Package ‘MTseeker’

October 16, 2019

Type Package

Title Bioconductor Tools for Human Mitochondrial Variant Analysis

Version 1.2.0

Author
Tim Triche [cre, aut], Noor Sohail [aut], Ben Johnson [aut], Tim Vickers [ctb], Azif Zubair [ctb]

Maintainer Tim Triche <`tim.triche@gmail.com`>

Description Variant analysis tools for mitochondrial genetics.

biocViews Genetics, Metabolomics, VariantAnnotation

Depends viridis, S4Vectors, GenomeInfoDb, GenomicAlignments,
VariantAnnotation

Imports xml2, utils, gmapR, methods, IRanges, Biobase, circlize,
jsonlite, graphics, Rsamtools, grDevices, Biostrings,
rtracklayer, VariantTools, Homo.sapiens, BiocGenerics,
GenomicRanges, GenomicFeatures, SummarizedExperiment

Suggests MTseekerData, BiocStyle, rmarkdown, ggthemes, ggplot2,
pkgdown, knitr, rsvg

License GPL-3

Encoding UTF-8

RoxygenNote 6.1.1

VignetteBuilder knitr

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

git_branch RELEASE_3_9

git_last_commit 51756bd

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

R topics documented:

`anno_rCRS` *annotation for the regions (genic and not) of the rCRS mitogenome*

Description

annotation for the regions (genic and not) of the rCRS mitogenome

Usage

```
anno_rCRS
```

Format

a GRanges object

Examples

```
data(anno_rCRS)
subset(anno_rCRS, region == "D-loop")
```

byStrand — *simple helper function to split a *RangesList by the strand of its mt target*

Description

If presented with a GAlignments/MAlignments, this method will split the element by strand, i.e. + alignments and - alignments. Otherwise the method retrieves ranges/variant calls that overlap genic elements on the heavy and light strands of the mitochondrial genome.

Usage

```
byStrand(x, anno = NULL)
```

Arguments

x	a *Ranges[List] or *Alignments[List]
anno	optional feature annotation, will use mtAnno.rCRS if NULL

Value

elements of x over features on each strand OR x split by strand

Examples

```
data(RONKSvariants, package="MTseekerData")
byStrand(RONKSvariants)
```

callMT — *call mitochondrial variants against rCRS from an MAlignments[List] object*

Description

‘callMTVars‘ is a helper function for callMT

Usage

```
callMT(mal, ..., parallel = FALSE, verbose = FALSE)

callMTVars(BAM, SIZE = 75, GENOME = "rCRS", CHR = "chrM",
  COV = NULL, verbose = FALSE)
```

Arguments

`mal`	an MAlignments (or, potentially, an MAlignmentsList)
`...`	other arguments to pass to VariantTools::callVariants
`parallel`	try to run in parallel? (FALSE; this is super unstable)
`verbose`	be verbose? (FALSE; turn on for debugging purposes)
`BAM`	the BAM filename (for callMTVars)
`SIZE`	the read length (for callMTVars; default is 75)
`GENOME`	the reference genome (for callMTVars; default is rCRS)
`CHR`	the mt contig name (for callMTVars; default is chrM)
`COV`	average read coverage (so we don't have to countBam)

Details

FIXME: transition gmapR from import to suggestion FIXME: use Rsamtools::pileup by default FIXME: optional haplogroup masking?

Value

an MVRanges (or, potentially, an MVRangesList)

Examples

```
library(MTseekerData)
BAMdir <- system.file("extdata", "BAMs", package="MTseekerData")
BAMs <- paste0(BAMdir, "/", list.files(BAMdir, pattern=".bam$"))
(mal <- getMT(BAMs[1]))
if (requireNamespace("GmapGenome.Hsapiens.rCRS", quietly=TRUE)) {
  (mvr <- callMT(mal))
  filt(snpCall(mvr))
} else {
  message("You have not yet installed an rCRS reference genome.")
  message("Consider running the indexMTgenome() function to do so.")
  message("The RONKSvariants object in MTseekerData is a result of callMT.")
}
```

`chrominfo.rCRS` *Sequence information (seqinfo) for the rCRS mitogenome*

Description

Sequence information (seqinfo) for the rCRS mitogenome

Usage

```
chrominfo.rCRS
```

Format

a Seqinfo object

Examples

```
data(chrominfo.rCRS)
library(GenomeInfoDb) # wat
seqlengths(chrominfo.rCRS)
```

`filterMT` *Filter SummarizedExperiment or DataFrame values based on its $mtCovg column*

Description

Griffin et al. (Genetics in Medicine 2014) recommends 20x coverage for mtDNA sequencing to have comparable error rates to Sanger sequencing. By default, that is the cutoff applied here to ensure halfway decent variant annotation.

Usage

```
filterMT(DFSE, minCovg = 20, fpFilter = FALSE, NuMT = FALSE)
```

Arguments

`DFSE`	a DataFrame/SummarizedExperiment with colData()$‘mtCovg‘
`minCovg`	minimum covg (20, cf. Griffin, Genetics in Medicine 2014)
`fpFilter`	apply Triska’s homopolymer false positive filter? (FALSE)
`NuMT`	apply the 0.03 VAF NuMT filter from Ju (GR 2015)? (FALSE)

Details

Triska (Cancer Res, in revision) suggests a small number of masked regions where homopolymers can be a problem; these are avoided if fpFilter

The NuMT filtration step (Ju, in eLife 2014, suggests a variant allele cutoff of 0.03 to avoid false positive calls from nuclear-mitochondrial translocated or ’NuMT’ fragments) is also a useful tool to cut down on nonsensical calls, although it may be important to use caution as low heteroplasmy can also resolve into apparent near-homoplasmy at the single-cell level, at least in our (TJT & co) experience.

As a consequence of the Wild West nature for published methods of high- throughput mitochondrial sequence variant analysis at the time of writing (2018), the default for this function is to filter on coverage only, as the user is expected to determine what additional filters to apply. We could envision changing these defaults down the road as standards congeal.

If DFSE is an MVRanges[List], the function will call filterMTvars instead.

Value

a filtered SE or data.frame

Examples

```
filterMT(data.frame(sample="foo", mtCovg=1000))
```

filterMTvars *sanitize PASSing mitochondrial variant calls to a moderate degree*

Description

sanitize PASSing mitochondrial variant calls to a moderate degree

Usage

```
filterMTvars(vars, fp = TRUE, NuMT = 0.03, covg = 20)
```

Arguments

`vars`	an MVRanges or MVRangesList (will be unlisted and relisted)
`fp`	use false positive filter[s]? (TRUE: use Triska fpFilter)
`NuMT`	variants with VAF < [this number] will be presumed NuMTs (0.03)
`covg`	minimum median read coverage across chrM to be considered (20)

Value

a filtered set of variants

Examples

```
library(MTseekerData)
filterMTvars(RONKSvariants$RO_1)
```

fixMetadata *fix metadata for an MAlignmentsList or MVRangesList, if needed and possible.*

Description

This function is punishingly simple – it just calls validMetadata() and assigns attr('fixedMeta') from the result. If no fixing is required, the object's existing metadata is used. The object (with fixed metadata) is then returned. If automatic fixes are impossible, a message is generated.

Usage

```
fixMetadata(x)
```

Arguments

`x`	an MAlignmentsList or MVRangesList

Value

the object, with fixed metadata (if needed and possible)

Examples

```
library(MTseekerData)
data(RONKSreads)
fixed <- fixMetadata(RONKSreads)
```

`fpFilter_RSRS` *false positive filter (fpFilter) for RSRS haplogroup-determining variants*

Description

false positive filter (fpFilter) for RSRS haplogroup-determining variants

Usage

```
fpFilter_RSRS
```

Format

a GRanges object

Examples

```
data(fpFilter_RSRS)
subset(fpFilter_RSRS, !is.na(L0))
subset(fpFilter_RSRS, !is.na(rCRS))
```

`fpFilter_Triska` *a false positive (fp) filter from Petr Triska's manuscript on mtDNA variants*

Description

a false positive (fp) filter from Petr Triska's manuscript on mtDNA variants

Usage

```
fpFilter_Triska
```

Format

a GRanges object

Examples

```
data(fpFilter_Triska)
show(fpFilter_Triska)
```

`getMT` *grab the mitochondrial reads from a BAM & estimate their fraction (of total)*

Description

This purely a convenience function, and an incredibly convenient one at that.

Usage

```
getMT(bam, filter = FALSE, parallel = FALSE, plotMAPQ = FALSE, ...)
```

Arguments

`bam`	a BAM filename, or DataFrame/SummarizedExperiment with $BAM
`filter`	filter on bam$mtCovg? (default is FALSE, don't filter)
`parallel`	load multiple BAMs in parallel, if possible? (FALSE)
`plotMAPQ`	plot distribution of mitochondrial mapping quality? (FALSE)
`...`	additional args to pass scanBamParam(), such as mapqFilter

Value

an MAlignments or MAlignmentsList object

Examples

```
library(MTseekerData)
BAMdir <- system.file("extdata", "BAMs", package="MTseekerData")
BAMs <- paste0(BAMdir, "/", list.files(BAMdir, pattern=".bam$"))
(mal <- getMT(BAMs[1]))
class(mal)

targets <- data.frame(BAM=BAMs, stringsAsFactors=FALSE)
rownames(targets) <- sapply(strsplit(basename(BAMs), "\\."), `[`, 1)
(mall <- getMT(targets))
class(mall)
```

`hg19TorCRS` *a liftOver chain for hg19 to rCRS attempts. We do not recommend this.*

Description

a liftOver chain for hg19 to rCRS attempts. We do not recommend this.

Usage

```
hg19TorCRS
```

Format

a Chain object

Examples

```
data(hg19TorCRS)
sapply(hg19TorCRS, offset)
```

---

indexMTGenome — *build and install GmapGenome.[organism].[mtGenome], currently Hsapiens.rCRS*

Description

gmapR needs a reference genome index in order for it to call any variants. We support rCRS (and only rCRS) as that reference, at least for the moment. This function creates & installs a reference (rCRS, the default) Gmap index. In principle, hg19 and mm10 could be supported; in practice, support is poor. (Also, the Yoruban chrM in hg19 is a terrible reference for variant calling.) We would be grateful for a patch to add mm10/GRCm38 support; eventually, we plan to add it in ourselves (as one might have guessed from Mouse Mitocarta).

Usage

```
indexMTGenome(mtGenome = "rCRS", fa = NULL, organism = "Hsapiens",
  destDir = NULL, install = TRUE, unlink = FALSE)
```

Arguments

`mtGenome`	mitochondrial reference genome to index (default is rCRS)
`fa`	FASTA file (default is to find included ‘mtGenome‘.fa)
`organism`	organism whose mitochondrial genome is indexed (Hsapiens)
`destDir`	optional destination for the package ($HOME is default)
`install`	install the package after creation? (default is TRUE)
`unlink`	if an index package already exists, remove it? (FALSE)

Details

Note: this function creates a "skeleton key" rCRS index for contigs named ’chrM’, ’MT’, ’rCRS’, ’NC_012920.1’, and/or ’gi|251831106|ref|NC_012920.1|’. The point of this kludge is to allow gmapR to call variants against various styles of contig names, whether NCBI, UCSC, Genbank, or colloquial rCRS.

Value

the path to the created package as a character string

Examples

```
if (.Platform$OS.type != "windows") {
  mtGenome <- "rCRS"
  fa <- system.file(paste0("extdata/", mtGenome, ".fa"), package="MTseeker")
  indexMTGenome(mtGenome=mtGenome, fa=fa, destDir=tempdir())
}
```

`injectMTVariants` *Inject (one or more) variants against rCRS.*

Description

FIXME: this function could most likely be orders of magnitude faster. FIXME: this ONLY considers variants injected against rCRS, not RSRS or hg19.

Usage

```
injectMTVariants(mvr, gr = NULL, aa = TRUE, canon = 0.99, refX = 1,
  altX = 1)
```

Arguments

`mvr`	An MVRanges, usually from callMT, often subsetted
`gr`	A GRanges, usually of protein-coding regions (the default)
`aa`	Attempt to translate codon(s) affected by variant(s)? (TRUE)
`canon`	Minimum VAF to treat variants as canonical by subject (0.99)
`refX`	Reference depth below which variant is deemed canonical (1)
`altX`	Alternative depth above which variants deemed canonical (1)

Value

The GRanges, with ref/var DNA and AA and

Examples

```
library(MTseekerData)
RO_2 <- RONKSvariants[["RO_2"]]
injectMTVariants(RO_2)
```

`MAlignments` *wrap a GAlignments for easier stats*

Description

Normally the MAlignments constructor will be called by getMT(bam).

Depending on how a generic was originally designated, the arguments to these methods can have various argument names, but all of them tend to take an MAlignments as their argument.

Usage

```
MAlignments(gal, bam)

genomeCoverage(x)

## S4 method for signature 'MAlignments'
Summary(x)

## S4 method for signature 'MAlignments'
show(object)

## S4 method for signature 'MAlignments'
fileName(object)

## S4 method for signature 'MAlignments'
scanBamHeader(files)

readLength(x)

## S4 method for signature 'MAlignments'
readLength(x)

genomeLength(x)

## S4 method for signature 'MAlignments'
genomeLength(x)
```

Arguments

`gal`	a GAlignments
`bam`	a bam filename
`x`	an MAlignments
`object`	an MAlignments
`files`	an MAlignments

Value

an MAlignments

various things, as appropriate to the methods

Examples

```
library(MTseekerData)
BAMdir <- system.file("extdata", "BAMs", package="MTseekerData")
BAMs <- paste0(BAMdir, "/", list.files(BAMdir, pattern=".bam$"))
mal <- getMT(BAMs[1])
class(mal)
show(mal)
```

`MAlignments-class` *wraps a GAlignments with information about coverage and its target BAM file*

Description

The runLength slot stores readLength and named for historic reasons.

`MAlignmentsList` *wrap a GAlignmentsList for viewing*

Description

Normally the MAlignmentsList constructor will be called by getMT.

Depending on how a generic was originally designated, the arguments to these methods can have various argument names, but all of them tend to take an MAlignmentsList as their argument.

Usage

```
MAlignmentsList(...)

## S4 method for signature 'MAlignmentsList'
genomeCoverage(x)

## S4 method for signature 'MAlignmentsList'
readLength(x)

## S4 method for signature 'MAlignmentsList'
fileName(object)

## S4 method for signature 'MAlignmentsList'
Summary(x)

## S4 method for signature 'MAlignmentsList'
show(object)
```

Arguments

`...`	MAlignments
`x`	an MAlignmentsList
`object`	an MAlignmentsList

Value

an MAlignments

various objects, as appropriate to the method

Examples

```
library(MTseekerData)
BAMdir <- system.file("extdata", "BAMs", package="MTseekerData")
print(BAMdir)
BAMs <- paste0(BAMdir, "/", list.files(BAMdir, pattern=".bam$"))
print(BAMs)
targets <- data.frame(BAM=BAMs, stringsAsFactors=FALSE)
rownames(targets) <- sapply(strsplit(basename(BAMs), "\\."), `[`, 1)
mall <- getMT(targets)
class(mall)
show(mall)
```

`MAlignmentsList-class` *wraps a GAlignmentsList (made up of MAlignments) for nicer viewing*

Description

wraps a GAlignmentsList (made up of MAlignments) for nicer viewing

---

`mtAnno` *annotation for the rCRS genome*

Description

annotation for the rCRS genome

Usage

```
mtAnno
```

Format

a GRanges object

MTcircos *plot a canonical human (or, in principle, any) mitochondrial genome*

Description

The default font sizes, orientations, etc. are optimized for a "cold" start; if you want to fiddle with the details, crack open the code and modify it... or alternatively, add sectors/dendrograms inside of this "framed" version.

Usage

```
MTcircos(variants = NULL, outside = NULL, inside = NULL,
  outcol = NULL, incol = NULL, anno = NULL, how = c("matrix",
  "VAF"), ...)
```

Arguments

`variants`	optional MVRanges or MVRangesList to split by strand & plot
`outside`	optional MVRanges or MVRangesList to plot outside the circle
`inside`	optional MVRanges or MVRangesList to plot inside the circle
`outcol`	optional color assignment function or matrix for outside
`incol`	optional color assignment function or matrix for inside
`anno`	a GRanges (optional, defaults to mtAnno.rCRS if none given)
`how`	optional specification for how to plot multiple samples
`...`	other arguments to pass on to called functions

Details

FIXME: add variant type coloration (del=blue, SNV=black, ins=red)

Value

invisibly, a list: ‘anno‘ (data.frame) + ‘pfun‘ (panel.fun)

Examples

```
library(MTseekerData)
data(RONKSvariants)
MTcircos(RONKSvariants)
# same as plot(RONKSvariants)
title("Renal oncocytomas and normal kidney samples")
```

`MTcomplex` *plot the (putative) functional impact of mutations to ETP genes as SVG*

Description

Tim Vickers created a beautiful illustration of the mitochondrial electron transport chain, and that's where coding mitochondrial DNA mutations will usually hit (we aren't plotting the mitoribosome or tRNAs just yet). So why reinvent the wheel (and possibly make it square)?

Usage

```
MTcomplex(variants, defColor = "#c9eded", verbose = FALSE)
```

Arguments

`variants`	an MVRanges or MVRangesList
`defColor`	default color (#c9eded is standard)
`verbose`	be verbose, for debugging? (FALSE)

Value

invisibly, the temporary file to which the SVG was written

Examples

```
library(MTseekerData)
data(RONKSvariants)
MTcomplex(RONKSvariants$RO_1)
```

`MTcoverage` *Mitochondrial genome coverage and plots for MAlignments or MVRanges objects*

Description

We co-opted the 'coverage' method to retrieve approximate coverage depth across the mitochondrial genome in MAlignments[List] and MVRanges[list], so this function gives back what it was supposed to do (provide an Rle) and can allow for some subsetting (e.g. variant-supporting-read coverage) that may be of interest when interpreting results.

Usage

```
MTcoverage(x, ...)

plotMTCoverage(x, ...)

plotStrandedMTCoverage(x, ...)
```

Arguments

x	an MAlignments or MVRanges
...	other arguments to pass to GenomicAlignments::coverage()

Details

The plotting functions can handle MAlignments or MVRanges objects directly.

plotMTCoverage does what one might expect, and plots (read or call) coverage.

plotStrandedMTCoverage does the same thing, but keeps track of which strand.

Value

an RleList (or, invisibly for plot functions, a result list)

Examples

```
library(MTseekerData)

data(RONKSreads)
MTcoverage(RONKSreads$RO_1)
plotMTCoverage(RONKSreads$RO_1)

data(RONKSvariants)
MTcoverage(RONKSvariants$RO_1)
plotMTCoverage(RONKSvariants$RO_1)

par(mfrow=c(1,2))
plotMTCoverage(RONKSreads$NKS_1)
title("Read coverage for normal kidney sample 1")
plotMTCoverage(RONKSreads$RO_1)
title("Read coverage for renal oncocytoma sample 1")

par(mfrow=c(1,2))
plotStrandedMTCoverage(RONKSreads$NKS_1)
title("Stranded read coverage for normal kidney sample 1")
plotStrandedMTCoverage(RONKSreads$RO_1)
title("Stranded read coverage for renal oncocytoma sample 1")
```

---

mtGenes *Base sequences of mitochondrial coding genes.*

Description

Base sequences of mitochondrial coding genes.

Usage

```
mtGenes
```

Format

a GRanges with a DNAStringSet of the base sequences as its mcols.

Examples

```
data(mtGenes)
width(mtGenes)
names(mtGenes$DNA)
```

---

mtGenes.rCRS — *All annotated gene regions (not just coding genes) from rCRS.*

Description

All annotated gene regions (not just coding genes) from rCRS.

Usage

```
mtGenes.rCRS
```

Format

a GRanges

Examples

```
data(mtGenes.rCRS)
subset(mtGenes.rCRS, region == "coding")
subset(mtGenes.rCRS, region == "tRNA")
```

MTHGVS — *convert mitochondrial variant calls to HGVS format for naming*

Description

convert mitochondrial variant calls to HGVS format for naming

Usage

```
MTHGVS(x, asMVR = FALSE, verbose = FALSE)
```

Arguments

`x`	an MVRanges (or, in a pinch, a GRanges)
`asMVR`	return a renamed MVRanges? (default is FALSE)
`verbose`	be yappy? (default is FALSE)

Value

proper HGVS names for the *Ranges (or a renamed *Ranges)

Examples

```
library(MTseekerData)
data(RONKSvariants)
MTHGVS(RONKSvariants$RO_1)
```

`MVRanges` *wrap a VRanges for mitochondrial use*

Description

Usually the MVRanges constructor will be called by callMT().

Many of these methods can be dispatched from an MVRangesList OR an MVRanges. In such cases, the method will usually, but not always, be apply()ed.

Usage

```
MVRanges(vr, coverage = NA_real_)

## S4 method for signature 'MVRanges'
genomeCoverage(x)

## S4 method for signature 'MVRanges'
coverage(x)

## S4 method for signature 'MVRanges'
type(x)

## S4 method for signature 'MVRanges'
genes(x)

## S4 method for signature 'MVRanges'
snpCall(object)

## S4 method for signature 'MVRanges'
pos(x)

## S4 method for signature 'MVRanges'
show(object)

## S4 method for signature 'MVRanges'
annotation(object)

## S4 method for signature 'MVRanges'
getAnnotations(annotations)

## S4 method for signature 'MVRanges'
encoding(x)

## S4 method for signature 'MVRanges'
```

```
filt(x)

## S4 method for signature 'MVRanges'
genome(x)

## S4 method for signature 'MVRanges,missing,missing'
locateVariants(query,
  filterLowQual = FALSE, ...)

## S4 method for signature 'MVRanges'
tallyVariants(x, filterLowQual = TRUE, ...)

## S4 method for signature 'MVRanges,missing,missing,missing'
predictCoding(query, subject,
  seqSource, varAllele, ...)

## S4 method for signature 'MVRanges,missing,missing'
summarizeVariants(query, subject,
  mode, ...)

## S4 method for signature 'MVRanges,ANY'
plot(x, y, ...)

## S4 method for signature 'MVRanges'
consensusString(x, ...)
```

Arguments

`vr`	the VRanges
`coverage`	estimated coverage
`x`	an MVRanges
`object`	an MVRanges
`annotations`	an MVRanges
`query`	an MVRanges
`filterLowQual`	boolean; drop non-PASSing variants from locateVariants?
`...`	miscellaneous args, passed through
`subject`	a GRanges, usually
`seqSource`	a BSgenome, usually
`varAllele`	variant alleles
`mode`	miscellaneous arguments
`y`	another MVRanges

Value

an MVRanges

depends on the method invoked.

Utility methods

'pos' returns a character vector describing variant positions. 'filt' returns a subset of variant calls where PASS == TRUE (i.e. filtered) 'coverage' returns an Rle of coverage across the mitochondrial genome 'genomeCoverage' returns the estimated mitochondrial read coverage depth

Annotation methods

'type' returns a character vector describing variant type (SNV or indel) 'genes' retrieves a GRanges of mitochondrial gene locations for an MVRanges 'snpCall' retrieves single nucleotide variant polymorphisms PASSing filters 'annotation' gets (perhaps oddly) an MVRanges object annotated against rCRS 'getAnnotations' returns the GRanges of gene/region annotations for an MVR 'encoding' returns variants residing in coding regions (consequence unknown) 'locateVariants' annotates variants w/region, gene, and localStart/localEnd 'predictCoding' returns variants consequence predictions as one might expect 'tallyVariants' returns a named vector of variant types by annotated region. 'summarizeVariants' uses MitImpact to attempt annotation of coding variants. 'consensusString' edits rCRS to create a consensus genotype for eg Haplogrep

Visualization methods

'plot' creates a circular plot of mitochondrial variant calls with annotation

Examples

```
library(MTseekerData)
BAMdir <- system.file("extdata", "BAMs", package="MTseekerData")
BAMs <- paste0(BAMdir, "/", list.files(BAMdir, pattern=".bam$"))
(mal <- getMT(BAMs[1]))
if (requireNamespace("GmapGenome.Hsapiens.rCRS", quietly=TRUE)) {
  (mvr <- callMT(mal))
  locateVariants(mvr)
  predictCoding(mvr)
} else {
  message("You have not yet installed an rCRS reference genome.")
  message("Consider running the indexMTgenome() function to do so.")
  message("An example MVRanges is RONKSvariants$RO_1 from MTseekerData.")
}

# summarizeVariants can take too long to run, and requires internet access
```

MVRanges-class *like a VRanges, but for mitochondria*

Description

like a VRanges, but for mitochondria

`MVRangesList` *Wrap a VRangesList for mitochondrial use.*

Description

Usually an MVRangesList will be created by callMT.

Usage

```
MVRangesList(...)

## S4 method for signature 'MVRangesList'
genomeCoverage(x)

## S4 method for signature 'MVRangesList'
genes(x)

## S4 method for signature 'MVRangesList'
snpCall(object)

## S4 method for signature 'MVRangesList'
getAnnotations(annotations)

## S4 method for signature 'MVRangesList'
encoding(x)

## S4 method for signature 'MVRangesList'
coverage(x)

## S4 method for signature 'MVRangesList,missing,missing,missing'
predictCoding(query,
  subject, seqSource, varAllele, ...)

## S4 method for signature 'MVRangesList'
show(object)

## S4 method for signature 'MVRangesList'
filt(x)

## S4 method for signature 'MVRangesList'
granges(x, filterLowQual = TRUE)

## S4 method for signature 'MVRangesList,missing,missing'
summarizeVariants(query,
  filterLowQual = TRUE, ...)

## S4 method for signature 'MVRangesList'
genome(x)

## S4 method for signature 'MVRangesList,missing,missing'
locateVariants(query,
    filterLowQual = TRUE, ...)

## S4 method for signature 'MVRangesList,ANY'
plot(x, y, ...)

## S4 method for signature 'MVRangesList'
consensusString(x, ...)
```

Arguments

`...`	miscellaneous args, passed through
`x`	an MVRangesList (for some methods)
`object`	an MVRangesList (for other methods)
`annotations`	an MVRangesList (for getAnnotations)
`query`	an MVRangesList (for predictCoding)
`subject`	a GRanges, usually
`seqSource`	a BSgenome, usually
`varAllele`	variant alleles
`filterLowQual`	opt. for ‘granges‘/‘summarizeVariants‘
`y`	another MVRangesList

Value

the MVRangesList

depends on the method invoked.

Utility methods

‘genomeCoverage‘ returns estimated mitochondrial read coverage depth ‘coverage‘ returns an RleList of coverage for each sample’s chrM ‘filt‘ removes variants where PASS != TRUE for each element

Annotation methods

‘genes‘ returns an annotated GRanges of mitochondrial genes ‘getAnnotations‘ returns a GRanges of annotated mitochondrial features ‘genome‘ returns the genome (or, perhaps, genomes) in an MVRL ‘encoding‘ returns mutations in coding regions for each element ‘granges‘ returns mildly annotated aggregates of variant sites ‘snpCall‘ retrieves single nucleotide variant polymorphisms ‘locateVariants‘ locates variants within genes, tRNA, rRNA, or D-loop ‘summarizeVariants‘ attempts mass functional annotation of variant sites ‘consensusString‘ creates consensus genotypes from rCRS for eg Haplogrep

Visualization methods

‘plot‘ creates circular plot of mitochondrial variant calls

Examples

```
library(MTseekerData)
BAMdir <- system.file("extdata", "BAMs", package="MTseekerData")
BAMs <- paste0(BAMdir, "/", list.files(BAMdir, pattern=".bam$"))
targets <- data.frame(BAM=BAMs, stringsAsFactors=FALSE)
rownames(targets) <- sapply(strsplit(basename(BAMs), "\\."), `[`, 1)
(mall <- getMT(targets))

if (requireNamespace("GmapGenome.Hsapiens.rCRS", quietly=TRUE)) {
  (mvrl <- callMT(mall))
  filt(mvrl$pt1_cell1)
} else {
  message("You have not yet installed an rCRS reference genome.")
  message("Consider running the indexMTgenome() function to do so.")
  message("An example MVRangesList is RONKSvariants from MTseekerData.")
}
```

`MVRangesList-class` *like a VRangesList, but for mitochondria*

Description

like a VRangesList, but for mitochondria

`rCRSeq` *The complete sequence of the human rCRS mitogenome. Yes, it's that small.*

Description

The complete sequence of the human rCRS mitogenome. Yes, it's that small.

Usage

```
rCRSeq
```

Format

a DNAStringSet of length 1

Examples

```
data(rCRSeq)
width(rCRSeq)
data(mtGenes.rCRS)
getSeq(rCRSeq, mtGenes.rCRS)
```

s4Methods — *List all defined methods for an S4 class (or classes, if you must)*

Description

Mostly for debugging and making architectural choices, e.g. about coverage()

Usage

```
s4Methods(...)
```

Arguments

`...`	name[s] of class[es] (please note, results will be union'ed)

Details

Note: this is borrowed from Hadley, who borrowed it from a BioC workshop!

Value

methods for the class[es], union'ed into a character vector

Examples

```
s4Methods("MVRangesList")
s4Methods("MAlignmentsList")
```

validMetadata — *Ensure that the metadata caches in MAlignmentsLists and MVRanges-Lists are OK*

Description

In order to avoid a lot of lengthy calculations, both MAlignmentsList and MVRangesList objects keep a cache of some relevant statistics and filenames in their metadata slot. If these caches get stale, it can cause problems.

Usage

```
validMetadata(x)
```

Arguments

`x`	an MAlignmentsList or an MVRangesList

Details

This function performs some sanity checks on the caches so that the above problems are unlikely to occur, provided that checkMetadataCache() is called at sensible times. This function is NOT a replacement for validObject().

Value

TRUE or FALSE (if FALSE, attr(res)$mismatches shows why)

Examples

```
library(MTseekerData)
data(RONKSreads)
if(validMetadata(RONKSreads)) message("RONKSreads has valid metadata")
```

Index