Package 'ChIPsim'

November 11, 2026

Type Package

Title Simulation of ChIP-seq experiments

Version 1.66.0

Date 2011-05-18

Author Peter Humburg

Maintainer Peter Humburg <Peter.Humburg@gmail.com>

Description A general framework for the simulation of ChIP-seq data. Although currently focused on nucleosome positioning the package is designed to support different types of experiments.

License GPL (>= 2)

LazyLoad yes

Depends Biostrings (>= 2.29.2)

Imports IRanges, XVector, Biostrings, ShortRead, graphics, methods, stats, utils

Suggests actuar, zoo

biocViews Infrastructure, ChIPSeq

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

git_branch RELEASE_3_23

git_last_commit 324ae9c

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

ChIPsim-package 2
bindDens2readDens 3
decodeQuality 4
defaultControl 5
defaultErrorProb 6

defaultFunctions . . . 7
defaultGenerator . . . 8
distDens . . . 9
extractQuality . . . 10
feat2dens . . . 12
featureDensity . . . 13
FeatureGenerators . . . 14
internal . . . 15
joinRegion . . . 16
makeFeatures . . . 16
placeFeatures . . . 18
pos2fastq . . . 20
readError . . . 21
readQualitySample . . . 22
readSequence . . . 23
reconcileFeatures . . . 24
sampleReads . . . 25
simChIP . . . 26
simpleNames . . . 28
writeFASTQ . . . 29
writeReads . . . 30

Index **31**

`ChIPsim-package` *Simulation of ChIP-seq experiments*

Description

This package provides a framework for the simulation of ChIP-seq experiments. An implementation of a simulation for nucleosome positioning experiments is provided as part of the package. Simulations for other experiments can be implemented using the provided framework.

Details

Package:	ChIPsim
Type:	Package
Version:	1.3.1
Date:	2010-07-30
License:	GPL (>= 2)
LazyLoad:	yes
Depends:	Biostrings
Imports:	IRanges, ShortRead
Suggests:	actuar, zoo

Function simChIP is the main driver of the simulation. To simulate different types of experiments the functions passed to the functions argument of simChIP have to be replaced. See the vignettes for more detail.

Author(s)

Peter Humburg

Maintainer: Peter Humburg <Peter.Humburg@well.ox.ac.uk>

References

~~ Literature or other references for background information ~~

See Also

ShortRead and its dependencies are used to handle short read and genomic sequences.

Examples

```
## See the accompanying vignette 'Introduction to ChIPsim' for a detailed
## example of how to use this package for nucleosome positioning simulations.
## A guide for the writing of extensions that cover other types of
## experiments is provided in 'Extending ChIPsim'.
```

---

bindDens2readDens *Convert a feature density into a read density*

Description

Given a feature density this function produces two read densities, one for each strand.

Usage

```
bindDens2readDens(bindDens, fragment, nfrag = 1e+05, bind = 147,
 minLength = 150, maxLength = 180, ...)
```

Arguments

bindDens	Numeric vector with the feature density for one chromosome.
fragment	Function giving the fragment length distribution.
nfrag	Number of fragments that should be simulated to generate the read distribution.
bind	Length of binding site.
minLength	Minimum fragment length.
maxLength	Maximum fragment length.
...	Further arguments to fragment.

Value

A two column matrix. The first column contains the read density for the forward strand, the second column the read density for the reverse strand.

Author(s)

Peter Humburg

See Also

feat2dens, sampleReads

Examples

```
set.seed(1)
## generate a (relatively short) sequence of nucleosome features
features <- placeFeatures(start=200, length=1e5)

## calculate feature density
featureDens <- feat2dens(features, length=1e5)

## convert to read density
readDens <- bindDens2readDens(featureDens, fragDens, meanLength=160)
```

---

decodeQuality *Conversion between numerical and ASCII representation of read qualities*

Description

These functions convert an ASCII encoded sequence of read qualities into a numeric vector of error probabilities and vice versa.

Usage

```
decodeQuality(quality, type = c("Illumina", "Sanger", "Solexa"))
encodeQuality(quality, type = c("Illumina", "Sanger", "Solexa"))
```

Arguments

quality	For decodeQuality a character string representing the read qualities for a single sequence read. For encodeQuality a numeric vector of error probabilities.
type	Type of encoding to use.

Details

See extractQuality for a description of the currently supported encodings.

Value

Either a numeric vector of error probabilities or a character string of encoded read quality scores. Each entry in the vector corresponds to one character in the input.

Author(s)

Peter Humburg

See Also

extractQuality

Examples

```
## decodeQuality and encodeQualty are the inverse operations
## of each other as one might expect
quality <- "IIIIIIIIIIIICIIGIIIIGII95III6II-II0"
errorProb <- decodeQuality(quality, type="Sanger")
qualitySanger <- encodeQuality(errorProb, type="Sanger")
all.equal(quality, qualitySanger)

## They can also be used to convert between encodings
qualityIllumina <- encodeQuality(errorProb, type="Illumina")
```

---

defaultControl *Default parameters for simChIP*

Description

Produces a list of parameters for each of the functions used to carry out the various stages of the simulation.

Usage

```
defaultControl(features = list(), bindDensity = list(),
 readDensity = list(), readNames = list(), readSequence = list())
```

Arguments

features	Parameters for feature generation.
bindDensity	Parameters for the conversion of feature sequence into binding site densities.
readDensity	Parameters for the conversion of binding site densities into read densities. Always provides parameters fragment Default: fragDens meanLength Default: 160
readNames	Parameters for the generation of read names.

`readSequence` Parameters for the conversion of read positions into read sequences. Always provides parameters

`qualityFun` `readQualitySample`

`errorFun` `readError`

`readLen` 36

Details

Any parameters passed as part of list to one of the arguments of `defaultControl` will be passed on to the corresponding function in `simChIP`. The build-in defaults can be overwritten by providing a list entry with the same name.

Value

List of parameters for use as the `control` argument to `simChIP`.

Author(s)

Peter Humburg

See Also

`defaultFunctions`, `simChIP`

Examples

```
defaultControl()
defaultControl(features=list(maxTail=0), readSequence=list(readLen=50))
```

`defaultErrorProb` *Replacement probabilities for sequencing errors*

Description

For each nucleotide this function provides probabilities indicating how likely it is to be replaced by any of the other nucleotides should a sequencing error occur.

Usage

```
defaultErrorProb()
```

Details

The probabilities used here are the ones determined by Dohm *et al.* for *Beta vulgaris*. They should be more appropriate than a uniform distribution but the use of species specific rates is recommended where available.

Value

A list of four vectors with replacement probabilities for each nucleotide.

Author(s)

Peter Humburg

References

Juliane C. Dohm, Claudio Lottaz, Tatiana Borodina, and Heinz Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucl. Acids Res.*, pages gkn425+, July 2008.

Examples

```
defaultErrorProb()
```

---

defaultFunctions *Default functions for simChIP*

Description

Provides default functions to carry out the different stages of the ChIP-seq simulation.

Usage

```
defaultFunctions()
```

Value

A list with components

features	placeFeatures
bindDensity	feat2dens
readDensity	bindDens2readDens
sampleReads	sampleReads
readSequence	writeReads
readNames	simpleNames

Author(s)

Peter Humburg

See Also

simChIP

Examples

```
defaultFunctions()
```

`defaultGenerator` *Defaults for Feature Generator*

Description

Functions to generate defaults for `makeFeatures`.

Usage

```
defaultGenerator()
defaultTransition()
defaultInit(prob=c(0.2, 0.05, 0, 0.25, 0.5),
 states=c("ReversePhasedFeature", "StableFeature",
  "PhasedFeature", "NFRFeature", "FuzzyFeature"))
defaultLastFeat(isEnd = c(FALSE, rep(TRUE, 4)),
 states = c("ReversePhasedFeature", "StableFeature",
  "PhasedFeature", "NFRFeature", "FuzzyFeature"))
```

Arguments

`prob`	Numeric vector giving the initial state distribution. This will be normalised if the probabilities do not add up to 1.
`isEnd`	Logical vector indicating which states, i.e. features, are allowed to be last in the sequence.
`states`	Character vector of state names.

Details

These functions generate data structures that can be passed as arguments to `makeFeatures`. Using this set of functions will create a nucleosome positioning simulation. Some of the defaults can be modified by passing different values to `defaultInit` and `defaultLastFeat`.

Value

Return values are suitable as arguments `generator`, `transition`, `init` and `lastFeat` of `makeFeatures`. See the documentation of `makeFeatures` for more detail.

Author(s)

Peter Humburg

Examples

```
set.seed(1)
## generate defaults
generator <- defaultGenerator()
transition <- defaultTransition()
lastFeat <- defaultLastFeat()

## change the initial state distribution such that it
## always starts with a fuzzy feature
init <- defaultInit(c(0, 0, 0, 0, 1))

## now generate some features for a stretch of 1 million base pairs
features <- makeFeatures(generator=generator, transition=transition,
 lastFeat=lastFeat, init=init, length=1e6)
```

`distDens` *Computing densities for nucleosome positioning simulation*

Description

These functions compute nucleosome densities for a given parameter set (usually provided through one of the feature classes).

Usage

```
distDens(x, minDist = 175, varDist = 337.5, meanDist = 200)
fragDens(x, minLength, maxLength, meanLength, bind)
indNuc(meanDist = 200, length = 2000, weight = 1)
noNuc(length, weight = 1)
stableDens(x, shift = 10, ratio = 1, weight = 1, stability = 1)
phaseNuc(stable, dist, minDist = 175, length = 2000, meanDist = 200,
 varDist = (meanDist - minDist) + (meanDist - minDist)^2/2,
 shift = 10, ratio = 1, weight = 1, stability = 1)
bindLocDens(x, fragLength)
```

Arguments

`x`	Position at which the density should be evaluated.
`minDist`	Minimum distance between nucleosomes.
`varDist`	Variance of nucleosome distances.
`meanDist`	Mean distance of nucleosomes.
`minLength`	Minimum fragment length.
`maxLength`	Maximum fragment length.
`meanLength`	Mean fragment length.
`bind`	Position of binding site within fragment.

length	Length of region.
weight	Weight of feature.
stable	Density function for stable nucleosome.
dist	Density function of distances between nucleosomes.
shift	Distance between alternative position for stable nucleosome.
ratio	Ratio of probability mass associated with central and alternative positions for stable nucleosome.
stability	Stability of stable nucleosome.
fragLength	Length of DNA fragment. If x is not in [0, 1] this is used to normalize x.

Value

Density evaluated at the given position.

Author(s)

Peter Humburg

See Also

feat2dens

---

extractQuality *Obtain read qualities from a Fastq file or ShortReadQ object*

Description

Converts the read qualities encoded in fastq formatted files into error probabilities.

Usage

```
extractQuality(reads, minLength = 25, dir,
 type = c("Illumina", "Sanger", "Solexa"))
```

Arguments

reads	Either the name of a fastq file or a ShortReadQ object (see Details).
minLength	Minimum read length required.
dir	Directory of fastq file.
type	Character string indicating the format the qualities are encoded in (see Details).

Details

If reads and dir are character strings it is assumed that 'dir/reads' is the name of a fastq file. Otherwise reads should be a ShortReadQ object in which case dir is ignored.

Currently three different encodings of read qualities are supported. The encoding has to be selected via the type argument. The supported formats are

Illumina The format currently used by Illumina (version 1.3). This is a phred score between 0 and 40 encoded as ASCII characters 64 to 104. [default]

Sanger The Sanger format uses a phred quality score between 0 and 93 encoded as ASCII characters 33 to 126.

Solexa The old Solexa format previously used by Solexa/Illumina uses a quality score between -5 and 40 encoded as ASCII characters 59 to 104.

Value

A list with a vector of error probabilities for each read in reads that is at least minLength nucleotides long.

Author(s)

Peter Humburg

See Also

decodeQuality, readQualitySample

Examples

```
## Not run:
 ## load reads from a fastq file with Sanger encoding
 qualities <- extractQuality("test.fastq", dir=".", type="Sanger")

 ## extract error probabilities for first 25bp of each read
 qualities25 <- sapply(qualities, "[", 1:25)

 ## plot average quality for each position
 plot(rowMeans(qualities25), type='b', xlab="Read position",
  ylab="Error probability")

## End(Not run)
```

feat2dens *Convert a list of features into a feature density*

Description

Given a list of features (as produced by makeFeatures) computes the feature density for each and combines them into a chromosome wide density.

Usage

```
feat2dens(features, length, featureBgr = TRUE, ...)
```

Arguments

features	A list of features.
length	Total length of feature density vector (i.e. chromosome length). If this is missing the length is inferred from the feature parameters.
featureBgr	Logical indicating whether feature specific background should be added to the density. If this is TRUE the resulting density for each feature is a mixture of the feature density and a fuzzy, i.e. uniform, feature density. The weights of the components are determined by the feature weight.
...	Further arguments to featureDensity.

Value

A vector with the feature density for each position along the chromosome.

Author(s)

Peter Humburg

See Also

The majority of the work is done by calls to featureDensity and joinRegion.

Examples

```
set.seed(1)
## generate a (relatively short) sequence of nucleosome features
features <- placeFeatures(start=200, length=1e5)

## calculate density
featureDens <- feat2dens(features, length=1e5)
```

`featureDensity` *Computing density for a given feature*

Description

This set of functions is used to generate the density of individual features of different types. `featureDensity` is an S3 generic, functions may be defined for different feature classes.

Usage

```
featureDensity(x, ...)
## S3 method for class 'StableFeature'
featureDensity(x, stable=stableDens, background=FALSE, ...)
## S3 method for class 'StablePhasedFeature'
featureDensity(x, stable=stableDens, dist=distDens, background=FALSE, ...)
## S3 method for class 'ReversePhasedFeature'
featureDensity(x, stable=stableDens, dist=distDens, background=FALSE, ...)
## S3 method for class 'NFRFeature'
featureDensity(x, background=FALSE, ...)
## S3 method for class 'FuzzyFeature'
featureDensity(x, ...)
```

Arguments

`x`	The feature for which the density should be computed.
`stable`	Function that should be used to compute the density of a stable feature.
`dist`	Function that should be used to compute the distribution of distances between adjacent features.
`background`	Logical indicating whether uniform background should be added to the feature.
`...`	Arguments to future functions.

Details

These functions are used internally by `feat2dens`. There should be no need to call them directly but it is important to supply suitable `featureDensity` functions for new feature types.

Value

A two column matrix. The first column contains the density, the second the weight at each position.

Author(s)

Peter Humburg

See Also

`feat2dens`, `makeFeatures`

Examples

```
## Create a single StableFeature
feature <- stableFeature(start = 200, weight = 0.8, shift = 10,
 stability = 1, ratio = 1)

## Convert the feature into a density (without background)
featDens <- featureDensity(feature)

## If we want featureDensity to automatically add uniform background
## we have to ensure that the feature has a 'meanDist' parameter
## (this is usually added by 'reconcileFeatures').
feature$meanDist <- 200
featDens2 <- featureDensity(feature, background = TRUE)
```

`FeatureGenerators` *Generating Features*

Description

These functions are used to generate the parameters for different nucleosome positioning related features.

Usage

```
stableFeature(start, minDist = 175, weight = seq(0.1, 1, by = 0.1),
 shift = c(0, 5, 10), ratio = seq(0, 4, by = 0.25),
 stability = seq(0.1, 5, by = 0.1), weightProb, shiftProb,
 ratioProb, stabilityProb, ...)
phasedFeature(minDist = 175, length = seq(1000, 10000, by = minDist),
 meanDist = minDist:300, lengthProb, meanDistProb, start, ...)
fuzzyFeature(start, length = seq(1000, 10000, by = 1000),
 meanDist = 175:400, lengthProb, meanDistProb, ...)
nfrFeature(start, length = seq(50, 500, by = 10),
 weight = seq(0.1, 1, by = 0.1), lengthProb, weightProb, ...)
```

Arguments

`start`	Start location of feature on chromosome.
`minDist`	Minimum distance between nucleosomes.
`length`	A numeric vector giving possible values for the length of the feature.
`meanDist`	A numeric vector giving possible values for the mean distance between nucleosomes.
`weight`	A numeric vector giving possible values for the weight of the feature.
`shift`	A numeric vector giving possible values for the distance between favoured positions of stable nucleosomes.

ratio	A numeric vector giving possible values for the ratio between probabilities for alternative and central position of stable nucleosomes.
stability	A numeric vector giving possible values for the stability of stable nucleosomes.
lengthProb	Length distribution of feature. This corresponds to the state duration distribution of the underlying generating model.
meanDistProb	Distribution of mean distances between nucleosomes.
weightProb	Distribution of feature weights.
shiftProb	Distribution of distances between favoured positions of stable nucleosome.
ratioProb	Ratio distribution.
stabilityProb	Stability distribution.
...	Further arguments (currently ignored).

Value

A list of parameters for the corresponding feature type. These parameters are later used to compute nucleosome densities.

Author(s)

Peter Humburg

See Also

simChIP

Examples

```
feature <- stableFeature(200)
```

---

internal *Internal and undocumented functions*

Description

These functions are only used internally or are lacking documentation.

Author(s)

Peter Humburg

joinRegion *Combining two feature densities*

Description

Function to take two vectors of feature densities and combine them into a single vector, using overlap between the two densities and smoothing the transition.

Usage

```
joinRegion(left, right, overlap, overlapWeights)
```

Arguments

`left`	First density vector.
`right`	Second density vector.
`overlap`	Overlap between the two features.
`overlapWeights`	Weights for overlapping region.

Value

Returns the combined density vector.

Note

This function is used as part of `feat2dens` and there should be no need to call it directly although it may be useful for possible extensions.

Author(s)

Peter Humburg

makeFeatures *Generating a list of genomic features*

Description

This function generates a list of genomic features for a single chromosome based on a Markov model.

Usage

```
makeFeatures(generator = defaultGenerator(),
 transition = defaultTransition(), init = defaultInit(),
 start = 1000, length, control = list(),
 globals = list(minDist = 175), lastFeat = defaultLastFeat(),
 experimentType = "NucleosomePosition",
 truncate = FALSE, maxTries = 10, force=FALSE)
```

Arguments

generator	A named list providing functions to generate the parameters associated with each type of feature. The name of each element in the list is the name of the state the function should be associated with.
transition	Named list of transition probabilities. Each element is a (named) numeric vector giving the transition probabilities for the state indicated by the element's name, i.e., each element of the list is a row of the transition probability matrix but zero probabilities can be omitted.
init	Named numeric vector of initial state probabilities. The names have to correspond to state names of the model. Zero probabilities may be omitted.
start	Numeric value indicating the position at which the first feature should be placed.
length	Maximum length of DNA that should be covered with features.
control	Named list with additional arguments to generator functions (one list per generator). Again the names should be the same as the state names.
globals	List of global arguments to be passed to all generator functions.
lastFeat	Named logical vector indicating for each feature type whether it can be the last feature.
experimentType	Type of experiment the simulated features belong to. This is used as the class of the return value.
truncate	Logical value indicating whether the final feature should be truncated to ensure that total length does not exceed `length` (if `FALSE`, a feature that would be truncated is removed instead).
maxTries	Maximum number of attempts made to generate a valid sequence of features. If no valid sequence is generated during the first `maxTries` attempts the function will fail either silently (returning an empty sequence) or raise an error, depending on the value of `force`.
force	Logical indicating whether the function should be forced to return a feature sequence, even if no valid sequence was found. If this is `TRUE` an empty sequence will be returned in that case otherwise an error is raised.

Details

This function will generate features from any first order Markov model specified by `init`, `transition` and `generator`. If `force` is `FALSE` the returned feature sequence is guaranteed to contain at least one feature and end in a state that is indicated as possible end state in `lastFeat`. Note that the states for which `lastFeat` is `TRUE` are not end states in the sense that the chain is terminated once the state

is entered or that the chain remains in the state once it is first entered. Instead this is a mechanism to ensure that some states cannot be the last in the sequence.

Due to the constrains on the total length of DNA covered by features as well as the possible constraint on the final feature of the sequence it is possible to specify models that cannot produce a legal sequence. In other cases it may be possible but unlikely to produce a feature sequence that satisfies both constraints. A serious attempt is made to satisfy both requirement, generating a new feature sequence or truncating an existing one if necessary. To ensure that this terminates eventually the number of attempts to generate a valid sequence are limited to `maxTries`.

In some cases it may be desirable to carry out some post-processing of the feature sequence to ensure that parameters of neighbouring features are compatible in some sense. For the default nucleosome positioning simulation the function `reconcileFeatures` provides this functionality and `placeFeatures` is an interface to `makeFeatures` that utilises `reconcileFeatures`.

Value

A list of features (with class determined by `experimentType`). Each feature is represented by a list of parameters and has a class with the same name as the state that generated the feature. In addition all features are of class `SimulatedFeature`.

Author(s)

Peter Humburg

See Also

Functions to generate default values for some of the arguments: `defaultGenerator`, `defaultInit`, `defaultTransition`, `defaultLastFeat`.

Use `feat2dens` to convert a feature sequence into feature densities.

`placeFeatures` is an interface to `makeFeature` for nucleosome positioning.

Examples

```
set.seed(1)
## generate a (relatively short) sequence of nucleosome features
features <- makeFeatures(length=1e6)

## check the total length of the features
sum(sapply(features, "[[", "length")) ## 995020
```

`placeFeatures` *Generating and reconciling a feature sequence*

Description

This function provides an interface to `makeFeatures` and `reconcileFeatures` that combines both steps of the feature generation process.

Usage

```
placeFeatures(..., maxTail = 0.01,
 compoundFeatures=list("StablePhasedFeature"))
```

Arguments

`...`	Arguments to `makeFeatures`.
`maxTail`	Maximum portion of total length of chromosome that may be left free of features (see Details).
`compoundFeatures`	List of feature classes that are produced by combining two features. This may happen during the call to `reconcileFeatures` and requires special handling when extending the feature list.

Details

This function (as well as `makeFeatures` which it calls) tries to fill as much of the genomic region with features as possible, i.e. an attempt is made to produce a feature sequence that covers `length` base pairs. In most cases the sequence will be slightly shorter. The `maxTail` argument determines how long a region without any features at the end of the genomic region is acceptable (as fraction of the total length). Note however that even `maxTail = 0` does not guarantee a feature sequence of exactly the requested length.

Value

A list of simulated features. The class of the return value as well as the features generated depend on the arguments passed to `makeFeatures`.

Note

Using the `reconcileFeatures` mechanism it is possible to introduce dependence between neighbouring features that is not easily expressed in terms of a simple Markov model. In some cases the same effect can be achieved by introducing additional states into the model but it may be more convenient to simply post-process the feature sequence.

Author(s)

Peter Humburg

See Also

`makeFeatures`, `reconcileFeatures`

Examples

```
set.seed(1)
## generate a (relatively short) sequence of nucleosome features
features <- placeFeatures(length=1e6, maxTail = 0)
```

```
## check the total length of the features
sum(sapply(features, "[[", "length")) ## 990509
```

---

pos2fastq *Convert read positions to fastq records*

Description

Convert read positions for a single chromosome (both strands) into read sequences + qualities and write them to file

Usage

```
pos2fastq(readPos, names, quality, sequence, qualityFun, errorFun,
 readLen = 36, file,
 qualityType = c("Illumina", "Sanger", "Solexa"), ...)
```

Arguments

readPos	A list of two numeric vectors (one per strand)
names	List of names to use for reads in fastq file. Has to be of same shape as name.
quality	Passed on as argument to qualityFun.
sequence	Reference sequence (a DNAString object).
qualityFun	Function to generate quality scores.
errorFun	Function to introduce sequencing errors.
readLen	Read length to generate.
file	Output file (either file name or connection).
qualityType	Encoding to use for read quality scores.
...	Further arguments (see Details).

Details

Arguments passed as part of ... will be passed on to qualityFun, except an argument called prob which is passed on to errorFun instead if present.

Value

Invisibly returns the number of records that were written.

Author(s)

Peter Humburg

See Also

See readError for a possible choice of errorFun and readQualitySample for a simple qualityFun.

Examples

```
set.seed(1)

## a function to generate random read qualities (in Sanger format)
randomQuality <- function(read, ...){
 paste(sample(unlist(strsplit(rawToChar(as.raw(33:126)),"")),
  length(read), replace = TRUE), collapse="")
}

## generate a reference sequence
chromosome <- DNAString(paste(sample(c("A", "C", "G", "T"),
 1e5, replace = TRUE), collapse = ""))

## and a few read positions
reads <- list(sample(100:9900, 5), sample(100:9900, 5))
names <- list(paste("read", 1:5, sep="_"), paste("read", 6:10, sep="_"))

## convert to fastq format
pos2fastq(reads, names, sequence=chromosome, qualityFun=randomQuality,
 errorFun=readError, file="")
```

`readError` *Introduce errors into read sequence based on quality scores*

Description

Given a read sequence and quality this function introduces errors by first choosing positions that should be modified based on the quality score and then exchanges nucleotides based on the probabilities given in `prob`.

Usage

```
readError(read, qual, alphabet = c("A", "C", "G", "T"),
 prob = defaultErrorProb(), ...)
```

Arguments

`read`	A character string representing a read sequence.
`qual`	Numeric vector of read error probabilities.
`alphabet`	Alphabet used for read sequence.
`prob`	Nucleotide exchange probabilities.
`...`	Further arguments (currently ignored).

Details

If the read sequence contains letters that are not part of `alphabet` they are replaced by the first entry of `alphabet` before positions of sequencing errors are determined. The alphabet used has to match the names used in `prob`.

Value

The modified read sequence.

Author(s)

Peter Humburg

See Also

defaultErrorProb, readSequence

Examples

```
set.seed(42)

## generate sequence read and quality
quality <- paste(sample(unlist(strsplit(rawToChar(as.raw(33:126)),"")),
 36, replace = TRUE), collapse="")
errorProb <- decodeQuality(quality, type = "Sanger")
read <- paste(sample(c("A", "C", "G", "T"), 36, replace = TRUE),
 collapse = "")

## use readError to introduce sequencing errors
read2 <- readError(read, errorProb)

all.equal(read, read2)  ## "1 string mismatch"
```

---

readQualitySample *Sample read qualities from a list*

Description

Given a read sequence and a list of read quality scores this function returns a (possibly truncated) quality score of the same length as the read.

Usage

```
readQualitySample(read, qualities, checkLength = TRUE, ...)
```

Arguments

read	A sequence read.
qualities	List of sequence read quality scores.
checkLength	Flag indicating whether the length of quality scores should be checked to ensure that they are at least as long as the read. If qualities contains entries shorter than read this has to be TRUE, but see below.
...	Further arguments, currently not used.

Details

Using checkLength = TRUE leads to a substantial decrease in performance and is impractical for a large simulation. To avoid this slow down it is recommended to remove short sequences from qualities beforehand so that checkLength = FALSE can be used.

Value

An read quality score string of the same length as read.

Author(s)

Peter Humburg

---

readSequence *Convert read position into read sequence*

Description

Given a read position, a reference sequence, a strand and a read length this function returns the read sequence.

Usage

```
readSequence(readPos, sequence, strand, readLen = 36)
```

Arguments

readPos	Numeric value indicating the start position on the chromosome.
sequence	Chromosome sequence (a DNAString)
strand	Strand indicator (+1 / -1)
readLen	Length of read.

Value

Read sequence.

Author(s)

Peter Humburg

See Also

readError, writeReads

`reconcileFeatures` *Post-processing of simulated features*

Description

The `reconcileFeatures` functions provide a facility to post-process a list of features representing a simulated experiment. `reconcileFeatures` is an S3 generic, new functions can be added for additional types of experiment. The current default is to call `reconcileFeatures.SimulatedExperiment` which, if called without further arguments, will simply return the feature list unchanged.

Usage

```
reconcileFeatures(features, ...)
## Default S3 method:
reconcileFeatures(features, ...)
## S3 method for class 'SimulatedExperiment'
reconcileFeatures(features, defaultValues=list(), ...)
## S3 method for class 'NucleosomePosition'
reconcileFeatures(features, defaultMeanDist = 200, ...)
```

Arguments

`features`	List of simulated features.
`defaultValues`	Named list of default parameter values. The method for class `SimulatedExperiment` ensures that all features have at least the parameters listed in `defaultValues`, adding them where necessary.
`defaultMeanDist`	Default value for the average distance between nucleosomes for nucleosome positioning experiments.
`...`	Further arguments to future functions.

Value

A list of features of the same class as `features`.

Author(s)

Peter Humburg

See Also

`makeFeatures`, `placeFeatures`

Examples

```
set.seed(1)
## generate a (relatively short) sequence of nucleosome features
features <- makeFeatures(length=1e6, )

## check the total length of the features
sum(sapply(features, "[[", "length")) ## 995020

## reconcile features to ensure smooth transitions
## For experiments of class NucleosomePosition this
## also combines some features and introduces
## some overlap between them.
features <- reconcileFeatures(features)

## check the total length of the features again
sum(sapply(features, "[[", "length")) ## 984170
```

`sampleReads` *Sampling sequence read positions from a read density.*

Description

Given a read density this function returns the starting positions of sequence reads.

Usage

```
sampleReads(readDens, nreads = 6e+06, strandProb = c(0.5, 0.5))
```

Arguments

`readDens`	A two column matrix of read densities (as produced by `bindDens2readDens`).
`nreads`	Number of read positions to generate.
`strandProb`	A numeric vector with two elements giving weights for forward and reverse strand.

Details

The expected number of reads for each strand is `strandProb * nreads`.

Value

A list with components `fwd` and `rev` giving the read positions on the forward and reverse strand respectively.

Author(s)

Peter Humburg

See Also

bindDens2readDens

Examples

```
set.seed(1)
## generate a (relatively short) sequence of nucleosome features
features <- placeFeatures(start=200, length=1e5)

## calculate feature density
featureDens <- feat2dens(features, length=1e5)

## convert to read density
readDens <- bindDens2readDens(featureDens, fragDens, meanLength=160)

## sample reads
## of course you would usually want a much larger number
readPos <- sampleReads(readDens, nreads=1000)
```

---

simChIP *Simulate ChIP-seq experiments*

Description

This function acts as driver for the simulation. It takes all required arguments and passes them on to the functions for the different stages of the simulation. The current defaults will simulate a nucleosome positioning experiment.

Usage

```
simChIP(nreads, genome, file, functions = defaultFunctions(),
 control = defaultControl(), verbose = TRUE, load = FALSE)
```

Arguments

nreads	Number of reads to generate.
genome	An object of class 'DNAStringSet' or the name of a fasta file containing the genome sequence.
file	Base of output file names (see Details).
functions	Named list of functions to use for various stages of the simulation, expected names are: 'features', 'bindDens', 'readDens', 'sampleReads', 'readNames', 'readSequence'
control	Named list of arguments to be passed to simulation functions (one list per function).
verbose	Logical indicating whether progress messages should be printed.
load	Logical indicating whether an attempt should be made to load intermediate results from a previous run.

Details

The simulation consists of six of stages:

1. generate feature sequence (for each chromosome): chromosome length -> feature sequence (list)
2. compute binding site density: feature sequence -> binding site density (vector)
3. compute read density: binding site density -> read density (two column matrix, one column for each strand)
4. sample read start sites: read density -> read positions (list)
5. create read names: number of reads -> unique names
6. obtain read sequence and quality: read positions, genome sequence, [qualities] -> output file

After each of the first three stages the results of the stage are written to a file and can be reused later. File names are created by appending '`_features.rdata`', '`_bindDensity.rdata`' and '`_readDensity.rdata`' to `file` respectively. Previous results will be loaded for reuse if `load` is `TRUE` and files with matching names are found. This is useful to sample repeatedly from the same read density or to recover partial results from an interrupted run.

The creation of files can be prevented by setting `file` = "". In this case all results will be returned in a list at the end. Note that this will require more memory since all intermediate results have to be held until the end.

The behaviour of the simulation is mainly controlled through the `functions` and `control` arguments. They are expected to be lists of the same length with matching names. The names indicate the stage of the simulation for which the function should be used; elements of `control` will be used as arguments for the corresponding functions.

Value

A list. The components are typically either lists (with one component per chromosome) or file names but note that this may depend on the return value of functions listed in `functions`. The components of the returned list are:

`features`	Either a list of generated features or the name of a file containing that list;
`bindDensity`	Either a list with binding site densities or the name of a file containing that list;
`readDensity`	Either a list of read densities or the name of a file containing that list;
`readPosition`	Either a list of read start sites or the name of a file containing that list;
`readSequence`	The return value of the function listed as '`readSequence`'. The default for this the name of the fastq file containing the read sequences;
`readNames`	Either a list of read names or the name of a file containing that list.

Author(s)

Peter Humburg

See Also

`defaultFunctions`, `defaultControl`

Examples

```
## Not run:
## To run the default nucleosome positioning simulation
## we can simply run something like the line below.
## This will result in 10 million reads sampled from the genome.
## Of course the file names have to be changed as appropriate.
simChIP(1e7, genome = "reference.fasta", file = "output/sim_10M")

## End(Not run)
```

`simpleNames` *Generate unique read names*

Description

Generates a set of unique (and very simple) read names.

Usage

```
simpleNames(n, nameBase = "read")
```

Arguments

`n`	Number of names to generate.
`nameBase`	Base name to use.

Value

A character vector with entries of the form 'nameBase_i' where i runs from 1 to `n`.

Author(s)

Peter Humburg

Examples

```
simpleNames(5)
```

writeFASTQ *Write read sequences and qualities to a FASTQ formatted file*

Description

This is intended to produce the final output of the simulation by providing a fastq file that may then be used in an analysis pipeline.

Usage

```
writeFASTQ(read, quality, name, file, append = FALSE)
```

Arguments

read	List of read sequences.
quality	List of read quality scores.
name	Read names.
file	File name. If this is "" results will be printed to the standard output connection.
append	Logical indicating the reads should be appended to an existing file.

Details

The first three arguments should have the same length but will be recycled if necessary.

Value

Called for its side effect.

Author(s)

Peter Humburg

See Also

readSequence, readQualitySample, writeReads

Examples

```
set.seed(1)

## generate sequence read and quality
quality <- paste(sample(unlist(strsplit(rawToChar(as.raw(33:126)),"")),
 36, replace = TRUE), collapse="")
read <- paste(sample(c("A", "C", "G", "T"), 36, replace = TRUE), collapse = "")

## write a fastq record
writeFASTQ(read, quality, "read_1", file="")
```

writeReads *Create fastq file from read positions*

Description

This is an interface to pos2fastq that writes all reads for a given genome to a single file.

Usage

```
writeReads(readPos, readNames, sequence, quality, file, ...)
```

Arguments

readPos	List of read positions with each component holding the read positions for one chromosome, which are themselves two component lists that provide forward and reverse strand positions.
readNames	List of the same shape as readPos providing read names.
sequence	Genome reference sequence (a DNAStringSet).
quality	Read quality scores (see Details).
file	Output file.
...	Further arguments to pos2fastq.

Details

If quality looks like it might refer to a fastq file an attempt is made to create a ShortReadQ object. The read qualities of any ShortReadQ object passed as quality (directly or indirectly as file name) are extracted and passed on to pos2fastq as quality argument. Otherwise it is passed on unchanged.

Value

The name of the output file.

Author(s)

Peter Humburg

See Also

pos2fastq

Index

∗ **datagen**
bindDens2readDens, 3
defaultGenerator, 8
distDens, 9
feat2dens, 12
featureDensity, 13
FeatureGenerators, 14
makeFeatures, 16
placeFeatures, 18
readError, 21
readQualitySample, 22
reconcileFeatures, 24
sampleReads, 25
simChIP, 26
∗ **internal**
distDens, 9
internal, 15
joinRegion, 16
∗ **package**
ChIPsim-package, 2
∗ **utilities**
decodeQuality, 4
defaultControl, 5
defaultErrorProb, 6
defaultFunctions, 7
defaultGenerator, 8
extractQuality, 10
pos2fastq, 20
readQualitySample, 22
readSequence, 23
simpleNames, 28
writeFASTQ, 29
writeReads, 30

bindDens2readDens, 3, *7*, *25, 26*
bindLocDens *(*distDens*)*, 9

ChIPsim *(*ChIPsim-package*)*, 2
ChIPsim-package, 2

decodeQuality, 4, *11*
defaultControl, 5, *27*
defaultErrorProb, 6, *22*
defaultFunctions, *6*, 7, *27*
defaultGenerator, 8, *18*
defaultInit, *18*
defaultInit *(*defaultGenerator*)*, 8
defaultLastFeat, *18*
defaultLastFeat *(*defaultGenerator*)*, 8
defaultTransition, *18*
defaultTransition *(*defaultGenerator*)*, 8
distDens, 9
DNAString, *20*, *23*
DNAStringSet, *30*

encodeQuality *(*decodeQuality*)*, 4
extractQuality, *4, 5*, 10

feat2dens, *4*, *7*, *10*, 12, *13*, *16*, *18*
featureDensity, *12*, 13
FeatureGenerators, 14
fragDens, *5*
fragDens *(*distDens*)*, 9
fuzzyFeature *(*FeatureGenerators*)*, 14

indNuc *(*distDens*)*, 9
internal, 15

joinRegion, *12*, 16

makeFeatures, *8*, *12, 13*, 16, *18, 19*, *24*

nfrFeature *(*FeatureGenerators*)*, 14
noNuc *(*distDens*)*, 9

phasedFeature *(*FeatureGenerators*)*, 14
phaseNuc *(*distDens*)*, 9
placeFeatures, *7*, *18*, 18, *24*
pos2fastq, 20, *30*

readError, *6*, *20*, 21, *23*

readQualitySample, *6*, *11*, *20*, 22, *29*
readSequence, *22*, 23, *29*
reconcileFeatures, *18, 19*, 24

sampleFromFile *(*internal*)*, 15
sampleReads, *4*, *7*, 25
ShortRead, *3*
ShortReadQ, *10, 11*, *30*
simChIP, *3*, *6, 7*, *15*, 26
simpleNames, *7*, 28
solexaNames *(*internal*)*, 15
stableDens *(*distDens*)*, 9
stableFeature *(*FeatureGenerators*)*, 14

writeFASTQ, 29
writeIllumina *(*internal*)*, 15
writeReads, *7*, *23*, *29*, 30