sdf, file=sprintf("%s/sigdf.txt", tempdir()))
```

---

searchIDATprefixes *Identify IDATs from a directory*

Description

The input is the directory name as a string. The function identifies all the IDAT files under the directory. The function returns a vector of such IDAT prefixes under the directory.

Usage

```
searchIDATprefixes(dir.name, recursive = TRUE, use.basename = TRUE)
```

Arguments

dir.name	the directory containing the IDAT files.
recursive	search IDAT files recursively
use.basename	basename of each IDAT path is used as sample name This won't work in rare situation where there are duplicate IDAT files.

Value

the IDAT prefixes (a vector of character strings).

Examples

```
## only search what are directly under
IDATprefixes <- searchIDATprefixes(
    system.file("extdata", "", package = "sesameData"))

## search files recursively is by default
IDATprefixes <- searchIDATprefixes(
    system.file(package = "sesameData"), recursive=TRUE)
```

---

segmentBins *Segment bins using DNAcopy*

Description

Segment bins using DNAcopy

Usage

```
segmentBins(bin.signals, bin.coords)
```

Arguments

bin.signals	bin signals (input)
bin.coords	bin coordinates

Value

segment signal data frame

sesamePlotIntensVsBetas *Plot Total Signal Intensities vs Beta Values This plot is helpful in revealing the extent of signal background and dye bias.*

Description

Plot Total Signal Intensities vs Beta Values This plot is helpful in revealing the extent of signal background and dye bias.

Usage

```
sesamePlotIntensVsBetas(sdf, mask = TRUE, intens.range = c(5, 15), ...)
```

Arguments

sdf	a SigDF
mask	whether to remove probes that are masked
intens.range	plot range of signal intensity
...	additional arguments to smoothScatter

Value

create a total signal intensity vs beta value plot

Examples

```
sesameDataCache("EPIC")
sdf <- # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sesamePlotIntensVsBetas(sdf)
```

sesamePlotRedGrnQQ *Plot red-green QQ-Plot using Infinium-I Probes*

Description

Plot red-green QQ-Plot using Infinium-I Probes

Usage

```
sesamePlotRedGrnQQ(sdf)
```

Arguments

sdf	a SigDF

Value

create a qqplot

Examples

```
sesameDataCache("EPIC")  # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sesamePlotRedGrnQQ(sdf)
```

`sesameQC` *Generate summary numbers that indicative of experiment quality Please provide a raw SigDF(before any preprocessing). Usually directly from readIDATpair*

Description

Generate summary numbers that indicative of experiment quality Please provide a raw SigDF(before any preprocessing). Usually directly from readIDATpair

Usage

```
sesameQC(sdf)
```

Arguments

`sdf`	a `SigDF` object

Value

a sesameQC class object

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sesameQC(sdf)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

sesamize *"fix" an RGChannelSet (for which IDATs may be unavailable) with Sesame The input is an RGSet and the output is a sesamized minfi::GenomicRatioSet*

Description

HDF5Array package required.

Usage

```
sesamize(
  rgSet,
  naFrac = 1,
  BPPARAM = SerialParam(),
  HDF5 = NULL,
  HDF5SEdestination = paste0(tempdir(check = TRUE), "/sesamize_HDF5_scratch"),
  replace = FALSE
)
```

Arguments

rgSet	an RGChannelSet, perhaps with colData of various flavors
naFrac	maximum NA fraction for a probe before it gets dropped (1)
BPPARAM	get parallel with MulticoreParam(n)
HDF5	is the rgSet HDF5-backed? if so, avoid eating RAM (perhaps)
HDF5SEdestination	character(1) path to where the HDF5-backed GenomicRatioSet will be stored
replace	logical(1) passed to saveHDF5SummarizedExperiment

Value

a sesamized GenomicRatioSet

Note

We employ BPREDO for a second chance if bplapply hits an error.

Examples

```
if(FALSE) {
    library(FlowSorted.CordBloodNorway.450k)
    sesamize(FlowSorted.CordBloodNorway.450k[,1:2],
        BPPARAM=MulticoreParam(2))
}
```

setMask *Set mask to only the probes specified*

Description

Set mask to only the probes specified

Usage

```
setMask(sdf, probes)
```

Arguments

sdf	a SigDF
probes	a vector of probe IDs or a logical vector with TRUE representing masked probes

Value

a SigDF with added mask

Examples

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sum(sdf$mask)
sum(setMask(sdf, "cg14959801")$mask)
sum(setMask(sdf, c("cg14057072", "cg22344912"))$mask)
```

setMaskBySpecies *Set mask using species-specific manifest*

Description

Set mask using species-specific manifest

Usage

```
setMaskBySpecies(sdf, species = "homo_sapiens")
```

Arguments

sdf	a SigDF
species	the species the sample is considered to be

Value

a SigDF with updated color channel and mask

Examples

```
sdf = sesameDataGet('Mammal40.1.SigDF')
sdf_mouse = setMaskBySpecies(sdf, "mus_musculus")
```

SigDF *SigDF constructor from a plain data frame*

Description

SigDF constructor from a plain data frame

Usage

```
SigDF(df, platform = "EPIC", ctl = NULL)
```

Arguments

df	a data.frame with Probe_ID, MG, MR, UG, UR, col and mask
platform	a string to specify the array platform
ctl	optional control probe data frame

Value

a SigDF object

Examples

```
sesameDataCache("EPIC") # if not done yet
df <- as.data.frame(sesameDataGet('EPIC.1.SigDF'))
```

SigDFsToRGChannelSet *Convert sesame::SigDF to minfi::RGChannelSet*

Description

Convert sesame::SigDF to minfi::RGChannelSet

Usage

```
SigDFsToRGChannelSet(sdfs, BPPARAM = SerialParam(), annotation = NA)
```

Arguments

sdfs	a list of sesame::SigDF
BPPARAM	get parallel with MulticoreParam(n)
annotation	the minfi annotation string, guessed if not given

Value

a minfi::RGChannelSet

Examples

```
if (FALSE) { # to avoid excessive memory usage in package builder
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
rgSet <- SigDFsToRGChannelSet(sdf)
}
```

SigDFToRatioSet *Convert one sesame::SigDF to minfi::RatioSet*

Description

Convert one sesame::SigDF to minfi::RatioSet

Usage

```
SigDFToRatioSet(sdf, annotation = NA)
```

Arguments

sdf	a sesame::SigDF
annotation	minfi annotation string

Value

a minfi::RatioSet

Examples

```
if (FALSE) { # to avoid excessive memory usage in package builder
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
ratioSet <- SigDFToRatioSet(sdf)
}
```

signalMU *report M and U for regular probes*

Description

report M and U for regular probes

Usage

```
signalMU(sdf, mask = TRUE)
```

Arguments

sdf	a SigDF
mask	whether to apply mask

Value

a data frame of M and U columns

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
head(signalMU(sdf))
```

sliceFileSet *Slice a fileSet with samples and probes*

Description

Slice a fileSet with samples and probes

Usage

```
sliceFileSet(fset, samples = fset$samples, probes = fset$probes, memmax = 10^5)
```

Arguments

fset	a sesame::fileSet, as obtained via readFileSet
samples	samples to query (default to all samples)
probes	probes to query (default to all probes)
memmax	maximum items to read from file to memory, to protect from accidental memory congestion.

Value

a numeric matrix of length(samples) columns and length(probes) rows

Examples

```
## create two samples
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))

## a hypothetical numeric array (can be beta values, intensities etc)
hypothetical <- setNames(runif(fset$n), fset$probes)

## map the numeric to file
mapFileSet(fset, 's1', hypothetical)

## get data
sliceFileSet(fset, 's1', 'cg00000292')
```

SNPcheck *Check sample identity using SNP probes*

Description

Check sample identity using SNP probes

Usage

```
SNPcheck(betas)
```

Arguments

betas	numeric matrix (row: probes, column: samples)

Value

grid object plotting SNP clustering

Examples

```
betas <- sesameDataGet('HM450.10.TCGA.PAAD.normal')
SNPcheck(betas)
```

summaryExtractTest *Extract slope information from DMLSummary*

Description

Extract slope information from DMLSummary

Usage

```
summaryExtractTest(smry)
```

Arguments

smry	DMLSummary from DML command

Value

a table of slope and p-value

Examples

```
sesameDataCache("HM450") # in case not done yet
data = sesameDataGet('HM450.76.TCGA.matched')
smry = DML(data$betas[1:1000,], ~type, meta=data$sampleInfo)
slopes = summaryExtractTest(smry)
```

testEnrichment *testEnrichment tests for the enrichment of set of probes (query set) in a number of features (database sets).*

Description

testEnrichment tests for the enrichment of set of probes (query set) in a number of features (database sets).

Usage

```
testEnrichment(
  querySet,
  databaseSets = NA,
  universeSet = NA,
  platform = NA,
  estimate.type = "ES",
  p.value.adj = FALSE,
  n.fdr = NA,
  return.meta = FALSE,
  verbose = FALSE
)
```

Arguments

querySet	Vector of probes of interest (e.g., significant probes)
databaseSets	List of vectors corresponding to the database sets of interest with associated meta data as an attribute to each element. Optional. (Default: NA)
universeSet	Vector of probes in the universe set containing all of the probes to be considered in the test. If it is not provided, it will be inferred from the provided platform. (Default: NA).
platform	String corresponding to the type of platform to use. Either MM285, EPIC, HM450, or HM27. If it is not provided, it will be inferred from the query set probeIDs (Default: NA).
estimate.type	String indicating the estimate to report. (Default: "ES")
p.value.adj	Logical value indicating whether to report the adjusted p-value. (Default: FALSE).
n.fdr	Integer corresponding to the number of comparisons made. Optional. (Default: NA).
return.meta	Logical value indicating whether to return meta data columns for those database sets containing sparse meta data information.
verbose	Logical value indicating whether to display intermediate text output about the type of test. Optional. (Default: FALSE).

Value

One list containing features corresponding the test estimate, p-value, and type of test.

Examples

```
library(SummarizedExperiment)
databaseSetNames = c('KYCG.MM285.seqContextN.20210630',
'KYCG.MM285.designGroup.20210210')
databaseSets = do.call(c, lapply(databaseSetNames, sesameDataGet))
MM285.tissueSignature = sesameDataGet('MM285.141.SE.tissueSignature')
df = rowData(MM285.tissueSignature)
querySet = df$Probe_ID[df$branch == "E-Brain"]
testEnrichment(querySet=querySet, databaseSets=databaseSets, verbose=FALSE)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

---

testEnrichment1 *testEnrichment1 tests for the enrichment of set of probes (query set) in a single given feature (database set)*

Description

testEnrichment1 tests for the enrichment of set of probes (query set) in a single given feature (database set)

Usage

```
testEnrichment1(
  querySet,
  databaseSet,
  universeSet,
  estimate.type = "ES",
  p.value.adj = FALSE,
  verbose = FALSE
)
```

Arguments

querySet	Vector of probes of interest (e.g., significant probes)
databaseSet	Vector corresponding to the database sets of interest with associated meta data as an attribute to each element.
universeSet	Vector of probes in the universe set containing all of the probes to be considered in the test.
estimate.type	String indicating the estimate to report. (Default: "ES")
p.value.adj	Logical value indicating whether to report the adjusted p-value. (Default: FALSE)
verbose	Logical value indicating whether to display intermediate text output about the type of test. Optional. (Default: FALSE)

Value

One list containing features corresponding the test estimate, p-value, and type of test.

---

testEnrichmentFGSEA *testEnrichmentFGSEA uses the FGSEA test to estimate the association of a categorical variable against a continuous variable.*

Description

testEnrichmentFGSEA uses the FGSEA test to estimate the association of a categorical variable against a continuous variable.

Usage

```
testEnrichmentFGSEA(
  querySet,
  databaseSet,
  p.value.adj = FALSE,
  estimate.type = "ES"
)
```

Arguments

querySet	Vector of probes of interest (e.g., significant probes)
databaseSet	Vector of probes corresponding to a single database set of interest.
p.value.adj	Logical value indicating whether to report the adjusted p-value. (Default: FALSE).
estimate.type	String indicating the estimate to report. Optional. (Default: "ES").

Value

A DataFrame with the estimate/statistic, p-value, and name of test for the given results.

---

testEnrichmentFisher *testEnrichmentFisher uses Fisher's exact test to estimate the association between two categorical variables.*

Description

testEnrichmentFisher uses Fisher's exact test to estimate the association between two categorical variables.

Usage

```
testEnrichmentFisher(querySet, databaseSet, universeSet)
```

Arguments

querySet	Vector of probes of interest (e.g., significant probes)
databaseSet	Vectors corresponding to the database set of interest with associated meta data as an attribute to each element.
universeSet	Vector of probes in the universe set containing all of the probes to be considered in the test. (Default: NULL)

Value

A DataFrame with the estimate/statistic, p-value, and name of test for the given results.

`testEnrichmentGene` *testEnrichmentGene tests for the enrichment of set of probes (querySet) in gene regions.*

Description

testEnrichmentGene tests for the enrichment of set of probes (querySet) in gene regions.

Usage

```
testEnrichmentGene(querySet, databaseSets = NA, platform = NA, verbose = FALSE)
```

Arguments

`querySet`	Vector of probes of interest (e.g., probes belonging to a given platform)
`databaseSets`	List of vectors corresponding to the database sets of interest with associated meta data as an attribute to each element. Optional. (Default: NA)
`platform`	String corresponding to the type of platform to use. Either MM285, EPIC, HM450, or HM27. If it is not provided, it will be inferred from the query set querySet (Default: NA)
`verbose`	Logical value indicating whether to display intermediate text output about the type of test. Optional. (Default: FALSE)

Value

One list containing features corresponding the test estimate, p-value, and type of test.

Examples

```
library(SummarizedExperiment)
MM285.tissueSignature = sesameDataGet('MM285.141.SE.tissueSignature')
df = rowData(MM285.tissueSignature)
querySet = df$Probe_ID[df$branch == "E-Brain"]
testEnrichmentGene(querySet, platform="MM285", verbose=FALSE)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

testEnrichmentSpearman

testEnrichmentSpearman uses the Spearman statistical test to estimate the association between two continuous variables.

Description

testEnrichmentSpearman uses the Spearman statistical test to estimate the association between two continuous variables.

Usage

```
testEnrichmentSpearman(querySet, databaseSet)
```

Arguments

querySet	Vector of probes of interest (e.g., significant probes)
databaseSet	List of vectors corresponding to the database set of interest with associated meta data as an attribute to each element.

Value

A DataFrame with the estimate/statistic, p-value, and name of test for the given results.

totalIntensities *M+U Intensities Array*

Description

The function takes one single SigDF and computes total intensity of all the in-band measurements by summing methylated and unmethylated alleles. This function outputs a single numeric for the mean.

Usage

```
totalIntensities(sdf, mask = FALSE)
```

Arguments

sdf	a SigDF
mask	whether to mask probes using mask column

Value

a vector of M+U signal for each probe

Examples

```
sesameDataCache("EPIC") # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
intensities <- totalIntensities(sdf)
```

`twoCompsEst2` *Estimate the fraction of the 2nd component in a 2-component mixture*

Description

Estimate the fraction of the 2nd component in a 2-component mixture

Usage

```
twoCompsEst2(
  pop1,
  pop2,
  target,
  use.ave = TRUE,
  diff_1m2u = NULL,
  diff_1u2m = NULL
)
```

Arguments

`pop1`	Reference methylation level matrix for population 1
`pop2`	Reference methylation level matrix for population 2
`target`	Target methylation level matrix to be analyzed
`use.ave`	use population average in selecting differentially methylated probes
`diff_1m2u`	A vector of differentially methylated probes (methylated in population 1 but unmethylated in population 2)
`diff_1u2m`	A vector of differentially methylated probes (unmethylated in population 1 but methylated in population 2)

Value

Estimate of the 2nd component in the 2-component mixture

`visualizeGene` *Visualize Gene*

Description

Visualize the beta value in heatmaps for a given gene. The function takes a gene name which is taken from the UCSC refGene. It searches all the transcripts for the given gene and optionally extend the span by certain number of base pairs. The function also takes a beta value matrix with sample names on the columns and probe names on the rows. The function can also work on different genome builds (default to hg38, can be hg19).

Usage

```
visualizeGene(
  geneName,
  betas,
  platform = c("EPIC", "HM450", "MM285"),
  upstream = 2000,
  dwstream = 2000,
  refversion = c("hg38", "hg19", "mm10"),
  ...
)
```

Arguments

`geneName`	gene name
`betas`	beta value matrix (row: probes, column: samples)
`platform`	HM450, EPIC, or MM285 (default)
`upstream`	distance to extend upstream
`dwstream`	distance to extend downstream
`refversion`	hg19, hg38, or mm10 (default)
`...`	additional options, see visualizeRegion

Value

None

Examples

```
betas <- sesameDataGet('HM450.76.TCGA.matched')$betas
visualizeGene('ADA', betas, 'HM450')
```

`visualizeProbes` *Visualize Region that Contains the Specified Probes*

Description

Visualize the beta value in heatmaps for the genomic region containing specified probes. The function works only if specified probes can be spanned by a single genomic region. The region can cover more probes than specified. Hence the plotting heatmap may encompass more probes. The function takes as input a string vector of probe IDs (cg/ch/rs-numbers). if draw is FALSE, the function returns the subset beta value matrix otherwise it returns the grid graphics object.

Usage

```
visualizeProbes(
  probeNames,
  betas,
  platform = c("EPIC", "HM450", "MM285"),
  refversion = c("hg38", "hg19", "mm10"),
  upstream = 1000,
  dwstream = 1000,
  ...
)
```

Arguments

`probeNames`	probe names
`betas`	beta value matrix (row: probes, column: samples)
`platform`	HM450, EPIC or MM285 (default)
`refversion`	hg19, hg38 or mm10 (default)
`upstream`	distance to extend upstream
`dwstream`	distance to extend downstream
`...`	additional options, see visualizeRegion

Value

None

Examples

```
betas <- sesameDataGet('HM450.76.TCGA.matched')$betas
visualizeProbes(c('cg22316575', 'cg16084772', 'cg20622019'), betas, 'HM450')
```

`visualizeRegion` *Visualize Region*

Description

The function takes a genomic coordinate (chromosome, start and end) and a beta value matrix (probes on the row and samples on the column). It plots the beta values as a heatmap for all probes falling into the genomic region. If 'draw=TRUE' the function returns the plotted grid graphics object. Otherwise, the selected beta value matrix is returned. 'cluster.samples=TRUE/FALSE' controls whether hierarchical clustering is applied to the subset beta value matrix.

Usage

```
visualizeRegion(
  chrm,
  plt.beg,
  plt.end,
  betas,
  platform = c("EPIC", "HM450", "MM285"),
  refversion = c("hg38", "hg19", "mm10"),
  sample.name.fontsize = 10,
  heat.height = NULL,
  draw = TRUE,
  show.sampleNames = TRUE,
  show.samples.n = NULL,
  show.probeNames = TRUE,
  cluster.samples = FALSE,
  nprobes.max = 1000,
  na.rm = FALSE,
  dmin = 0,
  dmax = 1
)
```

Arguments

`chrm`	chromosome
`plt.beg`	begin of the region
`plt.end`	end of the region
`betas`	beta value matrix (row: probes, column: samples)
`platform`	EPIC, HM450, or MM285
`refversion`	hg38, hg19, or mm10
`sample.name.fontsize`	sample name font size
`heat.height`	heatmap height (auto inferred based on rows)
`draw`	draw figure or return betas

show.sampleNames	whether to show sample names
show.samples.n	number of samples to show (default: all)
show.probeNames	whether to show probe names
cluster.samples	whether to cluster samples
nprobes.max	maximum number of probes to plot
na.rm	remove probes with all NA.
dmin	data min
dmax	data max

Value

graphics or a matrix containing the captured beta values

Examples

```
betas <- sesameDataGet('HM450.76.TCGA.matched')$betas
visualizeRegion('chr20', 44648623, 44652152, betas, 'HM450')
```

---

visualizeSegments *Visualize segments*

Description

The function takes a CNSegment object obtained from cnSegmentation and plot the bin signals and segments (as horizontal lines).

Usage

```
visualizeSegments(seg, to.plot = NULL)
```

Arguments

seg	a CNSegment object
to.plot	chromosome to plot (by default plot all chromosomes)

Details

require ggplot2, scales

Value

plot graphics

Examples

```
sesameDataCache("EPIC") # in case not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdfs.normal <- sesameDataGet('EPIC.5.SigDF.normal')[1:3]
seg <- cnSegmentation(sdf, sdfs.normal)

visualizeSegments(seg)

# release memory for Windows package builder
rm(list=ls(env=sesameData:::cacheEnv), envir=sesameData:::cacheEnv)
gc()
```

Index

∗ **DNAMethylation**
    sesame-package, 5
∗ **Microarray**
    sesame-package, 5
∗ **QualityControl**
    sesame-package, 5
_PACKAGE *(*sesame-package*)*, 5

addMask, 5
as.data.frame.sesameQC, 6
attachManifest, 7

BetaValueToMValue, 7
binSignals, 8
bisConversionControl, 8
bSubComplete, 9
bSubMostVariable, 9
bSubProbes, 10

calcDatabaseSetStatistics1, 11
calcDatabaseSetStatisticsAll, 11
checkLevels, 12
chipAddressToSignal, 13
cnSegmentation, 13
compareDatbaseSetOverlap, 14
compareMouseStrainReference, 15
compareMouseTissueReference, 16
controls, 16
createDatabaseSetNetwork, 17
createUCSCtrack, 18

deIdentify, 18
detectionPnegEcdf, 19
detectionPoobEcdf, 20
detectionPoobEcdf2, 21
diffRefSet, 22
dmContrasts, 22
DML, 23
DMR, 24
dyeBiasCorr, 25
dyeBiasCorrMostBalanced, 26
dyeBiasCorrTypeINorm, 26
dyeBiasDistortion, 27
dyeBiasNL *(*dyeBiasCorrTypeINorm*)*, 26

estimateCellComposition, 28
estimateLeukocyte, 28
extractDesign, 29

formatVCF, 30

getAFTypeIbySumAlleles, 31
getAutosomeProbes, 31
getBetas, 32
getBinCoordinates, 33
getDatabaseSetOverlap, 33
getNormCtls, 34
getProbesByChromosome, 35
getProbesByGene, 35
getProbesByRegion, 36
getProbesByTSS, 37
getRefSet, 38
getSexInfo, 38

inferEthnicity, 39
inferInfiniumIChannel, 40
inferSex, 40
inferSexKaryotypes, 41
inferSpecies, 42
inferStrain, 43
inferTissue, 43
initFileSet, 44
isUniqProbeID, 45

mapFileSet, 46
meanIntensity, 46
medianTotalIntensity, 47
MValueToBetaValue, 48

neob, 48
noMasked, 49

noob, 49

openSesame, 50
openSesameToFile, 51

plotLollipop, 51
plotVolcano, 52
pOOBAH *(*detectionPoobEcdf*)*, 20
pOOBAH2 *(*detectionPoobEcdf2*)*, 21
predictAgeHorvath353, 53
predictAgeSkinBlood, 53
predictMouseAgeInMonth, 54
print.DMLSummary, 55
print.fileSet, 55
print.sesameQC, 56
print.SigDF, 57
probeID_designType, 57
probeSuccessRate, 58

qualityMask, 59
qualityRank, 59

readFileSet, 60
readIDATpair, 61
reIdentify, 62
reopenSesame, 63
resetMask, 63
RGChannelSetToSigDFs, 64

scrub, 64
scrubSoft, 65
sdf_read_table, 66
sdf_write_table, 67
sdfPlatform, 66
searchIDATprefixes, 67
segmentBins, 68
sesame *(*sesame-package*)*, 5
sesame-package, 5
sesamePlotIntensVsBetas, 69
sesamePlotRedGrnQQ, 69
sesameQC, 70
sesamize, 71
setMask, 72
setMaskBySpecies, 72
SigDF, 73
SigDFsToRGChannelSet, 73
SigDFToRatioSet, 74
signalMU, 75
sliceFileSet, 75
SNPcheck, 76
summaryExtractTest, 77

testEnrichment, 77
testEnrichment1, 78
testEnrichmentFGSEA, 79
testEnrichmentFisher, 80
testEnrichmentGene, 81
testEnrichmentSpearman, 82
totalIntensities, 82
twoCompsEst2, 83

visualizeGene, 84
visualizeProbes, 85
visualizeRegion, 86
visualizeSegments, 87