Package ‘spiky’

February 21, 2024

Type  Package
Title  Spike-in calibration for cell-free MeDIP
Description
spiky implements methods and model generation for cfMeDIP (cell-free methylated DNA immunoprecipitation) with spike-in controls. CfMeDIP is an enrichment protocol which avoids destructive conversion of scarce template, making it ideal as a "liquid biopsy," but creating certain challenges in comparing results across specimens, subjects, and experiments. The use of synthetic spike-in standard oligos allows diagnostics performed with cfMeDIP to quantitatively compare samples across subjects, experiments, and time points in both relative and absolute terms.

Version  1.8.0
Date  2023-04-19
biocViews  DifferentialMethylation, DNAMethylation, Normalization, Preprocessing, QualityControl, Sequencing
URL  https://github.com/trichelab/spiky
BugReports  https://github.com/trichelab/spiky/issues
License  GPL-2
Depends  Rsamtools, GenomicRanges, R (>= 3.6.0)
Imports  stats, scales, bamlls, methods, tools, IRanges, Biostrings, GenomicAlignments, BlandAltmanLeh, GenomeInfoDb, BSgenome, S4Vectors, graphics, ggplot2, utils
Suggests  covr, testthat, markdown, markdown, knitr, devtools, BSgenome.Mmusculus.UCSC.mm10.masked, BSgenome.Hsapiens.UCSC.hg38.masked, BiocManager
RoxygenNote  7.2.1
Roxygen  list(markdown = TRUE)
VignetteBuilder  knitr
Encoding  UTF-8
LazyData  true
git_url  https://git.bioconductor.org/packages/spiky
git_branch  RELEASE_3_18
git_last_commit 9732d52
git_last_commit_date 2023-10-24
Repository Bioconductor 3.18
Date/Publication 2024-02-20
Author Samantha Wilson [aut],
Lauren Harmon [aut],
Tim Triche [aut, cre]
Maintainer Tim Triche <trichelab@gmail.com>

R topics documented:

add_frag_info .................................................. 3
bam_to_bins .................................................... 4
bin_pmol ......................................................... 5
convertPairedGRtoGR ........................................... 5
covg_to_df ....................................................... 6
dedup ............................................................ 7
find_spike_contigs ............................................. 7
genbank_mito .................................................... 8
generate_spike_fasta ........................................... 9
genomic_res ..................................................... 10
get_base_name ................................................ 11
get_binned_coverage .......................................... 11
get_merged_gr ................................................ 12
get_spiked_coverage .......................................... 13
get_spike_depth ............................................... 14
kmax ............................................................. 15
kmers ............................................................ 16
methylation_specificity ....................................... 17
model_bam_standards ......................................... 17
model_glm_pmol ................................................. 18
parse_spike_UMI ............................................... 19
phage ............................................................ 20
predict_pmol ................................................... 20
process_spikes ................................................ 21
read_bedpe ...................................................... 22
rename_spikes .................................................. 23
rename_spike_seqlevels ...................................... 24
scan_genomic_bedpe ........................................... 25
scan_genomic_contigs ........................................ 26
scan_methylation_specificity ................................ 27
scan_spiked_bam ............................................... 28
scan_spike_bedpe ............................................. 30
scan_spike_contigs ........................................... 30
scan_spike_counts ............................................ 31
seqinfo_from_header .......................................... 32
add_frag_info

decode fragment identifiers for spike-in standards

Description

given a vector of fragment identifiers like 160_2_35 or 80b_1C_35G-2, encoded typically as length-InBp_numberOfCpGs_GCpercent, and optionally a database of spike-in sequences corresponding to those fragments, add those columns to the source data (along with, if present in the database, other metadata such as standard concentrations, GC fraction, etc.) and return an updated DataFrame.

Usage

add_frag_info(x, fraggrp = "fraggrp", spike = NULL)

Arguments

x data.frame with a column of spike information (see above)
fraggrp column name for the spike contig information (fraggrp)
spike optional database of spike-in properties (none)

Value

the data.frame x, augmented with metadata columns

Examples

data(spike_cram_counts)
data(spike, package="spiky")
spike <- subset(spike, methylated == 1)
add_frag_info(spike_cram_counts, spike=spike)
bam_to_bins  
create a tiled representation of a genome from the BAM/CRAM file

Description
This function replaces a bedtools call: bedtools intersect -wao -a fragments.bed -b hg38_300bp_windows.bed > data.bed

Usage
bam_to_bins(x, width = 300, param = NULL, which = IRangesList(), ...)

Arguments
- **x**: a BAM or CRAM filename (or a BamFile object)
- **width**: the width of the bins to tile (default is 300)
- **param**: optional ScanBamParam (whence we attempt to extract which)
- **which**: an optional GRanges restricting the bins to certain locations
- **...**: additional arguments to pass on to seqinfo_from_header

Details
The idea is to skip the BED creation step for most runs, and just do it once. In order to count reads in bins, we need bins. In order to have bins, we need to know how long the chromosomes are. In order to have a BAM or CRAM file, we need to have those same lengths. This function takes advantage of all of the above to create binned ranges. Note that a very recent branch of Rsamtools is required for CRAM file bins.

Value
a GRangesList with y-base-pair-wide bins tiled across it

See Also
seqinfo_from_header

Examples
library(Rsamtools)
fl <- system.file("extdata", "ex1.bam", package="Rsamtools", mustWork=TRUE)
bam_to_bins(fl)
bin_pmol

Binned estimation of picomoles of DNA present in cfMeDIP assays

Description
Given the results of model_glm_pmol and predict_pmol, adjust the predictions to reflect picomoles of captured DNA overlapping a given bin in the genome.

Usage
bin_pmol(x)

Arguments
x results from predict_pmol (a data.frame or GRanges)

Value
the same object, but with a column `adjusted_pred_con`

See Also
model_glm_pmol
predict_pmol

Examples
data(spike, package="spiky")
data(spike_res, package="spiky")
data(genomic_res,package="spiky")
fit <- model_glm_pmol(covg_to_df(spike_res, spike=spike),spike=spike)
pred <- predict_pmol(fit, genomic_res, ret="df")
bin_pmol(pred)

convertPairedGRtoGR

Convert Pairs to GRanges

Description
Convert Pairs to GRanges

Usage
convertPairedGRtoGR(pairs)
covg_to_df

Arguments

pairs the Pairs object

Value

a GRanges

Description

reshape scan_spiked_bam results into data.frames for model_glm_pmol

Usage

covg_to_df(spike_gr, spike, meth = TRUE, ID = NULL)

Arguments

spike_gr GRanges of spike contigs (e.g. output object from scan_spiked_bam, scan_spike_contigs, or scan_spike_bedpe)
spike spike database (as from data(spike, package="spiky"))
meth only keep methylated spike reads? (TRUE; if FALSE, sum both)
ID an identifier for this sample, if running several (autogenerate)

Value

a data.frame with columns 'frag.grp', 'id', and 'read.count'

See Also

scan_spiked_bam

Examples

data(spike, package="spiky")
data(spike_res, package="spiky")
subsetted <- covg_to_df(spike_res, spike=spike, meth=TRUE)
summed <- covg_to_df(spike_res, spike=spike, meth=FALSE)
round((summed$read.count - subsetted$read.count) / summed$read.count, 3)
### dedup

*Spike-in counts for two samples, as a wide data.frame*

**Description**

A data.frame with spike-in results from control samples in the manuscript. This maps 1:1 onto `spike_read_counts` using `reshape2::melt`.

**Usage**

```r
data(dedup)
```

**Format**

A data.frame object with

- `frag_grp`: the encoded spike contig name: basepairs_CpGs_GCpercent
- `read_count_6547`: read coverage for this spike in sample 6547
- `read_count_6548`: read coverage for this spike in sample 6548

**Source**

This data was created using `inst/script/loadDedup.R`

---

### find_spike_contigs

*Find spike-in seqlevels in an object* \(x\), where `!is.null(seqinfo(x))`

**Description**

Find the spike-like contigs in a BAM with both natural and spiked contigs. This started out as glue in some other functions and got refactored out.

**Usage**

```r
find_spike_contigs(x, spike)
```

**Arguments**

- `x`: something with seqlevels
- `spike`: a DataFrame with spike-in information

**Details**

The indices have an attribute "mappings", which is a character vector such that `attr(find_spike_contigs(x), "mappings")` is the rowname in `spike` that corresponds to the original contig name.
Value

indices of which contigs in seqlevels(x) are spike-in contigs

See Also

generate

rename_spike_seqlevels

Examples

sb <- system.file("extdata", "example.spike.bam", package="spiky",
mustWork=TRUE)
si <- seqinfo_from_header(sb)
data(spike, package="spiky")
find_spike_contigs(si, spike=spike)

genbank_mito

| genbank_mito | various mitochondrial genomes sometimes used as endogenous spike-ins |

Description

A DataFrame with species, genome, accession, and sequence for GenBank mitochondrial genome depositions. No concentration provided; add if needed.

Usage

data(genbank_mito)

Format

A DataFrame object with

- **species** the species whence the record came, as a character string
- **genome** the genome assembly whence the mtDNA, as a character string
- **accession** the genbank accession, as a character string
- **sequence** genome sequence, as a DNAStringSet

Source

generate_spike_fasta

for CRAM files, a FASTA reference is required to decode; this builds that

Description

A FASTA reference is not always needed, so long as .crai indices are available for all contigs in the CRAM. See spike_counts for a fast and convenient alternative that extracts spike coverage from index stats. However, spike_counts has its own issues, and it's better to use fragments.

Usage

generate_spike_fasta(bam, spike, assembly = NULL, fa = "spike_contigs.fa")

Arguments

- **bam**: a BAM or CRAM file, hopefully with an index
- **spike**: the spike contig database (mandatory as of 0.9.99)
- **assembly**: optional BSgenome or seqinfo with reference contigs (NULL)
- **fa**: the filename for the resulting FASTA ("spikes.fa")

Details

If the contigs in a CRAM have even slightly different names from those in the reference, decoding will fail. In some cases there are multiple names for a given contig (which raises the question of whether to condense them), and thus the same reference sequence decodes multiple contig names.

This function generates an appropriate spike reference for a BAM or CRAM, using BAM/CRAM headers to figure out which references are used for which.

At the moment, CRAM support in Rsamtools only exists in the GitHub branch:

BiocManager::install("Bioconductor/Rsamtools@cram")

Using other versions of Rsamtools will yield an error on CRAM files.

Note that for merged genomic + spike reference BAMS/CRAMS, this function will only attempt to generate a FASTA for the spike contigs, not reference. If your reference contigs are screwed up, talk to your sequencing people, and keep better track of the FASTA reference against which you compress!

Value

invisibly, a DNAStringSet as exported to 'fa'

See Also

rename_contigs
Examples

```r
library(GenomicRanges)
data(spike, package="spiky")
sb <- system.file("extdata", "example.spike.bam", package="spiky",
                 mustWork=TRUE)
outFasta <- paste(system.file("extdata", package="spiky", mustWork=TRUE),"/spike_contigs.fa",sep="")
show(generate_spike_fasta(sb, spike=spike, fa=outFasta))
```

---

**genomic_res**

A Granges object with genomic coverage from chr21q22, binned every 300bp for the genomic contigs then averaged across the bin. (In other words, the default output of scan_genomic_contigs or scan_genomic_bedpe, restricted to a small enough set of genomic regions to be practical for examples.) This represents what most users will want to generate from their own genomic BAMs or BEDPEs, and is used repeatedly in downstream examples throughout the package.

---

**Description**

A Granges object with genomic coverage from chr21q22, binned every 300bp for the genomic contigs then averaged across the bin. (In other words, the default output of scan_genomic_contigs or scan_genomic_bedpe, restricted to a small enough set of genomic regions to be practical for examples.) This represents what most users will want to generate from their own genomic BAMs or BEDPEs, and is used repeatedly in downstream examples throughout the package.

---

**Usage**

```r
data(genomic_res)
```

---

**Format**

A GRanges of coverage results with one metadata column, coverage

---

**Source**

Generated using scan_genomic_bedpe or scan_genomic_contigs on an example bedpe or bam containing chr21q22 contigs.
get_base_name

Description

A common task between generate_spike_fasta, rename_spikes, and rename_spike_seqlevels is to determine what the largest common subset of characters between existing contig names and stored standardized contigs might be. This function eases that task.

Usage

get_base_name(contig_names, sep = "_")

Arguments

contig_names the names of contigs
sep separator character in contig names ("_")

Value

a vector of elements 1:3 from each contig name

Examples

sb <- system.file("extdata", "example.spike.bam", package="spiky",
    mustWork=TRUE)
bh <- scanBamHeader(BamFile(sb))
orig_contigs <- names(bh$targets)
get_base_name(orig_contigs)

get_binned_coverage

Description

refactored out of scan_spiked_bam

Usage

get_binned_coverage(bins, covg)

Arguments

bins the GRanges with bins
covg the coverage result (an RleList)
get_merged_gr

**Value**

a GRanges of summarized coverage

**See Also**

get_spiked_coverage

scan_spiked_bam

**Examples**

```r
sb <- system.file("extdata", "example.spike.bam", package="spiky", mustWork=TRUE)
data(spike, package="spiky")
si <- seqinfo_from_header(sb)
genome(si) <- "spike"
mgr <- get_merged_gr(si, spike=spike)
fl <- scanBamFlag(isDuplicate=FALSE, isPaired=TRUE, isProperPair=TRUE)
bp <- ScanBamParam(flag=fl)
bamMapqFilter(bp) <- 20
covg <- get_spiked_coverage(sb, bp=bp, gr=mgr)
get_binned_coverage(bins=GRanges(), covg=covg)
```

---

**Description**

refactored from scan_spiked_bam to clarify information flow

**Usage**

```r
get_merged_gr(si, spike, standard = TRUE)
```

**Arguments**

- `si` seqinfo, usually from a BAM/CRAM file with spike contigs
- `spike` database of spike-in standard sequence features (spike)
- `standard` trim to standard chromosomes? (TRUE)

**Details**

By default, `get_merged_gr` will return a GRanges with "standardized" genomic and spike contig names (i.e. genomic chr1-22, X, Y, M, and the canonical spike names in `data(spike, package="spiky")`).

The constraint to "standard" chromosomes on genomic contigs can be removed by setting `standard` to FALSE in the function arguments.
get_spiked_coverage

Value

GRanges with two genomes: the organism assembly and "spike"

Examples

sb <- system.file("extdata", "example.spike.bam", package="spiky", mustWork=TRUE)
si <- seqinfo_from_header(sb)
genome(si) <- "spike" # no genomic contigs
data(spike, package="spiky")
get_merged_gr(si, spike=spike) # note canonicalized spikes

get_spiked_coverage

tabulate coverage across assembly and spike contig subset in natural order

Description

FIXME: this is wicked slow, ask Herve if a faster version exists

Usage

get_spiked_coverage(bf, bp, gr)

Arguments

bf the BamFile object
bp the ScanBamParam object
gr the GRanges with sorted seqlevels

Details

Refactored from scan_spiked_bam, this is a very simple wrapper

Value

a list of Rles

See Also

scan_spiked_bam
coverage
get_spike_depth

get the (max, median, or mean) coverage for spike-in contigs from a BAM/CRAM

Usage

```r
get_spike_depth(covg, spike_gr = NULL, spike = NULL, how = c("max", "mean"))
```

Arguments

- `covg`: the coverage RleList
- `spike_gr`: the spike-in GRanges (default: figure out from seqinfo)
- `spike`: information about the spikes (default: load spike)
- `how`: how to summarize the per-spike coverage (max)

Value

A GRanges with summarized coverage and features for each

Examples

```r
sb <- system.file("extdata", "example.spike.bam", package="spiky", mustWork=TRUE)
si <- seqinfo_from_header(sb)
genome(si) <- "spike"
data(spike, package="spiky")
mgr <- get_merged_gr(si, spike=spike) # note canonicalized spikes

fl <- scanBamFlag(isDuplicate=FALSE, isPaired=TRUE, isProperPair=TRUE)
bp <- ScanBamParam(flag=fl)
bamMapqFilter(bp) <- 20
get_spiked_coverage(sb, bp=bp, gr=mgr)
```
bamMapqFilter(bp) <- 20

covg <- get_spiked_coverage(sb, bp=bp, gr=mgr)
get_spike_depth(covg, spike_gr=mgr, spike=spike)

---

**kmax**

**simple contig kmer comparisons**

---

**Description**

simple contig kmer comparisons

**Usage**

kmax(km, normalize = TRUE)

**Arguments**

km  
kmer summary

normalize  
normalize (divide by row sums)? (TRUE)

**Value**

the most common kmers for each contig, across all contigs

**Examples**

data(genbank_mito, package="spiky")
mtk6 <- kmers(genbank_mito, k=6)
rownames(mtk6) <- paste0(rownames(mtk6), "_MT")
kmax(mtk6)

data(phage, package="spiky")
phk6 <- kmers(phage, k=6)
kmax(phk6, normalize=FALSE)

stopifnot(identical(colnames(phk6), colnames(mtk6)))
k6 <- rbind(mtk6, phk6)
kmax(k6)
kmers

Description

oligonucleotideFrequency, but less letters and more convenient.

Usage

kmers(x, k = 6)

Arguments

x  BSgenome, DFrame with sequence column, or DNAStringSet
k  the length of the kmers (default is 6)

Details

The companion kmax function finds the maximum frequency kmer for each contig and plots all of them together for comparison purposes.

Value

a matrix of contigs (rows) by kmer frequencies (columns)

See Also

kmax

Examples

data(genbank_mito, package="spiky")
mtk6 <- kmers(genbank_mito, k=6)
kmax(mtk6)

data(phage, package="spiky")
phk6 <- kmers(phage, k=6)
kmax(phk6)
**methyltion_specificity**

compute methylation specificity for spike-in standards

---

**Description**

In a cfMeDIP experiment, the yield of methylated fragments should be >95% (ideally 98-99%) due to the nature of the assay.

**Usage**

```r
methyltion_specificity(spike_gr, spike)
```

**Arguments**

- `spike_gr`: GRanges of spike contigs (e.g. output object from scan_spiked_bam, scan_spike_contigs, or scan_spike_bedpe)
- `spike`: spike contig database, if needed (e.g. data(spike))

**Value**

list with median and mean coverage across spike contigs

**Examples**

```r
data(genomic_res)
data(spike_res)
data(spike, package="spiky")
methyltion_specificity(spike_res, spike=spike)
```

---

**model_bam_standards**

Build a Bayesian additive model from spike-ins to correct bias in *-seq

---

**Description**

Build a Bayesian additive model from spike-ins to correct bias in *-seq

**Usage**

```r
model_bam_standards(x, conc = NULL, fm = NULL, ...)
```
model_glm_pmol

Build a generalized linear model from spike-ins to correct bias in cfMeDIP

Arguments

  - `x`: data with assorted feature information (GCfrac, CpGs, etc)
  - `conc`: concentration for each spike (must be provided!)
  - `fm`: model formula (conc ~ read_count + fraglen + GCfrac + CpGs_3)
  - `...`: other arguments to pass to `bamlss`

Value

the model fit for the data

Examples

```r
library(bamlss)
data(spike_cram_counts, package="spiky")
data(spike, package="spiky")
scc <- add_frag_info(spike_cram_counts, spike=spike)
scc$conc <- scc$conc * 0.9 # adjust for dilution
scc$CpGs_3 <- scc$CpGs ^ (1/3)
fit0 <- model_bam_standards(scc,
  fm=conc ~ read_count + fraglen)
fit1 <- model_bam_standards(scc,
  fm=conc ~ read_count + fraglen + GCfrac + CpGs_3)
DIC(fit0, fit1)
```

Description

formerly `2020_model_glm_fmol`. Note that everything in x can be had from a BAM/CRAM with spike contigs named as frag_grp(len_CpGs_GC) in the index and in fact that is what `scan_spiked_bam` now does.

Usage

```r
model_glm_pmol(x, spike, conc = NULL, ...)
```

Arguments

  - `x`: data w/frag_grp, id, and read_count; or `scan_spiked_bam` result
  - `spike`: spike database, e.g. `data(spike, package='spiky')`
  - `conc`: concentration for each spike (will be referenced if NULL)
  - `...`: other arguments to pass to glm (e.g. family)
**Value**

the model fit for the data

**Examples**

```r
data(spike, package="spiky")
data(spike_read_counts, package="spiky")
fit1 <- model_glm_pmol(spike_read_counts, spike=spike)
data(spike_res) # scan_spiked_bam result
fit2 <- model_glm_pmol(spike_res, spike=spike)
```

---

**Description**

parse out the forward and reverse UMIs and contig for a BED/BAM

**Usage**

```r
parse_spike_UMI(UMI, pos = NULL, seqs = NULL)
```

**Arguments**

- **UMI**
  - a vector of UMIs
- **pos**
  - optional vector of positions (else all are set to 1)
- **seqs**
  - optional vector of read sequences (else widths default to 96)

**Value**

a GRanges
**predict_pmol**

**Description**

predict picomoles of DNA from a fit and read counts (coverage)

**Usage**

```r
predict_pmol(
  fit,
  genomic_gr,
  bsgenome = NULL,
  ret = c("gr", "df"),
  slide = FALSE
)
```

---

**phage**

lambda and phiX phage sequences, sometimes used as spike-ins

**Description**

A DataFrame with sequence, methylated, CpGs, GCfrac, and OECpG for phages

**Usage**

```r
data(phage)
```

**Format**

A DataFrame object with

- **sequence** genome sequence, as a DNAStringSet
- **methylated** whether CpGs are methylated, as an integer
- **CpGs** the number of CpGs in the phage genome, as an integer
- **GCfrac** the GC fraction of the phage genome, as a numeric
- **OECpG** the observed / expected CpG fraction, as a numeric

**Source**


---

**predict_pmol**

predict picomoles of DNA from a fit and read counts (coverage)

**Description**

FIXME: this could be made MUCH faster by precomputing CpG/GC stats per bin

**Usage**

```r
predict_pmol(
  fit,
  genomic_gr,
  bsgenome = NULL,
  ret = c("gr", "df"),
  slide = FALSE
)
```
process_spikes

QC, QA, and processing for a new spike database

Description

Sequence feature verification: never trust anyone, least of all yourself.

Usage

process_spikes(fasta, methylated = 0, ...)

Arguments

fasta          fasta file (or GRanges or DataFrame) w/spike sequences
methylated     whether CpGs in each are methylated (0 or 1, default 0)
...             additional arguments, e.g. kernels (currently unused)
Details

GCfrac is the GC content of spikes as a proportion instead of a percent. OECpG is (observed/expected) CpGs (expectation is 25% of GC dinucleotides).

Value

a DataFrame suitable for downstream processing

See Also

kmers

Examples

data(spike)
spikes <- system.file("extdata", "spikes.fa", package="spiky", mustWork=TRUE)
spikemeth <- spike$methylated
process_spikes(spikes, spikemeth)

data(phage)
phages <- system.file("extdata", "phages.fa", package="spiky", mustWork=TRUE)
identical(process_spikes(phage), phage)
identical(phage, process_spikes(phage))

data(genbank_mito)
(mt <- process_spikes(genbank_mito)) # see also genbank_mito.R
gb_mito <- system.file("extdata", "genbank_mito.R", package="spiky")

---

read_bedpe

*read a BEDPE file into Pairs of GRanges (as if a GAlignmentPairs or similar)*

Description

read a BEDPE file into Pairs of GRanges (as if a GAlignmentPairs or similar)

Usage

read_bedpe(
  x,
  ..., stranded = FALSE,
  fraglen = TRUE,
  optional = FALSE,
  keep = FALSE
)
Arguments

- `x`: a Tabixed BEDPE file, or a TabixFile of one
- `...`: additional arguments to pass to scanTabix internally
- `stranded`: Is the data stranded? (FALSE)
- `fraglen`: compute the fragment length? (TRUE)
- `optional`: scan the optional columns (name, score, strand1)? (FALSE)
- `keep`: keep additional columns? (FALSE)

Details

BEDPE import in R is a shambles. This is a bandaid on a GSW.

See the [BEDPE format definition](https://bedtools.readthedocs.io/en/latest/content/general-usage.html#bedpe-format) for full details.

In short, for a pair of ranges 1 and 2, we have fields chrom1, start1, end1, chrom2, start2, end2, and (optionally) name, score, strand1, strand2, plus any other user defined fields that may be included (these are not yet supported by read_bedpe). For example, two valid BEDPE lines are:

```
chr1 100 200 chr5 5000 5100 bedpe_example1 30
chr9 900 5000 chr9 3000 3800 bedpe_example2 99 + -
```

Value

- a Pairs of GRanges, perhaps with $score or $fraglen

See Also

bedpe_covg

Examples

```r
## Not run:
bedpe <- "GSM5067076_2020_A64_bedpe.bed.gz"
WT1_hg38 <- GRanges("chr1", IRanges(32387775, 32435564), "-")
read_bedpe(bedpe, param=WT1_hg38)
## End(Not run)
```

rename_spikes

for BAM/CRAM files with renamed contigs, we need to rename spike rows

Description

This function does that.
rename_spike_seqlevels

Usage
rename_spikes(x, spike)

Arguments
- x: a BAM/CRAM file, hopefully with an index
- spike: a DataFrame where spike$sequence is a DNAStringSet

Value
a DataFrame with renamed contigs (rows)

See Also
generate_spike_fasta

comments
rename_spike_seqlevels
for spike-in contigs in GRanges, match to standardized spike seqlevels

Description
This function is essentially the opposite of rename_spikes, except that it works well on GRanges/GAlignments from or for merged genome+spike BAMs. If spike contigs are found, it will assign genome='spike' to those, while changing the seqlevels to standardized names that match rownames(spike).

Usage
rename_spike_seqlevels(x, spike = NULL)

Arguments
- x: something with seqlevels (GRanges, GAlignments, Seqinfo...)
- spike: a DataFrame where spike$sequence is a DNAStringSet (or NULL)

Value
x, but with standardized spike seqlevels and genomes

See Also
rename_spikes
Description

Scan genomic BEDPE

Usage

```r
scan_genomic_bedpe(
    bedpe,
    bin = TRUE,
    binwidth = 300L,
    bins = NULL,
    standard = TRUE,
    genome = "hg38"
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bedpe</td>
<td>the BEDPE file path, or output from read_bedpe()</td>
</tr>
<tr>
<td>bin</td>
<td>Bin reads? (TRUE)</td>
</tr>
<tr>
<td>binwidth</td>
<td>width of the bins for chromosomal tiling (300)</td>
</tr>
<tr>
<td>bins</td>
<td>a pre-tiled GRanges for binning coverage (NULL)</td>
</tr>
<tr>
<td>standard</td>
<td>restrict non-spike contigs to &quot;standard&quot; chromosomes? (TRUE)</td>
</tr>
<tr>
<td>genome</td>
<td>Name of genome (default hg38)</td>
</tr>
</tbody>
</table>

Value

a GRanges with coverage

Examples

```r
fl <- system.file("extdata", "example_chr21_bedpe.bed.gz", package="spiky", mustWork=TRUE)
scan_genomic_bedpe(fl) # will warn user about spike contigs
```
scan_genomic_contigs

scan_genomic_contigs in a BAM/CRAM file

Description

The default workflow for spiky is roughly as follows:

Usage

scan_genomic_contigs(
  bam,
  spike,
  param = NULL,
  bin = TRUE,
  binwidth = 300L,
  bins = NULL,
  standard = TRUE,
  genome = "hg38",
  ...
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bam</td>
<td>the BAM or CRAM filename, or a vector of them</td>
</tr>
<tr>
<td>spike</td>
<td>the spike-in reference database (e.g. data(spike))</td>
</tr>
<tr>
<td>param</td>
<td>a ScanBamParam object specifying which reads to count (NULL)</td>
</tr>
<tr>
<td>bin</td>
<td>Bin reads? (TRUE)</td>
</tr>
<tr>
<td>binwidth</td>
<td>width of the bins for chromosomal tiling (300)</td>
</tr>
<tr>
<td>bins</td>
<td>a pre-tiled GRanges for binning coverage (NULL)</td>
</tr>
<tr>
<td>standard</td>
<td>restrict non-spike contigs to &quot;standard&quot; chromosomes? (TRUE)</td>
</tr>
<tr>
<td>genome</td>
<td>Name of genome (default hg38)</td>
</tr>
<tr>
<td>...</td>
<td>additional arguments to pass to scanBamFlag()</td>
</tr>
</tbody>
</table>

Details

1. Identify and quantify the spike-in contigs in an experiment.
2. Fit a model for sequence-based abundance artifacts using the spike-ins.
3. Quantify raw fragment abundance on genomic contigs, and adjust per step 2.

scan_genomic_contigs addresses the first half of step 3. The assumption is that anything which isn’t a spike contig is a genomic contig. This isn’t necessarily true, so the user can also supply a ScanBamParam object for the param argument and restrict scanning to whatever contigs they wish, which also allows for non-default MAPQ, pairing, and quality filters.

If multiple BAM or CRAM filenames are provided, all indices will be checked before attempting to run through any of the files.
scan_methylation_specificity

**Value**

a CompressedGRangesList with bin- and spike-level coverage

**See Also**

Rsamtools::ScanBamParam

**Examples**

```r
library(Rsamtools)
data(spike, package="spiky")

fl <- system.file("extdata", "ex1.bam", package="Rsamtools",
    mustWork=TRUE)
scan_genomic_contigs(fl, spike=spike,standard=FALSE) # will warn user about spike contigs

sb <- system.file("extdata", "example_chr21.bam", package="spiky",
    mustWork=TRUE)
scan_genomic_contigs(sb, spike=spike) # will warn user about genomic contigs
```

---

**scan_methylation_specificity**

*tabulate methylation specificity for multiple spike-in BAM/CRAM files*

**Description**

Methylation specificity is here defined as methylated_spike_covg/spike_covg

**Usage**

```r
scan_methylation_specificity(files, spike, sep = "_")
```

**Arguments**

- **files**: a vector of BAM/CRAM file names
- **spike**: a spike-in database
- **sep**: the separator for spike-in contig names ("_")

**Value**

a matrix with columns "mean" and "median"
Examples

```r
data(spike)
library(GenomicRanges)
sb <- system.file("extdata", "example.spike.bam", package="spiky",
                    mustWork=TRUE)
scan_methylation_specificity(sb, spike=spike)
```

---

**scan_spiked_bam**  
pretty much what it says: scan standard chroms + spike contigs from a BAM

---

Description

Note: behind the scenes, this is being refactored into `scan_spike_contigs` and `scan_genomic_contigs`. Once that is done, perhaps before release, the default workflow will switch to

Usage

```r
scan_spiked_bam(
  bam,
  spike,
  mapq = 20,
  binwidth = 300L,
  bins = NULL,
  how = c("max", "mean"),
  dupe = FALSE,
  paired = TRUE,
  standard = TRUE,
  ...
)
```

Arguments

- **bam** the BAM file
- **spike** the spike-in reference database (e.g. `data(spike)`)
- **mapq** minimum mapq value to count a pair (20)
- **binwidth** width of the bins for chromosomal tiling (300)
- **bins** a pre-tiled GRanges for binning coverage (NULL)
- **how** how to record spike read coverage (max or mean)? (max)
- **dupe** unique (FALSE), duplicate (TRUE), or all (NA) reads? (FALSE)
- **paired** restrict coverage to that from properly paired reads? (TRUE)
- **standard** restrict non-spike contigs to "standard" chromosomes? (TRUE)
- ... additional arguments to pass to `scanBamFlag()`
Details

1. scan spike contigs and count fragments per contig or per bin.
2. fit the appropriate model for adjusting genomic contigs based on spikes.
3. scan and adjust binned fragment tallies along genomic contigs per above.

This approach decouples binning schemes from model generation (using spikes) and model-based adjustment (using genomic fragment counts), decreasing code complexity while increasing the opportunities for caching & parallelization.

For a more realistic example (not run), one might do something like:

```r
data(spike, package="spiky"); bam <- "2021_ctl.hg38_withSpikes.bam"; ssb_res <- scan_spiked_bam(bam, mapq=20, spike=spike);
```

An extract from the resulting `ssb_res` object is available via

```r
data(ssb_res, package="spiky");
```

The full `ssb_res` is a GRangesList object with 300bp-binned coverage on the standard (chr1-22, chrX, chrY, chrM) chromosomes (as determined by the GenomeInfoDb::standardChromosomes() function against the assembly defined in the BAM or CRAM file, by default; if desired, a user can scan all genomic contigs by setting standard=FALSE when calling the function). By default, the mean base-level coverage of genomic bins is reported, and the maximum spike-level coverage is reported, though this can also be adjusted as needed. The results then inform the reliability of measurements from replicate samples in multiple labs, as well as the adjusted quantitative coverage in each bin once the absolute quantity of captured cell-free methylated DNA has been fit by model_glm_pmol and predict_pmol. In some sense, this function converts BAMs/CRAMs into usable data structures for high-throughput standardized cfMeDIP experiments.

The data extract used in other examples is the same as the full version, with the sole difference being that genomic bins are limited to chr22.

Value

- a CompressedGRangesList with bin- and spike-level coverage

See Also

- GenomeInfoDb::keepStandardChromosomes
- Rsamtools::ScanBamParam

Examples

```r
library(GenomicRanges)
data(spike, package="spiky")
sb <- system.file("extdata", "example.spike.bam", package="spiky", mustWork=TRUE)
res <- scan_spiked_bam(sb, spike=spike, bins=GRanges())
summary(res$spikes$coverage)
```
scan_spike_bedpe  

Scan spikes BEDPE

Description
Scan spikes BEDPE

Usage
scan_spike_bedpe(bedpe, spike, how = "max")

Arguments
- **bedpe**: the BEDPE file path, or output from read_bedpe()
- **spike**: information about the spikes (default: load spike)
- **how**: how to summarize the per-spike coverage (max)

Value
a GRanges with coverage

Examples
```r
data(spike, package="spiky")
fl <- system.file("extdata", "example_spike_bedpe.bed.gz", package="spiky", mustWork=TRUE)
scan_spike_bedpe(fl,spike=spike) # will warn user about spike contigs
```

scan_spike_contigs  

pretty much what it says: scan spike contigs from a BAM or CRAM file

Description
default workflow is

Usage
scan_spike_contigs(bam, spike, how = "max", param = NULL, mc.cores = 16, ...)

Arguments
- **bam**: the BAM or CRAM filename, or a vector of such filenames
- **spike**: the spike-in reference database (e.g. data(spike))
- **how**: how to summarize the per-spike coverage (max)
- **param**: a ScanBamParam object, or NULL (will default to MAPQ=20 etc)
- **mc.cores**: Number of cores to run on (default 16)
- **...**: additional arguments to pass to scanBamFlag()
scan_spike_counts

Details

1. scan spike contigs and count fragments per contig or per bin.
2. fit the appropriate model for adjusting genomic contigs based on spikes.
3. scan and adjust binned fragment tallies along genomic contigs per above.

scan_spike_contigs implements step 1.

If multiple BAM or CRAM filenames are provided, all indices will be checked before attempting to run through any of the files.

Value

a CompressedGRangesList with bin- and spike-level coverage

See Also

Rsamtools::ScanBamParam

Examples

library(GenomicRanges)
data(spike, package="spiky")
sb <- system.file("extdata", "example.spike.bam", package="spiky",
    mustWork=TRUE) # switch to a CRAM
res <- scan_spike_contigs(sb, spike=spike) # use default ScanBamParam
summary(res)

scan_spike_counts

run spike_counts on BAM/CRAM files and shape the results for model_glm_pmol

Description

Typically one will want to fit a correction model to multiple samples. This function eases this task by merging the output of spike_counts into a data.frame that model_glm_pmol can directly fit.

Usage

scan_spike_counts(files, spike, methylated = 1, sep = ".")

Arguments

files a vector of BAM/CRAM file names
spike a spike-in database
methylated a logical (0/1) to include only methylated fragments
sep the separator for spike-in contig names ("_")
seqinfo_from_header

create seqinfo (and thus a standard chromosome filter) from a BAM header

Description

create seqinfo (and thus a standard chromosome filter) from a BAM header

Usage

seqinfo_from_header(x, gen = NA, std = FALSE, ret = c("si", "gr"))

Arguments

x the BAM file or its header
gen genome of the BAM file, if known (NULL; autodetect)
std standard chromosomes only? (FALSE; will be empty if spikes)
ret return Seqinfo ("si", the default) or GRanges ("gr")? ("si")

Details

Setting std=TRUE on a spike-in BAM will produce an empty result.

Value

Seqinfo object or GRanges (or `as(seqinfo, "GRanges")`)

Examples

data(spike)
library(GenomicRanges)
sb <- system.file("extdata", "example.spike.bam", package="spiky",
mustWork=TRUE)
scan_spike_counts(sb, spike=spike)
fit <- model_glm_pmol(scan_spike_counts(sb, spike=spike),spike=spike)
Examples

```r
library(Rsamtools)
fl <- system.file("extdata", "ex1.bam", package="Rsamtools", mustWork=TRUE)

hdr <- scanBamHeader(BamFile(fl))
si <- seqinfo_from_header(hdr)
gr <- seqinfo_from_header(fl, ret="gr")
stoptifnot(identical(gr, as(si, "GRanges")))

std_si <- seqinfo_from_header(fl, std=TRUE)
seqlevels(std_si)

# for comparison with below
data(spike, package="spiky")
spike

sp <- system.file("extdata", "example.spike.bam", package="spiky")
sp_gr <- seqinfo_from_header(sp, ret="gr")
sp_gr
```

---

### Description

A DataFrame with sequence, concentration, and other properties of Sam’s synthetic cfMeDIP spike-in controls. The row names redundantly encode some of these properties, such as the number of CpGs in the spike-in sequence.

### Usage

```r
data(spike)
```

### Format

A DataFrame object with

- **sequence** contig sequence, as a DNAStringSet
- **methylated** are the CpGs in this spike-in methylated? 0 or 1
- **CpGs** number of CpG dinucleotides in the spike, from 1 to 16
- **fmol** femtomolar concentration of the spike-in for standard mix
- **molmass** molar mass of spike-in sequence

### Source

https://doi.org/10.1101/2021.02.12.430289
### spike_bland_altman_plot

**Bland-Altman plot for cfMeDIP spike standards**

**Description**

Bland-Altman plot for cfMeDIP spike standards

**Usage**

```r
spike_bland_altman_plot(fit)
```

**Arguments**

- `fit`  
  a model fit, from `predict_pmol` (?)

**Value**

a `ggplot2` object

**Examples**

```r
data(spike_res)
data(spike, package="spiky")
fit <- model_glm_pmol(covg_to_df(spike_res, spike=spike),spike=spike)
ba_plot <- spike_bland_altman_plot(fit)
```

### spike_counts

use the index of a spiked BAM/CRAM file for spike contig coverage

**Description**

It dawned on me one day that we don’t even have to bother reading the file if we have an index for a spiked BAM/CRAM result, since any fragments that map properly to the spike contigs are generated from synthetic templates. This function takes an index and a spike database (usually a DataFrame) as inputs and provides a rough coverage estimate over "rehabilitated" contig names (i.e., canonicalized contigs mapping to the database) as its output.
Usage

spike_counts(
  bam,
  spike,
  sep = "_",
  ref = "spike",
  verbose = FALSE,
  dump_idx = FALSE
)

Arguments

- **bam**: the BAM or CRAM file (MUST HAVE AN INDEX)
- **spike**: a data.frame, DataFrame, or similar with spikes
- **sep**: separator character in contig names ("_")
- **ref**: reference name for spike genome ("spike")
- **verbose**: be verbose? (FALSE)
- **dump_idx**: dump the renamed idxstats to aggregate? (FALSE)

Details

The argument spike has no default since we are attempting to refactor the spike-in databases into their own data packages and allow more general use.

Value

a GRanges of spike-in contig read counts

Examples

data(spike, package="spiky")
sb <- system.file("extdata", "example.spike.bam", package="spiky",
                 mustWork=TRUE)
spike_counts(sb, spike=spike)

Description

A data.frame with spike-in results from CRAM files (generated from scan_spike_counts(CRAMs, spike=spike))

Usage

data(spike_cram_counts)
**spike_read_counts**

**Format**

A data.frame object with

- **frag_grp** the encoded spike contig name: basepairs_CpGs_GCpercent
- **id** subject from whom cfMeDIP spike reads (column 3) were counted
- **read_count** read coverage for this spike in this subject (column 2)

**Source**

Generated from `scan_spike_counts(CRAMs, spike=spike)` using example CRAMs containing spike contigs

---

**spike_read_counts**  
spike-in counts, as a long data.frame

**Description**

A data.frame with spike-in results from control samples in the manuscript. This maps 1:1 onto dedup using reshape2::melt.

**Usage**

data(spike_read_counts)

**Format**

A data.frame object with

- **frag_grp** the encoded spike contig name: basepairs_CpGs_GCpercent
- **id** subject from whom cfMeDIP spike reads (column 3) were counted
- **read_count** read coverage for this spike in this subject (column 2)

**Source**

This data was created using inst/script/loadDedup.R
Description

A Granges object with spike-in sequence coverage, and summarized for each spike contig as (the default) max coverage. (In other words, the default output of scan_spike_contigs or scan_spike_bedpe) This represents what most users will want to generate from their own spike-in BAMs or BEDPEs, and is used repeatedly in downstream examples throughout the package.

Usage

data(spike_res)

Format

A GRanges of coverage results with one metadata column, coverage

Source

Generated using scan_spike_bedpe or scan_spike_contigs on an example bedpe or bam containing spike contigs.

---

spyky-methods

A handful of methods that I’ve always felt were missing

Description

Particularly, simple methods to plot coverage results.

Usage

```r
## S4 method for signature 'Rle,ANY'
plot(x, y, ...)

## S4 method for signature 'SimpleRleList,ANY'
plot(x, y, ...)
```
**Arguments**

- **x** an Rle or RleList, usually
- **y** not used an Rle or RleList, usually
- ... other params such as ylim passed to barplot

**Details**

selectMethod("plot", "Rle") and also selectMethod("plot", "RleList") too.

**Value**

invisibly, the plot details

---

### data(ssb_res)

**Format**

A CompressedGRangesList of coverage results, containing

- **genomic** a GRanges with one metadata column, coverage
- **spikes** a GRanges with one metadata column, coverage

**Source**

Generated using scan_spiked_bam on an example bam containing chr22 and spike contigs.
testGR

*a test GRanges with UMI’ed genomic sequences used as controls*

Description

Sources and overlap widths of various read sequences in a test CRAM.

Usage

data(testGR)

Format

A GRanges object with an mcols() DataFrame containing

- **UMI1** the unique molecular identifier on the forward read
- **UMI2** the unique molecular identifier on the reverse read
- **seq** the sequence of the fragment
- **name** the name of the fragment
- **score** whether the fragment passes filters (always 1)

Source

Generated using inst/script/loadTest.R

tile_bins

*Tile the assembly-based contigs of a merged assembly/spike GRanges.*

Description

refactored out of scan_spiked_bam for more explicit information flow

Usage

tile_bins(gr, binwidth = 300L)

Arguments

- **gr** the GRanges
- **binwidth** bin width to tile (default is 300)

Value

a GRanges of bins
Examples

```r
bam <- system.file("extdata", "ex1.bam", package="Rsamtools",
              mustWork=TRUE)
gr <- as(seqinfo_from_header(bam), "GRanges")
genome(gr) <- "notspike"
tile_bins(gr)
```
Index

* datasets
  dedup, 7
  genbank_mito, 8
  genomic_res, 10
  phage, 20
  spike, 33
  spike_cram_counts, 35
  spike_read_counts, 36
  spike_res, 37
  ssb_res, 38
  testGR, 39

  add_frag_info, 3
  bam_to_bins, 4
  bin_pmol, 5
  convertPairedGRtoGR, 5
  covg_to_df, 6
  dedup, 7
  find_spike_contigs, 7
  genbank_mito, 8
  generate_spike_fasta, 9
  genomic_res, 10
  get_base_name, 11
  get_binned_coverage, 11
  get_merged_gr, 12
  get_spike_depth, 14
  get_spiked_coverage, 13
  kmax, 15
  kmers, 16
  methylation_specificity, 17
  model_bam_standards, 17
  model_glm_pmol, 18
  parse_spike_UMI, 19

  phage, 20
  plot,Rle,ANY-method (spiky-methods), 37
  plot,SimpleRleList,ANY-method
    (spiky-methods), 37
  predict_pmol, 20
  process_spikes, 21
  read_bedpe, 22
  rename_spike_seqlevels, 24
  rename_spikes, 23
  scan_genomic_bedpe, 25
  scan_genomic_contigs, 26
  scan_methylation_specificity, 27
  scan_spike_bedpe, 30
  scan_spike_contigs, 30
  scan_spike_counts, 31
  scan_spiked_bam, 28
  seqinfo_from_header, 32
  spike, 33
  spike_bland_altman_plot, 34
  spike_counts, 34
  spike_cram_counts, 35
  spike_read_counts, 36
  spike_res, 37
  spiky-methods, 37
  ssb_res, 38

  testGR, 39
  tile_bins, 39