Package 'VariantTools'

November 12, 2026

Type Package

Title Tools for Exploratory Analysis of Variant Calls

Version 1.54.0

Author Michael Lawrence, Jeremiah Degenhardt, Robert Gentleman

Maintainer Michael Lawrence <lawremi@gmail.com>

Description Explore, diagnose, and compare variant calls using filters.

Depends R (>= 3.5.0), S4Vectors (>= 0.17.33), IRanges (>= 2.13.12), GenomicRanges (>= 1.31.8), VariantAnnotation (>= 1.11.16), methods

Imports Rsamtools (>= 1.31.2), BiocGenerics, Biostrings, parallel, GenomicFeatures (>= 1.31.3), Matrix, rtracklayer (>= 1.39.7), BiocParallel, GenomeInfoDb, BSgenome, Biobase

Suggests RUnit, LungCancerLines (>= 0.0.6), RBGL, graph, gmapR (>= 1.21.3), TxDb.Hsapiens.UCSC.hg19.knownGene, org.Hs.eg.db

License Artistic-2.0

LazyLoad yes

biocViews Genetics, GeneticVariability, Sequencing

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

git_branch RELEASE_3_23

git_last_commit 2b04d2b

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

annotateWithControlDepth 2
callGenotypes 3
callSampleSpecificVariants 5
callVariants 7

callWildtype 9
concordance 10
extractCoverageForPositions 11
FilterConstructors 12
matchVariants 13
pileupVariants 14
postFilterVariants 15
qaVariants 16
tallyVariants 17
variantGR2Vcf 19
vignette 21

Index **22**

`annotateWithControlDepth`

Annotate Case with Control Depth

Description

Matches the `case` variants to the (typically unfiltered) `control` variants and returns `case`, with the additional metadata columns `control.alt.depth` and `control.total.depth`, corresponding to `altDepth(control)` and `totalDepth(control)`, respectively.

Usage

```
annotateWithControlDepth(case, control, control.cov)
```

Arguments

`case`	The variants of interest, as a `VRanges`.
`control`	The control variants, typically unfiltered, as a `VRanges`.
`control.cov`	The control coverage, as an `RleList`.

Details

If a `case` variant is not found in `control`, a count of `0` is assigned to `control.alt.depth`, under the assumption that the `control` object is not filtered, i.e., it contains the raw tallies.

Value

`case`, plus two new metadata columns, `control.alt.depth` and `control.total.depth`

Author(s)

Michael Lawrence

See Also

callSampleSpecificVariants, which uses this function.

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
data(vignette)
case <- callVariants(tallies_H1993)
control <- tallies_H2073
control.cov <- coverage(bams$H2073)
annotateWithControlDepth(case, control, control.cov)
```

---

callGenotypes *Call Genotypes*

Description

Calls genotypes from a set of tallies (such as a VRanges or VCF file) and the coverage (currently as a BigWigFile). We call the genotype with the highest likelihood, where the likelihood is based on a binomial model of the variant frequency.

Usage

```
## S4 method for signature 'VRanges'
callGenotypes(variants, cov,
  param = CallGenotypesParam(variants),
  BPPARAM = defaultBPPARAM())
## S4 method for signature 'TabixFile'
callGenotypes(variants, cov,
  param = CallGenotypesParam(variants),
  BPPARAM = defaultBPPARAM())
CallGenotypesParam(genome,
                   gq.breaks = c(0, 5, 20, 60, Inf),
                   p.error = 0.05,
                   which = tileGenome(seqinfo(genome), ntile=ntile),
                   ntile = 100L)
```

Arguments

variants	Either VRanges as returned by tallyVariants, or a TabixFile object pointing to a VCF file. Typically, these tallies are *not* filtered by e.g. callVariants, because it would seem more appropriate to filter on the genotype quality.
cov	The coverage, as an RleList or a BigWigFile.
param	Parameters controlling the genotyping, constructed by CallGenotypesParam. The default value uses the genome from variants.
genome	An object with a getSeq method representing the genomic sequence used during tallying.

gq.breaks	A numeric vector representing an increasing sequence of genotype quality breaks to segment the wildtype runs.
p.error	The binomial probability for an error. This is used to calculate the expected frequency of hom-ref and hom-alt variants.
which	A GenomicRangesList indicating the genomic regions in which to compute the genotypes. The default is to partition the genome into ntile tiles.
ntile	When which is missing, this indicates the number of tiles to generate from the genome.
BPPARAM	A BiocParallelParam object communicating the parallelization strategy. One job is created per tile.

Details

In general, the behavior is very similar to that of the GATK UnifiedGenotyper (see references). For every position in the tallies, we compute a binomial likelihood for each of wildtype (0/0), het (0/1) and hom-alt (1/1), assuming the alt allele frequency to be p.error, 0.5 and 1 - p.error, respectively. The genotype with the maximum likelihood is chosen as the genotype, and the genotype quality is computed by taking the fraction of the maximum likelihood over the sum of the three likelihoods.

We assume that any position not present in the input tallies is wildtype (0/0) and compute the quality for every such position, using the provided coverage data. For scalability reasons, we segment runs of these positions according to user-specified breaks on the genotype quality. The segments become special records in the returned VRanges, where the range represents the segment, the ref is the first reference base, alt is <NON_REF> and the totalDepth is the mean of the coverage.

The genotype information is recorded as metadata columns named according to gVCF conventions:

GT The genotype call string: 0/0, 0/1, 1/1.

GQ The numeric genotype quality, phred scaled. For wildtype runs, this is minimum over the run.

PL A 3 column matrix with the likelihood for each genotype, phred scaled. We take the minimum over wildtype runs.

MIN_DP The minimum coverage over a wildtype run; NA for single positions.

Value

For callGenotypes, a VRanges annotated with the genotype call information, as described in the details.

Author(s)

Michael Lawrence

References

The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, Garimella K, Altshuler D, Gabriel S, Daly M, DePristo MA, 2010 GENOME RESEARCH 20:1297-303.

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
data(vignette)
tallies <- tallies_H1993

sampleNames(tallies) <- "H1993"
mcols(tallies) <- NULL

cov <- coverage(bams$H1993)

## simple usage
## (need gmapR to find the genome in the GMAP database, otherwise,
##  provide sequence directly as shown later)
if (requireNamespace("gmapR", quietly=TRUE)) {
    genotypes <- callGenotypes(tallies, cov,
                               BPPARAM=BiocParallel::SerialParam())
}

## customize
params <- CallGenotypesParam(genome_p53, p.error = 1/1000)
genotypes <- callGenotypes(tallies, cov, params)

## write to gVCF
writeVcf(genotypes, tempfile("genotypes", fileext="vcf"), index=TRUE)
```

`callSampleSpecificVariants` *Call Sample-Specific Variants*

Description

Calls sample-specific variants by comparing case and control variants from paired samples, starting from the BAM files or unfiltered tallies. For example, these variants would be considered somatic mutations in a tumor vs. normal comparison.

Usage

```
SampleSpecificVariantFilters(control, control.cov, calling.filters,
                             power = 0.8, p.value = 0.01)
## S4 method for signature 'BamFile,BamFile'
callSampleSpecificVariants(case, control,
  tally.param, calling.filters = VariantCallingFilters(), post.filters =
  FilterRules(), ...)
## S4 method for signature 'character,character'
callSampleSpecificVariants(case, control, ...)
## S4 method for signature 'VRanges,VRanges'
callSampleSpecificVariants(case,
  control, control.cov, ...)
```

```
## DEPRECATED
## S4 method for signature 'GenomicRanges,GenomicRanges'
callSampleSpecificVariants(case,
  control, control.cov,
  calling.filters = VariantCallingFilters(), post.filters =
  FilterRules(), ...)
```

Arguments

case	The BAM file for the case, or the called variants as output by callVariants.
control	The BAM file for the control, or the raw tallies as output by tallyVariants.
tally.param	Parameters controlling the variant tallying step, as typically constructed by TallyVariantsParam.
calling.filters	Filters to use for the initial, single-sample calling against reference, typically constructed by VariantCallingFilters.
post.filters	Filters that are applied after the initial calling step. These consider the set of variant calls as a whole and remove those with suspicious patterns. They are only applied to the case sample; only QA filters are applied to control.
...	For a BAM file, arguments to pass down to the GenomicRanges method. For the GenomicRanges method, arguments to pass down to SampleSpecificVariantFilters, except for control.cov, control.called, control.raw and lr.filter.
control.cov	The coverage for the control sample.
power	The power cutoff, beneath which a variant will not be called case-specific, due to lack of power in control.
p.value	The binomial p-value cutoff for determining whether the control frequency is sufficiently extreme (low) compared to the case frequency. A p-value below this cutoff means that the variant will be called case-specific.

Details

For each sample, the variants are tallied (when the input is BAM), QA filtered (case only), called and determined to be sample-specific. The callSampleSpecificVariants function is fairly high-level, but it still allows the user to override the parameters and filters for each stage of the process. See TallyVariantsParam, VariantQAFilters, VariantCallingFilters and SampleSpecificVariantFilters.

It is safest to pass a BAM file, so that the computations are consistent for both samples. The GenomicRanges method is provided mostly for optimization purposes, since tallying the variants over the entire genome is time-consuming. For small gene-size regions, performance should not be a concern.

This is the algorithm that determines whether a variant is specific to the case sample:

1. Filter out all case calls that were also called in control. The callSampleSpecificVariants function does **not** apply the QA filters when calling variants in control. This prevents a variant from being called specific to case merely due to questionable data in the control.
2. For the remaining case calls, calculate whether there was sufficient power in control under the likelihood ratio test, for a variant present at the p.lower frequency. If that is below the power cutoff, discard it.

3. For the remaining case calls, test whether the control frequency is sufficient extreme (low) compared to the case frequency, under the binomial model. The null hypothesis is that the frequencies are the same, so if the test p-value is above `p.value`, discard the variant. Otherwise, the variant is called case-specific.

Value

A `VRanges` with the case-specific variants (such as somatic mutations).

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
if (requireNamespace("gmapR", quietly=TRUE)) {
    tally.param <- TallyVariantsParam(gmapR::TP53Genome(),
                                      high_base_quality = 23L,
                                      which = gmapR::TP53Which())
    callSampleSpecificVariants(bams$H1993, bams$H2073, tally.param)
} else {
    data(vignette)
    calling.filters <- VariantCallingFilters(read.count = 3L)
    called.variants <- callVariants(tallies_H1993, calling.filters)
    callSampleSpecificVariants(called.variants, tallies_H2073,
                               coverage_H2073)
}
```

`callVariants` *Call Variants*

Description

Calls variants from either a BAM file or a `VRanges` object. The variants are called using a binomial likelihood ratio test. Those calls are then subjected to a post-filtering step.

Usage

```
## S4 method for signature 'BamFile'
callVariants(x, tally.param,
                                  calling.filters = VariantCallingFilters(...),
                                  post.filters = FilterRules(),
                                  ...)
## S4 method for signature 'character'
callVariants(x, ...)
## S4 method for signature 'VRanges'
callVariants(x,
     calling.filters = VariantCallingFilters(...),
```

```
    post.filters = FilterRules(),
    ...)
VariantCallingFilters(read.count = 2L, p.lower = 0.2, p.error = 1/1000)
```

Arguments

x	Either a path to an indexed bam, a BamFile object, or a VRanges as returned by tallyVariants.
tally.param	Parameters controlling the variant tallying step, as typically constructed by TallyVariantsParam.
calling.filters	Filters used in the calling step, typically constructed with VariantCallingFilters, see arguments listed below.
post.filters	Filters that are applied after the initial calling step. These consider the set of variant calls as a whole and remove those with suspicious patterns.
...	Arguments for VariantCallingFilters, listed below.
read.count	Require at least this many high quality reads with the alternate base. The default value is designed to catch sequencing errors where coverage is too low to rely on the LRT. Increasing this value has a significant negative impact on power.
p.lower	The lower bound on the binomial probability for a true variant.
p.error	The binomial probability for a sequencing error (default is reasonable for Illumina data with the default quality cutoff).

Details

There are two steps for calling variants: the actual statistical test that decides whether a variant exists in the data, and a post-filtering step. By default, the initial calling is based on a binomial likelihood ratio test (P(D|p=p.lower) / P(D|p=p.error) > 1). The test amounts to excluding putative variants with less than ~4% alt frequency. A variant is also required to be represented by at least 2 alt reads. The post-filtering stage considers the set of variant calls as a whole and removes variants with suspicious patterns. Currently, there is a single post-filter, disabled by default, that removes variants that are clumped together on the chromosome (see the max.nbor.count parameter).

Value

For callVariants, a VRanges of the called variants (the tallies that pass the calling filters). See the documentation of bam_tally for complete details.

For VariantCallingFilters, a FilterRules object with the filters for calling the variants.

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
if (requireNamespace("gmapR")) {
    tally.param <- TallyVariantsParam(gmapR::TP53Genome(),
```

```
                                    high_base_quality = 23L,
                                    which = gmapR::TP53Which())

    ## simple usage
    variants <- callVariants(bams$H1993, tally.param)
}
## customize
data(vignette)
calling.filters <- VariantCallingFilters(p.error = 1/1000)
callVariants(tallies_H1993, calling.filters)
```

`callWildtype` *Calling Wildtype*

Description

Decides whether a position is variant, wildtype, or uncallable, according to the estimated power of the given calling filters.

Usage

```
callWildtype(reads, variants, calling.filters, pos = NULL, ...)
minCallableCoverage(calling.filters, power = 0.80, max.coverage = 1000L)
```

Arguments

`reads`	The read alignments, i.e., a path to a BAM file, or the coverage, including a `BigWigFile` object.
`variants`	The called variants, a tally `GRanges`.
`calling.filters`	Filters used to call the variants.
`pos`	A `GRanges` indicating positions to query; output is in the same order. If this is `NULL`, the entire genome is considered. This is not called `which`, because we are indicating positions, not selecting from regions.
`power`	The chance of detecting a variant if one is there.
`max.coverage`	The max coverage to be considered for the minimum (should not need to be tweaked).
`...`	Arguments to pass down to `minCallableCoverage`.

Details

For each position (in the genome, or as specified by `pos`), the coverage is compared against the return value of `minCallableCoverage`. If the coverage is above the callable minimum, the position is called, either as a variant (if it is in `variants`) or wildtype. Otherwise, it is considered a no-call.

The `minCallableCoverage` function expects and only considers the filters returned by `VariantCallingFilters`.

Value

A logical vector (or logical `RleList` if `pos` is `NULL`), that is `TRUE` for wildtype, `FALSE` for variant, `NA` for no-call.

Author(s)

Michael Lawrence

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
bam <- bams$H1993

data(vignette)
called.variants <- callVariants(tallies_H1993)

pos <- c(called.variants, shift(called.variants, 3))
wildtype <- callWildtype(bam, called.variants, VariantCallingFilters(),
                         pos = pos, power = 0.85)
```

`concordance` *Variant Concordance*

Description

Functions for calculating concordance between variant sets and deciding whether two samples have identical genomes.

Usage

```
calculateVariantConcordance(gr1, gr2, which = NULL)
calculateConcordanceMatrix(variantFiles, ...)
callVariantConcordance(concordanceMatrix, threshold)
```

Arguments

`gr1`, `gr2`	The two tally `GRanges` to compare
`which`	A `GRanges` of positions to which the comparison is limited.
`variantFiles`	Character vector of paths to files representing tally `GRanges`. Currently supports serialized (rda) and VCF files. If the file extension is not "vcf", we assume rda. Will be improved in the future.
`concordanceMatrix`	A matrix of concordance fractions between sample pairs, as returend by `calculateConcordanceMatrix`.
`threshold`	The concordance fraction above which edges are generated between samples when forming the graph.
`...`	Arguments to pass to the loading function, e.g., `readVcf`.

Details

The `calculateVariantConcordance` calculates the fraction of concordant variants between two samples. Concordance is defined as having the same position and alt allele.

The `calculateConcordanceMatrix` function generates a numeric matrix with the concordance for each pair of samples. It accepts paths to serialized objects so that all variant calls are not loaded in memory at once. This probably should support VCF files, eventually.

The `callVariantConcordance` function generates a concordant/non-concordant/undecidable status for each sample (that are assumed to originate from the same individual), given the output of `calculateConcordanceMatrix`. The status is decided as follows. A graph is formed from the concordance matrix using `threshold` to generate the edges. If there are multiple cliques in the graph that each have more than one sample, every sample is declared undecidable. Otherwise, the samples in the clique with more than one sample, if any, are marked as concordant, and the others (in singleton cliques) are marked as discordant.

Value

Fraction of concordant variants for `calculateVariantConcordance`, a numeric matrix of concordances for `calculateConcordanceMatrix`, or a character vector of status codes, named by sample, for `callVariantConcordance`.

Author(s)

Cory Barr (code), Michael Lawrence (inferred documentation)

`extractCoverageForPositions` *Get Coverage at Positions*

Description

Gets values from an `RleList` corresponding to positions (width 1 ranges) in a `GRanges` (or `VRanges`). The result is a simple atomic vector.

Usage

```
extractCoverageForPositions(cov, pos)
```

Arguments

`cov`	An `RleList` like that returned by `coverage`.
`pos`	A `GRanges` consisting only of width-1 ranges.

Value

Atomic vector with one value from `cov` per position in `pos`.

Author(s)

Michael Lawrence

`FilterConstructors` *Variant Filter Constructors*

Description

These functions construct filters (implemented as functions) suitable for collection into `FilterRules` objects, which are then used to filter variant calls. See examples.

Usage

```
SetdiffVariantsFilter(other)
MinTotalDepthFilter(min.depth = 10L)
MaxControlFreqFilter(control, control.cov, max.control.freq = 0.03)
DepthFETFilter(control, control.cov, p.value.cutoff = 0.05)
```

Arguments

`other`	The set of variants (as a `VRanges`) to subtract from the set being filtered.
`min.depth`	The minimum depth for a variant to pass.
`control`	The control set of variants (as a `VRanges`) to use when filtering for case-specific variants.
`control.cov`	The coverage (as an `RleList`) for the sample corresponding to the calls in `control`.
`max.control.freq`	The maximum alt frequency allowed in the control for a variant to be considered case-specific.
`p.value.cutoff`	Passing variants must have a p-value below this value.

Value

In all cases, a closure that returns a logical vector indicating which elements of its argument should be retained.

Author(s)

Michael Lawrence

See Also

There are some convenience functions that construct `FilterRules` objects that contain one or more of these filters. Examples are `VariantQAFilters` and `VariantCallingFilters`.

Examples

```
## Find case-specific variants in a case/control study
bams <- LungCancerLines::LungCancerBamFiles()

data(vignette)
case <- callVariants(tallies_H1993)
control <- callVariants(tallies_H2073)

control.cov <- coverage(bams$H2073)

filters <-
  FilterRules(list(caseOnly = SetdiffVariantsFilter(control),
                   minTotalDepth = MinTotalDepthFilter(min.depth=10L),
                   maxControlFreq = MaxControlFreqFilter(control,
                       control.cov, max.control.freq=0.03),
                   depthFET = DepthFETFilter(control, control.cov,
                       p.value.cutoff=0.05)
                   ))

specific <- subsetByFilter(case, filters)
```

`matchVariants` *Match variants by position and allele*

Description

These are **deprecated** functions for operating on the old variant GRanges. New code should use `match` and `%in%`. This function behaves like `match`, where two elements match when they share the same position and "alt" allele.

Usage

```
matchVariants(x, table)
x %variant_in% table
```

Arguments

`x`	The variants (`GRanges`) to match into `table`; the alt allele must be in the "alt" metacolumn.
`table`	The variants (`GRanges`) to be matched into; the alt allele must be in the "alt" metacolumn.

Value

For `matchVariants`, an integer vector with the matching index in `table` for each variant in `x`, or `NA` if there is no match. For `%variant_in%`, a logical vector indicating whether there was such a match.

Author(s)

Michael Lawrence

`pileupVariants` *Nucleotide pileup from alignments*

Description

This is an alternative to `tallyVariants` for generating a `VRanges` from a set of alignments (BAM file) by counting the nucleotides at each position. This function uses the samtools-based `applyPileups` function, instead of `bam_tally`. Fewer dependencies, with fewer statistics (none beyond the fixed columns) available in the output.

Usage

```
pileupVariants(bams, genome, param = ApplyPileupsParam(), minAltDepth = 1L,
               baseOnly = TRUE, BPPARAM = defaultBPPARAM())
```

Arguments

`bams`	A vector/list of BAM files as interpreted by `PileupFiles`.
`genome`	An object that provides sequence information via `getSeq`.
`param`	A `ApplyPileupsParam` object that specifies the mode of iteration and various filters.
`minAltDepth`	Minimal alt depth to be included in the output. The default avoids outputting results for positions/alleles that show no differences.
`baseOnly`	Whether to drop records with "N" in either the ref or alt.
`BPPARAM`	Not yet supported.

Value

A VRanges object with read depth information for each position, allele, and sample.

Author(s)

Michael Lawrence

See Also

`tallyVariants` for more statistics.

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
if (requireNamespace("gmapR")) {
    param <- Rsamtools::ApplyPileupsParam(which=gmapR::TP53Which())
    pileup <- pileupVariants(bams, gmapR::TP53Genome(), param)
}
```

postFilterVariants *Post-filtering of Variants*

Description

Applies filters to a set of called variants. The only current filter is a cutoff on the weighted neighbor count of each variant. This filtering is performed automatically by callVariants, so these functions are for when more control is desired.

Usage

```
postFilterVariants(x, post.filters = VariantPostFilters(...), ...)
VariantPostFilters(max.nbor.count = 0.1, whitelist = NULL)
```

Arguments

x	A tally GRanges containing called variants, as output by callVariants.
post.filters	The filters applied to the called variants.
...	Arguments passed to VariantPostFilters, listed below.
max.nbor.count	Maximum allowed number of neighbors (weighted by distance)
whitelist	Positions to ignore; these will always pass the filter, and are excluded from the neighbor counting.

Details

The neighbor count is calculated within a 100bp window centered on the variant. Each neighbor is weighted by the inverse square root of the distance to the neighbor. This was motivated by fitting logistic regression models including a term the count (usually 0, 1, 2) at each distance. The inverse square root function best matched the trend in the coefficients.

Value

For postFilterVariants, a tally GRanges of the variants that pass the filters.

For VariantPostFilters, a FilterRules object with the filters.

Author(s)

Michael Lawrence and Jeremiah Degenhardt

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
## post-filters are not enabled by default during calling
data(vignette)
called.variants <- callVariants(tallies_H1993)
## but can be applied at a later time...
postFilterVariants(called.variants, max.nbor.count = 0.15)
```

```
# or enable during calling
called.variants <- callVariants(tallies_H1993,
                                post.filters = VariantPostFilters())
```

---

qaVariants *QA Filtering of Variants*

Description

Filters a tally GRanges through a series of simple checks for strand and read position (read position) biases.

Usage

```
qaVariants(x, qa.filters = VariantQAFilters(...), ...)
VariantQAFilters(fisher.strand.p.value = 1e-4, min.mdfne = 10L)
```

Arguments

x	A tally GRanges as output by tallyVariants.
qa.filters	The filters used for the QA process, typically constructed with VariantQAFilters, see arguments below.
...	Arguments passed to VariantQAFilters, listed below.
fisher.strand.p.value	p-value cutoff for the Fisher's Exact Test for strand bias (+/- counts, alt vs. ref). Any variants with p-values below this cutoff are discarded.
min.mdfne	Minimum allowed median distance of alt calls from their nearest end of the read.

Details

There are currently two QA filters:

- Median distance of alt calls from nearest end of the read is required to be >= min.mdfne, which defaults to 10.
- Fisher's Exact Test for strand bias, using the +/- counts, alt vs. ref. If the null is rejected, the variant is discarded.

Value

For qaVariants, a tally GRanges of the variants that pass the QA checks.

For VariantQAFilters, a FilterRules object with the QA and sanity filters.

Author(s)

Michael Lawrence and Jeremiah Degenhardt

Examples

```
data(vignette)
qaVariants(tallies_H1993, fisher.strand.p.value = 1e-4)
```

`tallyVariants` *Tally the positions in a BAM file*

Description

Tallies the bases, qualities and read positions for every genomic position in a BAM file. By default, this only returns the positions for which an alternate base has been detected. The typical usage is to pass a BAM file, the genome, the (fixed) `readlen` and (if the variant calling should consider quality) an appropriate `high_base_quality` cutoff.

Passing a `which` argument allows computing on only a subregion of the genome. `which` is a 'RangesList' or something coercible to one that limits the tally to that range or set of ranges. By default, the entire genome is processed.

For parallel evaluation (see `BPPARAM`): Specifically, `which` can be a 'GenomicRanges' or a 'GRangesList'. If `which` is a 'GenomicRanges' and has length 1 it is tiled to create chunks for parallel evaluation. If it is longer than 1, each range becomes a chunk for parallel evaluation. If `which` is a 'GRangesList', each element (i.e. each 'GenomicRanges') becomes a chunk. The latter can be useful to ensure balanced worker load, e.g. in the case of regions covering multiple sequences(see `equisplit`).

Usage

```
## S4 method for signature 'BamFile'
tallyVariants(x, param = TallyVariantsParam(...), ...,
                                  BPPARAM = defaultBPPARAM())
## S4 method for signature 'BamFileList'
tallyVariants(x, ...)
## S4 method for signature 'character'
tallyVariants(x, ...)
TallyVariantsParam(genome,
                   read_pos_breaks = NULL,
                   high_base_quality = 0L,
                   minimum_mapq = 13L,
                   variant_strand = 1L, ignore_query_Ns = TRUE,
                   ignore_duplicates = TRUE,
                   mask = GRanges(), keep_extra_stats = TRUE,
                   read_length = NA_integer_,
                   read_pos = !is.null(read_pos_breaks),
                   high_nm_score = NA_integer_,
                   ...)
```

Arguments

x	An indexed BAM file, either a path, BamFile or BamFileList object. If the latter, the tallies are computed separately for each file, and the results are stacked with stackSamples into a single VRanges.
param	The parameters for the tallying process, as a BamTallyParam, typically constructed with TallyVariantsParam, see arguments below.
...	For tallyVariants, arguments to pass to TallyVariantsParam, listed below. For TallyVariantsParam, arguments to pass to BamTallyParam.
genome	The genome, either a GmapGenome or something coercible to one.
read_pos_breaks	The breaks used for tabulating the read positions (read positions) at each position. If this information is included (not NULL), qaVariants will use it during filtering.
high_base_quality	The minimum cutoff for whether a base is counted as high quality. By default, callVariants will use the high quality counts in the likelihood ratio test. Note that bam_tally will shift your quality scores by 33 no matter what type they are. If Illumina (pre 1.8) this will result in a range of 31-71. If Sanger/Illumina1.8 this will result in a range of 0-40/41. The default counts all bases as high quality. We typically use 56 for old Illumina, 23 for Sanger/Illumina1.8.
minimum_mapq	Minimum MAPQ of a read for it to be included in the tallies. This depend on the aligner; the default is reasonable for gsnap.
variant_strand	On how many strands must an alternate base be detected for a position to be returned. Highly recommended to set this to at least 1 (otherwise, the result is huge and includes many uninteresting reference rows).
ignore_query_Ns	Whether to ignore N calls in the reads. Usually, there is no reason to set this to FALSE. If it is FALSE, beware of low quality datasets returning enormous results.
ignore_duplicates	whether to ignore reads flagged as PCR/optical duplicates
mask	A GRanges specifyin a mask; all variants falling within the mask are discarded.
read_length	The expected read length, used for calculating the "median distance from nearest" end statistic. If not specified, an attempt is made to guess the read length from a random sample of the BAM file. If read length is found to be variable, statistics depending on the read length are not calculated.
read_pos	Whether to tally read positions, which can be computationally intensive.
high_nm_score	If not NA, counts of reads with NM (mismatch count) score equal to or greater are returned in the count.high.nm and count.high.nm.ref columns.
keep_extra_stats	Whether to keep various summary statistics generated from the tallies; setting this to FALSE will save memory. The extra statistics are most useful for algorithm diagnostics and development.
BPPARAM	A BiocParallelParam object specifying the resources and strategy for parallelizing the tally operation over the chromosomes.

Value

For `tallyVariants`, the tally `GRanges`.

For `TallyVariantsParam`, an object with parameters suitable for variant calling.

Note

The `VariantTallyParam` constructor is **DEPRECATED**.

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

```
if (requireNamespace("gmapR")) {
    tally.param <- TallyVariantsParam(gmapR::TP53Genome(),
                                      high_base_quality = 23L,
                                      which = gmapR::TP53Which())
    bams <- LungCancerLines::LungCancerBamFiles()
    raw.variants <- tallyVariants(bams$H1993, tally.param)
}
```

---

variantGR2Vcf *Create a VCF for some variants*

Description

The **deprecated** way to create a VCF object from a variant/tally `GRanges`. This can then be output to a file using `writeVcf`. The flavor of VCF is specific for calling variants, not genotypes; see below.

Usage

```
variantGR2Vcf(x, sample.id, project = NULL,
              genome = unique(GenomeInfoDb::genome(x)))
```

Arguments

`x`	The variant/tally `GRanges`.
`sample.id`	Unique ID for the sample in the VCF.
`project`	Description of the project/experiment; will be included in the VCF header.
`genome`	`GmapGenome` object, or the name of one (in the default genome directory). This is used for obtaining the anchor base when outputting indels.

Details

A variant GRanges has an element for every unique combination of position and alternate base. A VCF object, like the file format, has a row for every position, with multiple alternate alleles collapsed within the row. This is the fundamental difference between the two data structures. We feel that the GRanges is easier to manipulate for filtering tasks, while VCF is obviously necessary for communication with external databases and tools.

Normally, despite its name, VCF is used for communicating *genotype* calls. We are calling *variants*, not genotypes, so we have extended the format accordingly.

Here is the mapping in detail:

- The rowRanges is formed by dropping the metadata columns from the GRanges.
- The colData consists of a single column, "Samples", with a single row, set to 1 and named sample.id.
- The exptData has an element "header" with element "reference" set to the seqlevels(x) and element "samples" set to sample.id. This will also include the necessary metadata for describing our extensions to the format.
- The fixed table has the "REF" and "ALT" alleles, with "QUAL" and "FILTER" set to NA.
- The geno list has six matrix elements, all with a single column. The first is the mandatory "GT" element, the genotype, which we set to NA. Then there is "AD" (list matrix with the read count for each REF and ALT), "DP" (integer matrix with the total read count), and "AP" (list matrix of 0/1 flags for whether whether REF and/or ALT was present in the data).

Value

A VCF object.

Note

This function is **DEPRECATED**. The callVariants function now returns a VRanges object that can be coerced to a VCF object via as(x, "VCF").

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

```
## Not run:
vcf <- variantGR2Vcf(variants, "H1993", "example")
writeVcf(vcf, "H1993.vcf", index = TRUE)

## End(Not run)
```

vignette *Vignette Data*

Description

Precomputed data for use in the vignette, mostly for the sake of Windows, where gmapR and its tallying functionality are unsupported.

Usage

```
data(vignette)
```

Format

The following objects are included:

tallies_H1993, tallies_H2073 Tallies for the two samples.

coverage_H1993, coverage_H2073 Coverage for the two samples.

p53 A GRanges of the p53 exons

genome_p53 DNAStringSet with the genome sequence of the p53 region

Details

The following demonstrates how we created these objects:

```
bams <- LungCancerLines::LungCancerBamFiles()
tally.param <- TallyVariantsParam(gmapR::TP53Genome(),
                                  high_base_quality = 23L,
      which = range(p53) + 5e4,
                                  indels = TRUE, read_length = 75L)
tallies_H1993 <- tallyVariants(bams$H1993, tally.param)
tallies_H2073 <- tallyVariants(bams$H2073, tally.param)
coverage_H1993 <- coverage(bams$H1993)
coverage_H2073 <- coverage(bams$H2073)
genome_p53 <- DNAStringSet(getSeq(gmapR::TP53Genome()))
p53 <- gmapR:::exonsOnTP53Genome("TP53")
```

Source

Computed from the data in the LungCancerLines package.

Examples

```
data(vignette)
```

Index

∗ **datasets**
    vignette, 21
%variant_in% *(*matchVariants*)*, 13
%variant_in%,GenomicRanges,GenomicRanges-method *(*matchVariants*)*, 13

annotateWithControlDepth, 2
applyPileups, *14*
ApplyPileupsParam, *14*

bam_tally, *8*, *14*
BamTallyParam, *18*
BiocParallelParam, *18*

calculateConcordanceMatrix *(*concordance*)*, 10
calculateVariantConcordance *(*concordance*)*, 10
callGenotypes, 3
callGenotypes,TabixFile-method *(*callGenotypes*)*, 3
callGenotypes,VRanges-method *(*callGenotypes*)*, 3
CallGenotypesParam, *3*
CallGenotypesParam *(*callGenotypes*)*, 3
callSampleSpecificVariants, *3*, 5
callSampleSpecificVariants,BamFile,BamFile-method *(*callSampleSpecificVariants*)*, 5
callSampleSpecificVariants,character,character-method *(*callSampleSpecificVariants*)*, 5
callSampleSpecificVariants,GenomicRanges,GenomicRanges-method *(*callSampleSpecificVariants*)*, 5
callSampleSpecificVariants,VRanges,VRanges-method *(*callSampleSpecificVariants*)*, 5
callVariantConcordance *(*concordance*)*, 10
callVariants, *3*, *6*, 7, *15*, *18*
callVariants,BamFile-method *(*callVariants*)*, 7
callVariants,character-method *(*callVariants*)*, 7
callVariants,GenomicRanges-method *(*callVariants*)*, 7
callVariants,VRanges-method *(*callVariants*)*, 7
callWildtype, 9
concordance, 10
coverage, *11*
coverage_H1993 *(*vignette*)*, 21
coverage_H2073 *(*vignette*)*, 21

DepthFETFilter *(*FilterConstructors*)*, 12

equisplit, *17*
extractCoverageForPositions, 11

FilterConstructors, 12
FilterRules, *8*, *15*, *16*

genome_p53 *(*vignette*)*, 21
getSeq, *14*
GmapGenome, *18*
gsnap, *18*

matchVariants, 13
MaxControlFreqFilter *(*FilterConstructors*)*, 12
minCallableCoverage *(*callWildtype*)*, 9
MinTotalDepthFilter *(*FilterConstructors*)*, 12

p53 *(*vignette*)*, 21
PileupFiles, *14*
pileupVariants, 14
postFilterVariants, 15

qaVariants, 16, *18*

SampleSpecificVariantFilters *(*callSampleSpecificVariants*)*, 5
SetdiffVariantsFilter *(*FilterConstructors*)*, 12

stackSamples, *18*

tallies_H1993 *(*vignette*)*, 21
tallies_H2073 *(*vignette*)*, 21
tallyVariants, *3*, *6*, *8*, *14*, *16*, 17
tallyVariants,BamFile-method
*(*tallyVariants*)*, 17
tallyVariants,BamFileList-method
*(*tallyVariants*)*, 17
tallyVariants,character-method
*(*tallyVariants*)*, 17
TallyVariantsParam, *6*, *8*
TallyVariantsParam *(*tallyVariants*)*, 17

VariantCallingFilters, *6*, *9*, *12*
VariantCallingFilters *(*callVariants*)*, 7
variantGR2Vcf, 19
VariantPostFilters
*(*postFilterVariants*)*, 15
VariantQAFilters, *6*, *12*
VariantQAFilters *(*qaVariants*)*, 16
VariantTallyParam *(*tallyVariants*)*, 17
VCF, *19*
vignette, 21
VRanges, *20*

writeVcf, *19*