Package 'VariantTools'

October 9, 2013

Type Package

Title Tools for Working with Genetic Variants

Version 1.2.2

Author Michael Lawrence, Jeremiah Degenhardt, Robert Gentleman

Maintainer Michael Lawrence <michafla@gene.com>

Description Tools for detecting, filtering, calling,comparing and plotting variants.

Depends IRanges (>= 1.17.10), GenomicRanges (>= 1.9.52),VariantAnnotation (>= 1.3.20), methods

Imports IRanges, Rsamtools (>= 1.11.10), GenomicRanges, BiocGenerics,Biostrings, parallel, gmapR (>= 1.1.16), GenomicFeatures,VariantAnnotation, methods, RBGL, graph, Matrix, rtracklayer

Suggests RUnit, LungCancerLines (>= 0.0.6)

biocViews Genetics, GeneticVariability, HighThroughputSequencing

License Artistic-2.0

LazyLoad yes

R topics documented:

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

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

Arguments

case	The BAM file for the case, or the raw tallies as output by tallyVariants.
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 VariantTallyParam.
qa.filters	Filters to use in the QA process, typically generated by VariantQAFilters.
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 `VariantTallyParam`, `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 tally `GRanges` with the case-specific variants (such as somatic mutations).

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
tally.param <- VariantTallyParam(gmapR::TP53Genome(),
                                 readlen = 100L,
                                 high_base_quality = 23L,
                                 which = gmapR::TP53Which())
callSampleSpecificVariants(bams$H1993, bams$H2073, tally.param)
```

`callVariants` *Call Variants*

Description

Calls variants from either a BAM file or a tally `GRanges` object. The variants are first filtered with `qaVariants`, and the remaining candidates 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,
                                qa.filters = VariantQAFilters(),
                                calling.filters = VariantCallingFilters(...),
                                post.filters = VariantPostFilters(),
                                ...)
## S4 method for signature 'character'
callVariants(x, ...)
## S4 method for signature 'GenomicRanges'
callVariants(x,
     calling.filters = VariantCallingFilters(...),
     post.filters = VariantPostFilters(),
     ...)
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 GRanges as returned by tallyVariants.
tally.param	Parameters controlling the variant tallying step, as typically constructed by VariantTallyParam.
qa.filters	Filters used in the QA step, see VariantQAFilters.
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).
...	Arguments to pass to VariantCallingFilters.

Details

After QA filtering, there are two remaining 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 that removes variants that are clumped together on the chromosome (see the max.nbor.count parameter).

Value

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

For `VariantCallingFilters`, a `FilterRules` object with the filters for calling the variants (presumably after the QA filters have been applied).

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

```
bams <- LungCancerLines::LungCancerBamFiles()
tally.param <- VariantTallyParam(gmapR::TP53Genome(),
                                 readlen = 100L,
                                 high_base_quality = 23L,
                                 which = gmapR::TP53Which())

## simple usage
variants <- callVariants(bams$H1993, tally.param)

## customize
qa.filters <- VariantQAFilters(fisher.strand.p.value = 1e-4)
calling.filters <- VariantCallingFilters(p.error = 1/1000)
callVariants(bams$H1993, tally.param, qa.filters, 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

```
p53 <- gmapR:::exonsOnTP53Genome("TP53")
bams <- LungCancerLines::LungCancerBamFiles()
bam <- bams$H1993
tally.param <- VariantTallyParam(gmapR::TP53Genome(),
                                 readlen = 100L,
                                 high_base_quality = 23L,
                                 which = range(p53))
called.variants <- callVariants(bam, tally.param)

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 or VCF files so that all variant calls are not loaded in memory at once.

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)

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

matchVariants *Match variants by position and allele*

Description

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

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

```
p53 <- gmapR:::exonsOnTP53Genome("TP53")
bams <- LungCancerLines::LungCancerBamFiles()
tally.param <- VariantTallyParam(gmapR::TP53Genome(),
                                 readlen = 100L,
                                 high_base_quality = 23L,
                                 which = range(p53))
# disable post-filtering during variant calling
called.variants <- callVariants(bams[[1]], tally.param,
                                post.filters = FilterRules())
# and apply at a later time...
postFilterVariants(called.variants, max.nbor.count = 0.15)
```

---

qaVariants *QA Filtering of Variants*

Description

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

Usage

```
qaVariants(x, qa.filters = VariantQAFilters(...), ...)
VariantQAFilters(cycle.count = 2L, fisher.strand.p.value = 1e-4,
                 read.pos.p.value = 1e-4, mask = GRanges())
```

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.
cycle.count	Minimum number of unique cycles for the alternate base.
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.
read.pos.p.value	p-value cutoff for the read position t-test between the variant and reference calls
mask	A GRanges with regions to exclude from consideration; e.g., simple repeats. Strand is not considered.

Details

There are currently three QA filters:

- Alternate base was read at a minimum (2) number of unique cycles. This avoids false positives from one aberrant cycle.
- Fisher's Exact Test for strand bias, using the +/- counts, alt vs. ref. If the null is rejected, the variant is discarded.
- If the tallies contain cycle bin counts, the variant must have at least one count in the middle bins (those not at the start or end). We trust the internal cycles more.
- Read position t-test comparing the mean read position for the reference and alt reads. Any imbalance probably indicates mapping issues.
- Mask of blacklisted positions, such as simple repeats, low complexity regions, i.e., uninteresting, problematic regions.

Prior to the QA checks, the variants are passed through a simple sanity filter that discards positions where reference has an N.

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

```
bams <- LungCancerLines::LungCancerBamFiles()
tally.param <- VariantTallyParam(gmapR::TP53Genome(),
                                 readlen = 100L,
                                 high_base_quality = 23L,
                                 which = gmapR::TP53Which())
tally.variants <- tallyVariants(bams$H1993, tally.param)
qaVariants(tally.variants, 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.

Usage

```
## S4 method for signature 'BamFile'
tallyVariants(x, param = VariantTallyParam(...), ...,
                                    mc.cores = getOption("mc.cores", 2))
## S4 method for signature 'character'
tallyVariants(x, ...)
VariantTallyParam(genome, readlen = NA,
                  cycle_flank_width = 10L,
                  cycle_breaks = flankingCycleBreaks(readlen,
                  cycle_flank_width),
                  high_base_quality = 0L,
                  minimum_mapq = 13L,
                  variant_strand = 1L, ignore_query_Ns = TRUE,
                  ignore_duplicates = TRUE,
                  ...)
```

Arguments

`x`	An indexed BAM file, either a path or a `BamFile` object.
`param`	The parameters for the tallying process, as a `BamTallyParam`, typically constructed with `VariantTallyParam`, see arguments below.
`...`	For `tallyVariants`, arguments to pass to `VariantTallyParam`, listed below. For `VariantTallyParam`, arguments to pass to `BamTallyParam`.
`genome`	The genome, either a `GmapGenome` or something coercible to one.

readlen, cycle_flank_width
: If cycle_breaks is missing, these two arguments are used to generate a cycle_breaks for three bins, with the two outside bins having cycle_flank_width. If readlen is NA, cycle_breaks is not generated.

cycle_breaks
: The breaks used for tabulating the cycles (read positions) at each position. If this information is included (not NULL), qaVariants will use it during filtering.

high_base_quality
: The 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

mc.cores
: The number of cores to use when parallelizing over the chromosomes.

Value

For tallyVariants, the tally GRanges.

For VariantTallyParam, an object with parameters suitable for variant calling.

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

```
tally.param <- VariantTallyParam(gmapR::TP53Genome(),
                                 readlen = 100L,
                                 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

Creates 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(GenomicRanges::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 rowData 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.

Author(s)

Michael Lawrence, Jeremiah Degenhardt

Examples

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

## End(Not run)
```

Index