Package ‘DNAModAnnot’

December 15, 2020

Type Package

Title Toolbox for DNA Modifications filtering and annotation

Version 0.0.0.9016

Description Use modifications detection output (from kineticsTools and PacBio sequencing data) to filter out potential false positives and analyze the distribution of DNA modifications in the genome assembly provided. Aligned Illumina sequencing data (bam files) can also be used for analysis.

License GPL-3

Depends R (>= 4.0.0)

Imports methods, Biostrings (>= 2.10.0), GenomicRanges (>= 1.38.0), BSgenome (>= 1.28.0), Biobase (> 2.1.0), BiocGenerics (>= 0.34.0), GenomeInfoDb (>= 1.14.0), Gviz (>= 1.29.1), IRanges (>= 2.20.0), Logolas (>= 1.3.1), S4Vectors (>= 0.24.0), data.table (>= 1.13.0), rtracklayer (>= 1.30.0), Rsamtools (>= 2.0.0)

Suggests knitr, GenomicAlignments

Encoding UTF-8

LazyData true

NeedsCompilation no

RoxygenNote 7.1.1

VignetteBuilder knitr

R topics documented:

`AddToModBasePropDistFromFeaturePlot`

AddToModBasePropDistFromFeaturePlot Function (ModAnnot)

Description

Add an additional axis with data on a plot generated by DrawModBasePropDistFromFeature function.

Usage

```
AddToModBasePropDistFromFeaturePlot(
  dPosCountsDistFeatureStart,
  dPosCountsDistFeatureEnd = NULL,
  cSubtitleContent,
  cParamYLabel,
  cParamColor = "cyan3",
  cParamType = "l",
  cParamLwd = 3,
  cParamLty = 3,
  lAddAxisOnLeftSide = TRUE,
  nYLimits = NULL
)
```

Arguments

`dPosCountsDistFeatureStart`
A dataframe containing the data counts by distance to feature positions.

`dPosCountsDistFeatureEnd`
A second dataframe containing the data counts by distance to the second feature positions. If NULL, only 1 position will be plotted (featureStart). Defaults to NULL.

`cSubtitleContent`
A character vector giving the content of the subtitle to add below the title of the plot.

`cParamYLabel` A character vector giving the content of the label on the new Y axis to add on the plot.

`cParamColor` The color of the line and new axis to be added. Defaults to "cyan3".

`cParamType` The type of plot to be added. See "type" argument plot base function. Defaults to "l".

`cParamLwd` The width of the line/points to be added. Defaults to 3.

`cParamLty` If a line is plotted, change the type of the line to be added. Defaults to 3.

`lAddAxisOnLeftSide`
If TRUE, add the new axis on the left side of the plot. If FALSE, add the new axis on the right side of the plot. Defaults to TRUE.

`nYLimits` Numeric vector giving the limits for the new Y axis. Defaults to NULL (will use the minimum and the maximum of both data provided (dPosCountsDistFeatureStart and dPosCountsDistFeatureEnd)).

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
myAnnotations <- rtracklayer::readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

# Retrieve, in a list, dataframes of ModBase counts per Distance values from feature positions
myModDistCountsList <- GetDistFromFeaturePos(
  grangesAnnotations = myAnnotations,
  cSelectFeature = "gene",
  grangesData = myGrangesPacBioGFF,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
myBaseDistCountsList <- GetDistFromFeaturePos(
  grangesAnnotations = myAnnotations,
  cSelectFeature = "gene",
  grangesData = myGposPacBioCSV,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
DrawModBasePropDistFromFeature(
  listModCountsDistDataframe = myModDistCountsList,
  listBaseCountsDistDataframe = myBaseDistCountsList,
  cFeaturePosNames = c("TSS", "TTS"),
  cBaseMotif = "A",
  cModMotif = "6mA"
)

# Loading Bam data
myBamfile <- Rsamtools::BamFile(file = system.file(
  package = "DNAModAnnot", "extdata",
  "PTET_MonoNuc_3-2new.pe.sca171819.sorted.bam"
))
myBam_GRanges <- as(GenomicAlignments::readGAlignments(myBamfile), "GRanges")
myBam_GRanges <- GetGposCenterFromGRanges(grangesData = myBam_GRanges)

# Retrieve dataframes of Read center counts per Distance values in a list
myCountsDist_List_bamfile <- GetDistFromFeaturePos(
  grangesAnnotations = myAnnotations,
  cSelectFeature = "gene",
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = FALSE,
  grangesData = myBam_GRanges,
  cWhichStrandVsFeaturePos = "both",
  nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)

# adding new axis to plot from DrawModBasePropDistFromFeature function
AddToModBasePropDistFromFeaturePlot(
  dPosCountsDistFeatureStart = myCountsDist_List_bamfile[[1]],
  dPosCountsDistFeatureEnd = myCountsDist_List_bamfile[[2]],
  cSubtitleContent = "Along with nucleosome center distance (MonoNuc_3-2newreplicate)",
  cParamYLabel = "Nucleosome center count (MonoNuc_3-2newreplicate)",
  cParamColor = "cyan3",
  lAddAxisOnLeftSide = TRUE
)
```

DrawBarplotBothStrands

DrawBarplotBothStrands Function (GloModAn)

Description

Return a barplot describing some parameter values provided by strand for each contig.

Usage

```
DrawBarplotBothStrands(
  nParamByContigForward,
  nParamByContigReverse,
  cContigNames,
  cGraphName,
  lIsOrderedLargestToSmallest = TRUE
)
```

Arguments

`nParamByContigForward`
: A numeric vector containing the parameter values of the forward strand to be plotted. Must have the same order and same length as cContigNames and nParamByContigReverse.

`nParamByContigReverse`
: A numeric vector containing the parameter values of the reverse strand to be plotted. Must have the same order and same length as nParamByContigForward and cContigNames.

`cContigNames`
: A character vector containing the names of the contigs. Must have the same order and same length as nParamByContigForward and nParamByContigReverse.

`cGraphName`
: The graph name to be displayed on top of the plot.

`lIsOrderedLargestToSmallest`
: If contigs are ordered from largest to smallest contig, add TRUE to display "(from largest to smallest)" below the plot. Defaults to FALSE.

Examples

```
DrawBarplotBothStrands(
  nParamByContigForward = c(100, 86, 75, 56),
  nParamByContigReverse = c(96, 88, 80, 83),
  cContigNames = c("chrI", "chrII", "chrIII", "chrIV"),
  cGraphName = "Mean Coverage per contig",
  lIsOrderedLargestToSmallest = TRUE
)
```

DrawContigCumulLength *DrawContigCumulLength Function (SeQual)*

Description

Return a line-plot describing the cumulative length of contigs (ordered from largest to smallest contig).

Usage

```
DrawContigCumulLength(
  nContigCumsumLength,
  cOrgAssemblyName,
  lGridInBackground = FALSE,
  cLineColor = "red",
  nLineWidth = 2,
  nLineType = 1
)
```

Arguments

`nContigCumsumLength`
A numeric vector containing the cumulative length of contigs (ordered from largest to smallest contig).

`cOrgAssemblyName`
The name of the genome assembly corresponding to these contigs.

`lGridInBackground`
If TRUE, add a grid in the background. Defaults to FALSE.

`cLineColor` The color of the line. Defaults to "red".

`nLineWidth` The width of the line. Defaults to 2.

`nLineType` The type of the line. Defaults to 1.

Examples

```
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

myContig_cumul_len_t <- GetContigCumulLength(dnastringsetGenome = myGenome)

DrawContigCumulLength(
  nContigCumsumLength = myContig_cumul_len_t$cumsum_Mbp_length,
  cOrgAssemblyName = "ptetraurelia_mac_51"
)
```

DrawDistriHistBox *DrawDistriHistBox Function (GloModAn)*

Description

Return a line-plot describing the cumulative length of contigs (ordered from largest to smallest contig).

Usage

```
DrawDistriHistBox(
  nParam,
  cGraphName,
  cParamName,
  lTrimOutliers = FALSE,
  nXLimits = NULL
)
```

Arguments

nParam	A numeric vector containing the parameter values to be plotted.
cGraphName	The graph name to be displayed on top of the plot.
cParamName	The name of the parameter to be displayed below the x-axis.
lTrimOutliers	If TRUE, remove the outliers from the boxplot and trim the histogram to the borders of the boxplot. Defaults to FALSE.
nXLimits	A numeric vector giving the limits of the plot on the x-axis. If NULL, the limits will be set to the minimum and the maximum of the nParam data (or to the inner fences (1.5*Interquartile Range) of the boxplot if lTrimmingOutliers is TRUE). Defaults to NULL.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# Preparing a gposPacBioCSV dataset
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

DrawDistriHistBox(head(myGposPacBioCSV$coverage, 100000),
  cGraphName = "Coverage distribution of some bases sequenced",
  cParamName = "Coverage",
  lTrimOutliers = TRUE
)
```

DrawFdrEstList *DrawFdrEstList Function (FDREst)*

Description

Return a plot describing the false discovery rate estimations by threshold on the parameter provided for each dataframe in the list provided.

Usage

```
DrawFdrEstList(
  listFdrEstByThr,
  cNameParamToTest,
  nFdrPropForFilt = 0.05,
  lAdjustFdr = TRUE
)
```

Arguments

`listFdrEstByThr`

A list composed of x dataframes. In each dataframe:

- fdr: The false discovery rate estimated for this threshold.
- threshold: The threshold on the parameter.
- fdr_cummin: The minimum false discovery rate estimated for this threshold and less stringent thresholds (adjusted false discovery rate).

`cNameParamToTest`

The name of the column containing the parameter to be tested.

`nFdrPropForFilt`

A number indicating the false discovery rate to be used for filtering: this will allow to choose the closest threshold below this number and represent it on the plot. Defaults to 0.05 (so fdr of 5%).

`lAdjustFdr` If TRUE, display fdr_cummin (adjusted false discovery rate) instead of fdr column. Defaults to TRUE.

Examples

```
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
```

```
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a gposPacBioCSV dataset with sequences
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base", "score",
      "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGrangesBaseCSV <- as(myGposPacBioCSV[myGposPacBioCSV$base == "A"], "GRanges")
myGrangesBaseCSVWithSeq <- GetGRangesWindowSeqandParam(
  grangesData = myGrangesBaseCSV,
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1
)

# FDR estimation by motif associated to modifications
myFdr_score_per_motif_list <-
  GetFdrEstListByThresh(
    grangesDataWithSeq = myGrangesBaseCSVWithSeq,
    cNameParamToTest = "score",
    nRoundDigits = 1,
    cModMotifsAsForeground = c("AG", "AT")
  )

DrawFdrEstList(
  listFdrEstByThr = myFdr_score_per_motif_list,
  cNameParamToTest = "score",
  nFdrPropForFilt = 0.05
)
```

DrawModBaseCountsWithinFeature *DrawModBaseCountsWithinFeature Function (ModAnnot)*

Description

Return a plot describing the proportion of the base modified (Mod) and the base letter of the modified base (Base) between windows (of relative size) of feature provided. Example: for Mod="6mA", Base="A"; for Mod="5mC", Base="C".

Usage

```
DrawModBaseCountsWithinFeature(
  grangesAnnotationsWithCountsByWindow,
  cFeatureName,
  cBaseMotif,
  cModMotif
)
```

Arguments

`grangesAnnotationsWithCountsByWindow`
A GRanges object containing feature annotation with the counts:

- Modcount: The number of "Mod" within this window of this feature.
- ModcountSum: Total number of "Mod" within all windows of this feature.
- Modprop: (Modcount / ModcountSum)*100
- Basecount: The number of "Base" within this window of this feature.
- BasecountSum: Total number of "Base" within all windows of this feature.
- Baseprop: (Basecount/BasecountSum)*100

The Genomic features categories must be in a column named "type".

`cFeatureName` Name of the feature (which is cut into windows) to be displayed.

`cBaseMotif` The name of the motif with the base letter of the modified base.

`cModMotif` The name of the motif with the modification in the output.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Retrieve annotations with "Mod" and "Base" counts (and counts per kbp)
myAnn_ModBase_counts_by_window <-
  GetModBaseCountsWithinFeature(
    grangesAnnotations = myAnnotations[myAnnotations$type == "gene", ],
    grangesModPos = myGrangesPacBioGFF,
    gposModTargetBasePos = myGposPacBioCSV,
    nWindowsNb = 20
  )

DrawModBaseCountsWithinFeature(myAnn_ModBase_counts_by_window,
  cFeatureName = "gene",
  cBaseMotif = "A",
  cModMotif = "6mA"
)
```

DrawModBasePropByFeature

DrawModBasePropByFeature Function (ModAnnot)

Description

Return a plot describing, between the features to be displayed, the proportion (or proportion per KiloBase pairs (kbp)) of the base modified (Mod) and the base letter of the modified base (Base). Example: for Mod="6mA", Base="A"; for Mod="5mC", Base="C".

Usage

```
DrawModBasePropByFeature(
  grangesAnnotationsWithCounts,
  cFeaturesToCompare = c("gene", "intergenic"),
  lUseCountsPerkbp = FALSE,
  cBaseMotif,
  cModMotif
)
```

Arguments

`grangesAnnotationsWithCounts`

A GRanges object based on grangesAnnotations with the counts:

- Modcount: The number of "Mod" within this feature.
- Modcount_perkbp: (The number of "Mod" within this feature divided by the size of the feature)*1000.
- Basecount: The number of "Base" within this feature.
- Basecount_perkbp: (The number of "Base" within this feature divided by the size of the feature)*1000.

The Genomic features categories must be in a column named "type".

`cFeaturesToCompare`
Names of the features to be displayed and compared. Defaults to c("gene", "intergenic").

`lUseCountsPerkbp`
If TRUE, use counts per kbp (Modcount_perkbp and Basecount_perkbp) to calculate the proportion between features displayed. If FALSE, use counts (Modcount and Basecount) to calculate the proportion between features displayed. Defaults to FALSE.

`cBaseMotif` The name of the motif with the base letter of the modified base.

`cModMotif` The name of the motif with the modification in the output.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))
myAnnotations <- PredictMissingAnnotation(
  grangesAnnotations = myAnnotations,
  grangesGenome = myGrangesGenome,
  cFeaturesColName = "type",
  cGeneCategories = c("gene"),
  lAddIntronRangesUsingExon = TRUE
)

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Retrieve annotations with "Mod" and "Base" counts (and counts per kbp)
myAnn_ModBase_counts <- GetModBaseCountsByFeature(
  grangesAnnotations = myAnnotations,
  grangesModPos = myGrangesPacBioGFF,
  gposModTargetBasePos = myGposPacBioCSV
)

DrawModBasePropByFeature(
  grangesAnnotationsWithCounts = myAnn_ModBase_counts,
  cFeaturesToCompare = c("gene", "intergenic"),
  lUseCountsPerkbp = TRUE,
  cBaseMotif = "A",
  cModMotif = "6mA"
)
```

DrawModBasePropDistFromFeature

DrawModBasePropDistFromFeature Function (ModAnnot)

Description

Return, in dataframes via a list, the counts (or proportion) of "Mod" (or "Base") positions by distance from feature positions. If the input list contains 4 GRanges, 4 dataframes ("Mod" vs featureStart; "Mod" vs featureEnd; "Base" vs featureStart; "Base" vs featureEnd) will be exported in the output instead of 2 dataframes ("Mod" vs featureStart; "Base" vs featureStart). "Mod": the base modified. "Base": the base letter of the modified base. Example: for Mod="6mA", Base="A"; for Mod="5mC", Base="C".

Usage

```
DrawModBasePropDistFromFeature(
  listModCountsDistDataframe,
  listBaseCountsDistDataframe,
  cFeaturePosNames = c("Start", "End"),
  cBaseMotif,
  cModMotif,
  nDensityBaseMotif = 50
)
```

Arguments

`listModCountsDistDataframe`

A list with 1 or 2 dataframe(s) containing "Mod" counts by distance to feature positions:

- If 1 dataframe is provided in listModCountsDistDataframe, 1 position will be plotted (featureStart).

- If 2 dataframes are provided in listModCountsDistDataframe, 2 positions will be plotted (featureStart; featureEnd).

Must have the same length as the list provided with "listBaseCountsDistDataframe".

`listBaseCountsDistDataframe`
A list with 1 or 2 dataframe(s) containing "Base" counts by distance to feature positions:

- If 1 dataframe is provided in listModCountsDistDataframe, 1 position will be plotted (featureStart).
- If 2 dataframes are provided in listModCountsDistDataframe, 2 positions will be plotted (featureStart; featureEnd).

Must have the same length as the list provided with "listModCountsDistDataframe".

`cFeaturePosNames`
A character vector returning the names of the feature positions provided. Defaults to c("Start","End").

- If 2 dataframes are provided in listModBaseCountsDistDataframe, the name of the feature will be the first element of the vector.
- If 4 dataframes are provided in listModBaseCountsDistDataframe, the names of the feature borders will be the first element then the second element.

`cBaseMotif` The name of the motif with the base letter of the modified base.

`cModMotif` The name of the motif with the modification in the output.

`nDensityBaseMotif`
Numeric vector giving the density of the polygon made with the "Base" counts by distance to feature positions.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Retrieve, in a list, dataframes of ModBase counts per Distance values from feature positions
myModDistCountsList <- GetDistFromFeaturePos(
  grangesAnnotations = myAnnotations,
  cSelectFeature = "gene",
  grangesData = myGrangesPacBioGFF,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
myBaseDistCountsList <- GetDistFromFeaturePos(
  grangesAnnotations = myAnnotations,
  cSelectFeature = "gene",
  grangesData = myGposPacBioCSV,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
DrawModBasePropDistFromFeature(
  listModCountsDistDataframe = myModDistCountsList,
  listBaseCountsDistDataframe = myBaseDistCountsList,
  cFeaturePosNames = c("TSS", "TTS"),
  cBaseMotif = "A",
  cModMotif = "6mA"
)
```

DrawModLogo *DrawModLogo Function (GloModAn)*

Description

Return a plot describing the sequence motif associated to the sequences provided. Sequences that do not have full width and sequences that have some N or some gaps "-" are automatically removed before drawing the sequence plot. If a position is fixed with one base only, that position will not be corrected with the background in order to avoid other letters to appear below this base if using cLogoType="EDLogo".

Usage

```
DrawModLogo(
  dnastringsetSeqAroundMod,
  cLogoType = "EDLogo",
  nGenomicBgACGT = c(0.25, 0.25, 0.25, 0.25),
  cColorsCTAG = c("orange2", "green3", "red2", "blue2")
)
```

Arguments

`dnastringsetSeqAroundMod`
A DNAStringSet object containing the sequence around each DNA modification. All these sequences must have the same size and the modification must have the same position in each sequence (example: at the center).

`cLogoType` Can be "Logo" or "EDLogo". Defaults to "EDLogo".

- Logo: A classic logo showing sequence enrichment around the modification.
- EDLogo: A logo showing sequence enrichment and depletion around the modification.

`nGenomicBgACGT` A numeric vector giving the background to be corrected with the genomic composition in Adenine (A) then Cytosine (C) then Guanine (G) then Thymine (T). Defaults to c(0.25, 0.25, 0.25, 0.25).

`cColorsCTAG` A character vector giving the color for the Cytosine (C) then Thymine (T) then Adenine (A) then Guanine (G) letters on the plot. Defaults to c("orange2","green3","red2","blue2").

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a grangesPacBioGFF datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )

# Retrieve GRanges with sequence
myPositions_Mod_Granges_wSeq <- GetGRangesWindowSeqandParam(
  grangesData = myGrangesPacBioGFF,
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  nUpstreamBpToAdd = 5,
  nDownstreamBpToAdd = 5
)

DrawModLogo(
  dnastringsetSeqAroundMod = as(myPositions_Mod_Granges_wSeq$sequence, "DNAStringSet"),
  cLogoType = "EDLogo",
  nGenomicBgACGT = c(0.35, 0.15, 0.15, 0.35)
)
```

DrawParamPerModBaseCategories

DrawParamPerModBaseCategories Function (ModAnnot)

Description

Return boxplots describing, for each feature provided, the distribution of a parameter provided by categories of counts (or counts per KiloBase pairs (kbp)) of the base modified (Mod) and the base letter of the modified base (Base). These categories should have a comparable amount of Mod (or Base) between them. Example: for Mod="6mA", Base="A"; for Mod="5mC", Base="C".

Usage

```
DrawParamPerModBaseCategories(
  grangesAnnotationsWithCounts,
  cParamColname,
  cParamFullName = cParamColname,
  cParamYLabel = cParamColname,
  cSelectFeature = NULL,
  lUseCountsPerkbp = TRUE,
  lKeepOutliers = FALSE,
  lUseSameYAxis = FALSE,
  cBaseMotif,
  cModMotif,
  lBoxPropToCount = TRUE
)
```

Arguments

`grangesAnnotationsWithCounts`
A GRanges object based on grangesAnnotations with the counts:

- Modcount: The number of "Mod" within this feature.
- Modcount_perkbp: (The number of "Mod" within this feature divided by the size of the feature)*1000.
- Basecount: The number of "Base" within this feature.
- Basecount_perkbp: (The number of "Base" within this feature divided by the size of the feature)*1000.

The Genomic features categories must be in a column named "type".

`cParamColname` Name of the column, in the grangesAnnotationsWithCounts provided, containing the parameter to be compared with Mod|Base categories.

`cParamFullName` Name of the parameter to be displayed in the title of the plot. Defaults to cParamColname.

cParamYLabel Name of the parameter to be displayed on the Y-axis of the plot. Defaults to cParamColname.

cSelectFeature The name of the feature from the annotation to be analysed along counts and parameter provided. Defaults to NULL (No subsetting: counts and parameter provided from all features in the annotation provided will be used).

lUseCountsPerkbp
If TRUE, use counts per kbp (Modcount_perkbp and Basecount_perkbp) to calculate the Mod|Base categories to be displayed. If FALSE, use counts (Modcount and Basecount) to calculate the Mod|Base categories to be displayed. Defaults to TRUE.

lKeepOutliers If FALSE, remove outliers before plotting. Defaults to FALSE.

lUseSameYAxis If TRUE, the 2 plots will use the same range for the y-axis. If FALSE, y-axis of the 2 plots will be independant. Defaults to FALSE.

cBaseMotif The name of the motif with the base letter of the modified base.

cModMotif The name of the motif with the modification in the output.

lBoxPropToCount
If TRUE, the width of each boxplot depends on the number of Mod (or Base) in the categories. If FALSE, all boxplots will have the same size. Defaults to TRUE.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Retrieve annotations with "Mod" and "Base" counts (and counts per kbp)
myAnn_ModBase_counts <- GetModBaseCountsByFeature(
  grangesAnnotations = myAnnotations,
  grangesModPos = myGrangesPacBioGFF,
  gposModTargetBasePos = myGposPacBioCSV
)

# Add Parameter by feature to annotation file
myAnn_ModBase_counts$ParamToPlot <- width(myAnn_ModBase_counts)

DrawParamPerModBaseCategories(myAnn_ModBase_counts,
  cParamColname = "ParamToPlot",
  cParamFullName = "Gene Width",
  cParamYLabel = "Gene Width (bp)",
  cSelectFeature = c("gene"),
  lUseCountsPerkbp = TRUE,
  lKeepOutliers = FALSE,
  lUseSameYAxis = FALSE,
  cBaseMotif = "A",
  cModMotif = "6mA",
  lBoxPropToCount = FALSE
)
```

ExportFilesForGViz *ExportFilesForGViz Function (ModAnnot)*

Description

Export data as files that can be directly used with Gviz Package (making tracks + displaying). Except for gff3 format, all available format allows streaming while making and displaying tracks with Gviz package. Multiple objects can be exported at the same time. All arguments (except "dnastringsetGenome" argument) must have the same length.

Usage

```
ExportFilesForGViz(
  dnastringsetGenome,
  cFileNames,
  listObjects,
  cFileFormats,
  cBigwigParameters = rep(NA, length(cFileNames)),
  lBigwigParametersByStrand = rep(NA, length(cFileNames)),
  cBamXaParameters = rep(NA, length(cFileNames))
)
```

Arguments

dnastringsetGenome
: A DNAStringSet object containing the sequence for each contig.

cFileNames
: A character vector giving the names of the files to be exported.

listObjects
: A list of objects to be exported. Required format of the objects is described in "cFileFormats" argument documentation.

cFileFormats
: A character vector giving the format for each file to be exported. The following exportation formats are supported:

- Fasta using DNAStrinSet object : for sequenceTracks (streaming)
- Bam using GRanges-like object : for alignmentTracks (streaming) (only the positions of the ranges will be retrieved) or for dataTracks (streaming) (only coverage of provided ranges will be displayed) or for annotationTracks (streaming) (positions of the ranges will be retrieved with the names of each range (e.g. gene name))
- bw (bigwig) using GRanges-like object : for dataTracks (streaming) (only the chosen numeric parameter will be retrieved for each range provided)
- gff3 using GRanges-like object : for annotationTracks or geneRegionTracks (not in streaming: more memory required for displaying)

cBigwigParameters
: A character vector describing the name of the parameter to be stored in bigwig files. Must correspond to the name of a column in the provided Granges object. Use NA as value in the vector if the associated file is not to be exported as a bigwig. Defaults to "rep(NA, length(cFileNames)))"

lBigwigParametersByStrand
: A logical vector: if TRUE, the bigwig parameter will be negative for each range that is located on the reverse strand in the GRanges object provided. Use NA as value in the vector if the associated file is not to be exported as a bigwig. Defaults to "rep(NA, length(cFileNames)))"

cBamXaParameters
: A character vector describing a parameter to be stored as a "Xa" optional field in the exported bam file. Use NA as value in the vector if the associated file is not to be exported as a bam. If the exported file is a bam and if the value is NA or NULL, the bam will be exported without the "Xa" optional field. Defaults to "rep(NA, length(cFileNames)))"

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# Preparing a grangesPacBioCSV dataset
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

## NOT RUN!
## Export files for Gviz
# ExportFilesForGViz(dnastringsetGenome = myGenome,
#                   cFileNames = c("ipdRatio_for_each_A.bw",
#                                  "score_for_all_bases.bw",
#                                  "newFastaOnlyscaffold51_17.fa"),
#                   listGRangesObjects = c(myGposPacBioCSV[myGposPacBioCSV$base == "A"],
#                                          myGposPacBioCSV,
#                                          myGenome["scaffold51_17"]),
#                   cFileFormats = c("bw", "bw", "fa"),
#                   cBigwigParameters = c("ipdRatio", "score", NA),
#                   lBigwigParametersByStrand = c(TRUE, TRUE, NA),
#                   cBamXaParameters = c(NA, NA, NA))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

## NOT RUN!
## Export files for Gviz
# ExportFilesForGViz(dnastringsetGenome = myGenome,
#                   cFileNames = c("genes.bam"),
#                 listGRangesObjects = list(myAnnotations[myAnnotations$type == "gene"]),
#                   cFileFormats = c("bam"),
#                   cBamXaParameters = c("Name") )
```

ExtractListModPosByModMotif

ExtractListModPosByModMotif Function (GloModAn)

Description

Return the GRanges object provided with the sequence associated to each position (and can also retrieve the sequence around each position).

Usage

```
ExtractListModPosByModMotif(
  grangesModPos,
  grangesGenome,
  dnastringsetGenome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1,
  nModMotifMinProp,
  nModPositionInMotif = 1 + nUpstreamBpToAdd,
  cBaseLetterForMod,
  cModNameInOutput
)
```

Arguments

`grangesModPos` A GRanges object containing Modifications Positions data to be extracted with the sequence.

`grangesGenome` A GRanges object containing the width of each contig.

`dnastringsetGenome`
A DNAStringSet object containing the sequence for each contig.

`nUpstreamBpToAdd`
Number of base pairs to add upstream of the range from the GRanges object provided to obtain some sequence upstream of range. If some new ranges do not fit in the ranges of the contigs (provided with grangesGenome), those new ranges will be removed. New windows with gaps are also removed. Defaults to 0.

`nDownstreamBpToAdd`
Number of base pairs to add downstream of the range from the GRanges object provided to obtain some sequence downstream of range. If some new ranges do not fit in the ranges of the contigs (provided with grangesGenome), those new ranges will be removed. New windows with gaps are also removed. Defaults to 0.

`nModMotifMinProp`
A number indicating the false discovery rate to be used for filtering: this will allow to choose the closest threshold below this number. Defaults to 0.05 (so fdr of 5%).

`nModPositionInMotif`
The position of the modification in the window after resizing with nUpstreamBpToAdd and nDownstreamBpToAdd. If GRanges are 1-bp positions, then 1+nUpstreamBpToAdd will return the right position of the modification. Defaults to 1+nUpstreamBpToAdd.

`cBaseLetterForMod`
The name of the base letter of the modified base.

`cModNameInOutput`
Name for the modification in the output.

Value

A list of 4 objects:

motifs_to_analyse A character vector containing the sequence of motifs associated to the modification.

mod_motif A character vector containing the sequence of motifs associated to the modification with the modification represented inside those motifs.

motif_pct A table containing the percentage of modifications in each motif tested.

GRangesbyMotif A list of GRanges objects with the sequence: one GRanges object by motif associated to the modification.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a grangesPacBioGFF dataset
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )

# Retrieve GRanges with sequence
myMotif_pct_and_GRangesList <- ExtractListModPosByModMotif(
  grangesModPos = myGrangesPacBioGFF,
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1,
  nModMotifMinProp = 0.05,
  cBaseLetterForMod = "A",
  cModNameInOutput = "6mA"
)

myMotif_pct_and_GRangesList$motifs_to_analyse
myMotif_pct_and_GRangesList$mod_motif
myMotif_pct_and_GRangesList$motif_pct
myMotif_pct_and_GRangesList$GRangesbyMotif
```

FiltContig *FiltContig Function (Filter)*

Description

Filter out data from contigs that do not reach criterias of selection.

Usage

```
FiltContig(
  gposModBasePos,
  grangesModPos,
  cContigToBeRemoved = NULL,
  dnastringsetGenome,
  nContigMinSize = -1,
  listPctSeqByContig,
  nContigMinPctOfSeq = 95,
  listMeanCovByContig,
  nContigMinCoverage = 20
)
```

Arguments

gposModBasePos An UnStitched GPos object containing PacBio CSV data to be filtered.

grangesModPos A GRanges object containing PacBio GFF data to be filtered.

cContigToBeRemoved
Names of contigs for which the data will be removed. Defaults to NULL.

dnastringsetGenome
A DNAStringSet object containing the sequence for each contig.

nContigMinSize Minimum size for contigs to keep. Contigs with a size below this value will be removed. Defaults to -1 (= no filter).

listPctSeqByContig
List containing, for each strand, the percentage of sequencing for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "seqPct" column returning percentage of sequencing.

nContigMinPctOfSeq
Minimum percentage of sequencing for contigs to keep. Contigs with a percentage below this value will be removed. Defaults to 95.

listMeanCovByContig
List containing, for each strand, the mean of coverage for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "mean_coverage" column returning mean of coverage.

nContigMinCoverage
Minimum mean coverage for contigs to keep. Contigs with a mean coverage below this value will be removed. Defaults to 20.

Value

A list with filtered gposModBasePos and filtered grangesModPos.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a gposPacBioCSV and a grangesPacBioGFF datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

# Preparing ParamByStrand Lists
myPct_seq_csv <- GetSeqPctByContig(myGposPacBioCSV, grangesGenome = myGrangesGenome)
myMean_cov_list <- GetMeanParamByContig(
  grangesData = myGposPacBioCSV,
  dnastringsetGenome = myGenome,
  cParamName = "coverage"
)

# Filtering
myFiltered_data <- FiltContig(myGposPacBioCSV, myGrangesPacBioGFF,
  cContigToBeRemoved = NULL,
  dnastringsetGenome = myGenome, nContigMinSize = 1000,
  listPctSeqByContig = myPct_seq_csv, nContigMinPctOfSeq = 95,
  listMeanCovByContig = myMean_cov_list, nContigMinCoverage = 20
)
myFiltered_data$csv
myFiltered_data$gff
```

FiltDeepSignal *FiltModDeepSignal Function (Filter)*

Description

Filter out data from contigs or Modifications that do not reach criterias of selection. Can also be used to obtain a gposDeepSignalMod object by simply filtering target sites which have a fraction above 0.

Usage

```
FiltDeepSignal(
  gposDeepSignalModBase = NULL,
  gposDeepSignalMod = NULL,
  cContigToBeRemoved = NULL,
  dnastringsetGenome,
  nContigMinSize = -1,
  listPctSeqByContig,
  nContigMinPctOfSeq = -1,
  listMeanCovByContig,
  nContigMinCoverage = -1,
  cParamNameForFilter = NULL,
  lFiltParam = FALSE,
  nFiltParamLoBoundaries = NULL,
  nFiltParamUpBoundaries = NULL,
  cFiltParamBoundariesToInclude = NULL,
  listMeanParamByContig = NULL,
  nContigFiltParamLoBound = NULL,
  nContigFiltParamUpBound = NULL,
  nModMinCoverage = NULL
)
```

Arguments

`gposDeepSignalModBase`
: An UnStitched GPos object containing DeepSignal modification target sites data to be filtered. Defaults to NULL.

`gposDeepSignalMod`
: An UnStitched GPos object containing DeepSignal modified sites data to be filtered. Defaults to NULL.

`cContigToBeRemoved`
: Names of contigs for which the data will be removed. gposPacBioCSV must be provided if using this argument. Defaults to NULL.

`dnastringsetGenome`
: A DNAStringSet object containing the sequence for each contig.

`nContigMinSize`
: Minimum size for contigs to keep. Contigs with a size below this value will be removed. gposPacBioCSV must be provided if using this argument. Defaults to -1 (= no filter).

`listPctSeqByContig`
: List containing, for each strand, the percentage of sequencing for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "seqPct" column returning percentage of sequencing. gposPacBioCSV must be provided if using this argument.

`nContigMinPctOfSeq`
: Minimum percentage of sequencing for contigs to keep. Contigs with a percentage below this value will be removed. gposPacBioCSV must be provided if using this argument. Defaults to 95.

`listMeanCovByContig`
: List containing, for each strand, the mean of coverage for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "mean_coverage" column returning mean of coverage. gposPacBioCSV must be provided if using this argument.

`nContigMinCoverage`
: Minimum mean coverage for contigs to keep. Contigs with a mean coverage below this value will be removed. gposPacBioCSV must be provided if using this argument. Defaults to 20.

`cParamNameForFilter`
: A character vector giving the name of the parameter to be filtered. Must correspond to the name of one column in the object provided with grangesModPos.

`lFiltParam` If TRUE, remove modifications which have values of the given parameter that are not included in the intervals provided with "nFiltParamLoBoundaries" and "nFiltParamUpBoundaries". Defaults to FALSE.

`nFiltParamLoBoundaries` A numeric vector returning the lower boundaries of intervals. Must have the same length as "nFiltParamUpBoundaries". Defaults to NULL.

`nFiltParamUpBoundaries` A numeric vector returning the upper boundaries of intervals. Must have the same length as "nFiltParamLoBoundaries". Defaults to NULL.

`cFiltParamBoundariesToInclude` A character vector describing which interval boundaries must be included in the intervals provided. Can be "upperOnly" (only upper boundaries), "lowerOnly" (only lower boundaries), "both" (both upper and lower boundaries) or "none" (do not include upper and lower boundaries). If NULL, both upper and lower boundaries will be included (= "both"). Defaults to NULL. cFiltParamBoundariesToInclude = NULL #can be "upperOnly","lowerOnly","both", "none' (NULL = "both" for all)

`listMeanParamByContig` List containing, for each strand, the mean of a given parameter for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "mean_"[parameter name] column returning the mean of the given parameter. If not NULL, remove contigs that are too far away from the mean of the Parameter of all contigs (which are not included in the interval centered on the mean) in the list provided. Defaults to NULL.

`nContigFiltParamLoBound` A numeric value to be removed from the mean of the given parameter of all contigs (calculates the lower bound of the interval centered on the mean). Defaults to NULL.

`nContigFiltParamUpBound` A numeric value to be added to the mean of the given parameter of all contigs (calculates the upper bound of the interval centered on the mean). Defaults to NULL.

`nModMinCoverage` Minimum coverage for all Modifications to be kept. Modifications with a coverage below this value will be removed. Defaults to NULL (no filter).

Examples

```
# Loading Nanopore data
myDeepSignalModPath <- system.file(
  package = "DNAModAnnot", "extdata",
  "FAB39088-288418386-Chr1.CpG.call_mods.frequency.tsv"
)
mygposDeepSignalModBase <- ImportDeepSignalModFrequency(
  cDeepSignalModPath = myDeepSignalModPath,
  lSortGPos = TRUE,
  cContigToBeAnalyzed = "all"
)
mygposDeepSignalModBase

# Filtering
mygposDeepSignalMod <- FiltDeepSignal(
  gposDeepSignalModBase = mygposDeepSignalModBase,
  cParamNameForFilter = "frac",
  lFiltParam = TRUE,
  nFiltParamLoBoundaries = 0,
  nFiltParamUpBoundaries = 1,
  cFiltParamBoundariesToInclude = "upperOnly"
)$Mod
mygposDeepSignalMod
```

FiltFdrBased *FiltFdrBased Function (Filter)*

Description

Filter out modifications which have a parameter (tested with FDR estimations) that do not reach criterias of selection.

Usage

```
FiltFdrBased(
  grangesModPosWithSeq,
  listFdrEstByThrIpdRatio = NULL,
  listFdrEstByThrScore = NULL
)
```

Arguments

`grangesModPosWithSeq`
: A List of GRanges object, with the sequence for each motif associated to the modification, and containing PacBio GFF data to be filtered.

`listFdrEstByThrIpdRatio`
: A list of thresholds on ipdRatio for each motif associated to the modification. Defaults to NULL.

`listFdrEstByThrScore`
: A list of thresholds on score for each motif associated to the modification. Defaults to NULL.

Value

A filtered grangesModPosWithSeq.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a grangesPacBioGFF dataset
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = c("scaffold51_17", "scaffold51_18", "scaffold51_19")
  )

myMotif_pct_and_GRangesList <- ExtractListModPosByModMotif(
  grangesModPos = myGrangesPacBioGFF,
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1,
  nModMotifMinProp = 0.05,
  cBaseLetterForMod = "A",
  cModNameInOutput = "6mA"
)
# Filtering
myPosMod_GRangesbyMotif_filt <-
  FiltFdrBased(
    grangesModPosWithSeq = myMotif_pct_and_GRangesList$GRangesbyMotif,
    listFdrEstByThrIpdRatio = list(2.1, 2.1, 2.1),
    listFdrEstByThrScore = list(42, 42, 42)
  )
myPosMod_GRangesbyMotif_filt
```

FiltPacBio *FiltPacBio Function (Filter)*

Description

Filter out data from contigs or Modifications that do not reach criterias of selection.

Usage

```
FiltPacBio(
  grangesPacBioGFF,
  gposPacBioCSV = NULL,
  cContigToBeRemoved = NULL,
  dnastringsetGenome,
  nContigMinSize = -1,
  listPctSeqByContig,
  nContigMinPctOfSeq = -1,
  listMeanCovByContig,
  nContigMinCoverage = -1,
  cParamNameForFilter = NULL,
  listMeanParamByContig = NULL,
  nContigFiltParamLoBound = NULL,
  nContigFiltParamUpBound = NULL,
  lFiltParam = FALSE,
  nFiltParamLoBoundaries = NULL,
  nFiltParamUpBoundaries = NULL,
  cFiltParamBoundariesToInclude = NULL,
  nModMinIpdRatio = NULL,
  nModMinScore = NULL,
  nModMinCoverage = NULL,
  listFdrEstByThrIpdRatio = NULL,
  listFdrEstByThrScore = NULL
)
```

Arguments

`grangesPacBioGFF`
A GRanges object containing PacBio GFF data to be filtered OR A List of GRanges object, with the sequence for each motif associated to the modification, and containing PacBio GFF data to be filtered.

`gposPacBioCSV`
An UnStitched GPos object containing PacBio CSV data to be filtered. Defaulst to NULL.

`cContigToBeRemoved`
Names of contigs for which the data will be removed. gposPacBioCSV must be provided if using this argument. Defaults to NULL.

`dnastringsetGenome`
A DNAStringSet object containing the sequence for each contig.

`nContigMinSize`
Minimum size for contigs to keep. Contigs with a size below this value will be removed. gposPacBioCSV must be provided if using this argument. Defaults to -1 (= no filter).

`listPctSeqByContig`
List containing, for each strand, the percentage of sequencing for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "seqPct" column returning percentage of sequencing. gposPacBioCSV must be provided if using this argument.

`nContigMinPctOfSeq`
Minimum percentage of sequencing for contigs to keep. Contigs with a percentage below this value will be removed. gposPacBioCSV must be provided if using this argument. Defaults to 95.

`listMeanCovByContig`
List containing, for each strand, the mean of coverage for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "mean_coverage" column returning mean of coverage. gposPacBioCSV must be provided if using this argument.

`nContigMinCoverage`
Minimum mean coverage for contigs to keep. Contigs with a mean coverage below this value will be removed. gposPacBioCSV must be provided if using this argument. Defaults to 20.

`cParamNameForFilter`
A character vector giving the name of the parameter to be filtered. Must correspond to the name of one column in the object provided with grangesModPos.

`listMeanParamByContig`
List containing, for each strand, the mean of a given parameter for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and

r_strand. In each dataframe, "refName" column returning names of contigs and "mean_"[parameter name] column returning the mean of the given parameter. If not NULL, remove contigs that are too far away from the mean of the Parameter of all contigs (which are not included in the interval centered on the mean) in the list provided. Defaults to NULL.

`nContigFiltParamLoBound`
: A numeric value to be removed from the mean of the given parameter of all contigs (calculates the lower bound of the interval centered on the mean). Defaults to NULL.

`nContigFiltParamUpBound`
: A numeric value to be added to the mean of the given parameter of all contigs (calculates the upper bound of the interval centered on the mean). Defaults to NULL.

`lFiltParam`
: If TRUE, remove modifications which have values of the given parameter that are not included in the intervals provided with "nFiltParamLoBoundaries" and "nFiltParamUpBoundaries". Defaults to FALSE.

`nFiltParamLoBoundaries`
: A numeric vector returning the lower boundaries of intervals. Must have the same length as "nFiltParamUpBoundaries". Defaults to NULL.

`nFiltParamUpBoundaries`
: A numeric vector returning the upper boundaries of intervals. Must have the same length as "nFiltParamLoBoundaries". Defaults to NULL.

`cFiltParamBoundariesToInclude`
: A character vector describing which interval boundaries must be included in the intervals provided. Can be "upperOnly" (only upper boundaries), "lowerOnly" (only lower boundaries), "both" (both upper and lower boundaries) or "none" (do not include upper and lower boundaries). If NULL, both upper and lower boundaries will be included (= "both"). Defaults to NULL. cFiltParamBoundariesToInclude = NULL #can be "upperOnly","lowerOnly","both", "none' (NULL = "both" for all)

`nModMinIpdRatio`
: Minimum ipdRatio for all Modifications to be kept. Modifications with an ipdRatio below this value will be removed. Defaults to NULL (no filter).

`nModMinScore`
: Minimum score for all Modifications to be kept. Modifications with a score below this value will be removed. Defaults to NULL (no filter).

`nModMinCoverage`
: Minimum coverage for all Modifications to be kept. Modifications with a coverage below this value will be removed. Defaults to NULL (no filter).

`listFdrEstByThrIpdRatio`
: A list of thresholds on ipdRatio for each motif associated to the modification.

`listFdrEstByThrScore`
: A list of thresholds on score for each motif associated to the modification.

Value

A list with filtered gposPacBioCSV and filtered gposPacBioGFF.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a gposPacBioCSV and a grangesPacBioGFF datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

# Preparing ParamByStrand Lists
myPct_seq_csv <- GetSeqPctByContig(myGposPacBioCSV, grangesGenome = myGrangesGenome)
myMean_cov_list <- GetMeanParamByContig(
  grangesData = myGposPacBioCSV,
  dnastringsetGenome = myGenome,
  cParamName = "coverage"
)

# Filtering
myFiltered_data <- FiltPacBio(
  grangesPacBioGFF = myGrangesPacBioGFF,
  gposPacBioCSV = myGposPacBioCSV, cContigToBeRemoved = NULL,
  dnastringsetGenome = myGenome, nContigMinSize = 1000,
  listPctSeqByContig = myPct_seq_csv, nContigMinPctOfSeq = 95,
  listMeanCovByContig = myMean_cov_list, nContigMinCoverage = 20
)
myFiltered_data$csv
myFiltered_data$gff
```

FiltParam *FiltParam Function (Filter)*

Description

Filter out modifications which have a chosen parameter that do not reach criterias of selection.

Usage

```
FiltParam(
  grangesModPos,
  cParamNameForFilter,
  lFiltParam = FALSE,
  nFiltParamLoBoundaries = NULL,
  nFiltParamUpBoundaries = NULL,
  cFiltParamBoundariesToInclude = NULL,
  listMeanParamByContig = NULL,
  nContigFiltParamLoBound = NULL,
  nContigFiltParamUpBound = NULL
)
```

Arguments

`grangesModPos` A GRanges object containing Modifications positions data to be filtered.

`cParamNameForFilter`
A character vector giving the name of the parameter to be filtered. Must correspond to the name of one column in the object provided with grangesModPos.

`lFiltParam` If TRUE, remove modifications which have values of the given parameter that are not included in the intervals provided with "nFiltParamLoBoundaries" and "nFiltParamUpBoundaries". Defaults to FALSE.

`nFiltParamLoBoundaries`
A numeric vector returning the lower boundaries of intervals. Must have the same length as "nFiltParamUpBoundaries". Defaults to NULL.

`nFiltParamUpBoundaries`
A numeric vector returning the upper boundaries of intervals. Must have the same length as "nFiltParamLoBoundaries". Defaults to NULL.

`cFiltParamBoundariesToInclude`
A character vector describing which interval boundaries must be included in the intervals provided. Can be "upperOnly" (only upper boundaries), "lowerOnly" (only lower boundaries), "both" (both upper and lower boundaries) or "none" (do not include upper and lower boundaries). If NULL, both upper and lower boundaries will be included (= "both"). Defaults to NULL. cFiltParamBoundariesToInclude = NULL #can be "upperOnly","lowerOnly","both", 'none' (NULL = "both" for all)

`listMeanParamByContig`
List containing, for each strand, the mean of a given parameter for each contig. This list must be composed of 2 dataframes (one by strand) called f_strand and r_strand. In each dataframe, "refName" column returning names of contigs and "mean_"[parameter name] column returning the mean of the given parameter. If not NULL, remove contigs that are too far away from the mean of the Parameter of all contigs (which are not included in the interval centered on the mean) in the list provided. Defaults to NULL.

`nContigFiltParamLoBound`
A numeric value to be removed from the mean of the given parameter of all contigs (calculates the lower bound of the interval centered on the mean). Defaults to NULL.

`nContigFiltParamUpBound`
A numeric value to be added to the mean of the given parameter of all contigs (calculates the upper bound of the interval centered on the mean). Defaults to NULL.

Value

A filtered grangesModPos object.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# Preparing a grangesPacBioGFF dataset
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = c("scaffold51_17", "scaffold51_18", "scaffold51_19")
  )

# Preparing ParamByStrand List
myMean_fra_list <- GetMeanParamByContig(
  grangesData = myGrangesPacBioGFF,
  dnastringsetGenome = myGenome,
  cParamName = "frac"
)

# Filtering Out Modif with Frac 20% above/below the mean Frac
myGRangesPacBioGFF_filt <- FiltParam(
  grangesModPos = myGrangesPacBioGFF,
  cParamNameForFilter = "frac",
  listMeanParamByContig = myMean_fra_list,
  nContigFiltParamLoBound = 0.2,
  nContigFiltParamUpBound = 0.2
)
myGRangesPacBioGFF_filt

# Filtering Out Modif with Frac below 5%
myGRangesPacBioGFF_filt <- FiltParam(
  grangesModPos = myGrangesPacBioGFF,
  cParamNameForFilter = "frac",
  listMeanParamByContig = myMean_fra_list,
  lFiltParam = TRUE,
  nFiltParamLoBoundaries = 0.05,
  nFiltParamUpBoundaries = 1.00
)
myGRangesPacBioGFF_filt

# Keeping Modif with Frac between 40% and 60%;
# or between 90% and 100%  (adapted to genome of diploid organisms)
myGRangesPacBioGFF_filt <- FiltParam(
  grangesModPos = myGrangesPacBioGFF,
  cParamNameForFilter = "frac",
  lFiltParam = TRUE,
  nFiltParamLoBoundaries = c(0.40, 0.90),
  nFiltParamUpBoundaries = c(0.60, 1.00)
)
```

GetAssemblyReport — *GetAssemblyReport Function (SeQual) importFrom base data.frame max sum return importFrom Biostrings alphabetFrequency letterFrequency*

Description

Return a report with global characteristics of genome assembly.

Usage

```
GetAssemblyReport(dnastringsetGenome, cOrgAssemblyName)
```

Arguments

`dnastringsetGenome`
A DNAStringSet object containing the sequence for each contig.

`cOrgAssemblyName`
The name of the genome assembly provided.

Examples

```
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

myReport <- GetAssemblyReport(
  dnastringsetGenome = myGenome,
  cOrgAssemblyName = "ptetraurelia_mac_51"
)

myReport
```

GetContigCumulLength — *GetContigCumulLength Function (SeQual)*

Description

Return a dataframe with the length (and cumulative length) of contigs (ordered from largest to smallest contig) provided in genome assembly.

Usage

```
GetContigCumulLength(dnastringsetGenome)
```

Arguments

`dnastringsetGenome`
A DNAStringSet object containing the sequence for each contig.

Value

A dataframe:

- contig_names: The names of each contig.
- Mbp_length: The size of each contig (in Megabase pairs (Mbp)).
- cumsum_Mbp_length: The cumulative size (from largest to smallest contig) for each contig (in Megabase pairs (Mbp)).

Examples

```
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

myContig_cumul_len_t <- GetContigCumulLength(dnastringsetGenome = myGenome)

myContig_cumul_len_t
```

GetDistFromFeaturePos *GetDistFromFeaturePos Function (ModAnnot)*

Description

Return, in GRanges objects via a GRangesList, the distance between the "Positions" provided with "grangesData" argument (e.g. position of some target sites) and the feature positions provided with "grangesAnnotations". If the ranges in "grangesAnnotations" are not 1-bp positions, the positions of the boundaries will be used as the feature positions: in this case, 2 GRanges ("Position" vs featureStart; "Position" vs featureEnd) will be exported in the output instead of 1 Granges ("Position" vs featureStart). This function can also directly compute counts or proportion of the provided "Positions" at each nucleotide position around provided features (see "lGetGRangesInsteadOfListCounts" argument).

Usage

```
GetDistFromFeaturePos(
  grangesAnnotations,
  cSelectFeature = NULL,
  grangesData,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  nWindowSizeAroundFeaturePos,
  cWhichStrandVsFeaturePos = "both",
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("Start", "End")
)
```

Arguments

`grangesAnnotations`
: A GRanges object containing the annotation for the genome assembly corresponding to the grangesModPos and gposModTargetBasePos provided. The Genomic features categories must be in a column named "type".

`cSelectFeature`
: The name of the feature from the annotation to be analysed. Defaults to NULL (all ranges from grangesAnnotations will be kept).

`grangesData`
: A GRanges object containing "Positions" to be counted around features positions.

`lGetGRangesInsteadOfListCounts`
: If FALSE, return, in dataframes via a list, the counts (or proportion if "lGetPropInsteadOfCounts" is TRUE) of "Positions" at each base position around feature positions. Defaults to FALSE.

`lGetPropInsteadOfCounts`
: If "lGetGRangesInsteadOfListCounts" and "lGetPropInsteadOfCounts" are TRUE, return the proportion of "Positions" near feature positions: counts / sum of counts. If the ranges in "grangesAnnotations" are not 1-bp positions, the proportion of "Positions" is calculated near both feature positions (Start and End): counts / (sum of counts near feature1 + sum of counts near feature2). Defaults to TRUE.

`nWindowSizeAroundFeaturePos`
: Size, in base pairs, of the viewing window around the feature positions.

`cWhichStrandVsFeaturePos`
: A character value describing if distance comparison must be made between "Mod" (or "Base") and the feature positions...
  - "same": ...if these positions are on the same strand only.
  - "opposite":...if these positions are on opposite strands only.
  - "both": ...for all of these positions: same and opposite strands.

`lAddCorrectedDistFrom5pTo3p`
: If TRUE, the distance will be corrected to reflect 5’ to 3’ direction and will be stored in a new column (dist_5to3). Defaults to TRUE.

`cFeaturePosNames`
: A character vector returning the names of the feature positions provided. Defaults to c("Start","End").
  - If the width of ranges is equal to 1, the name of the feature will be the first element of the vector.
  - If the width of ranges is above 1, the names of the feature borders will be the first element then the second element.

Value

A GRangesList with 1 or 2 GRanges objects containing ranges of "Positions" with their distance to feature positions:

- If the width of annotation ranges is equal to 1, 1 GRanges is provided ("Position" vs featureStart).
- If the width of annotation ranges is above 1, 2 GRanges are provided ("Position" vs featureStart; "Position" vs featureEnd).

If "lGetGRangesInsteadOfListCounts" is FALSE, retrieve instead a list with 1 or 2 dataframe(s) containing "Positions" counts by distance to feature positions:

- If the width of annotation ranges is equal to 1, 1 dataframe is provided ("Position" vs featureStart).
- If the width of annotation ranges is above 1, 2 dataframes are provided ("Position" vs featureStart; "Position" vs featureEnd).

If a "Position" is within "nWindowSizeAroundFeaturePos" base pairs of x different feature positions: this "Position" will then reported x times with the distance to each feature position around this "Position".

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Retrieve, in a list, dataframes of Mod counts per Distance values from feature positions
myModDistCountsList <- GetDistFromFeaturePos(
  grangesAnnotations = myAnnotations,
  cSelectFeature = "gene",
  grangesData = myGrangesPacBioGFF,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
myModDistCountsList
```

GetFdrBasedThreshLimit

GetFdrBasedThreshLimit Function (FDREst)

Description

Return a plot describing the false discovery rate (fdr) estimations by threshold on the parameter provided for each dataframe in the list provided.

Usage

```
GetFdrBasedThreshLimit(
  listFdrEstByThr,
  nFdrPropForFilt = 0.05,
  lUseBestThrIfNoFdrThr = TRUE
)
```

Arguments

`listFdrEstByThr`
A list composed of x dataframes. In each dataframe:

- fdr: The false discovery rate estimated for this threshold.
- threshold: The threshold on the parameter.
- fdr_cummin: The minimum false discovery rate estimated for this threshold and less stringent thresholds (adjusted false discovery rate).

`nFdrPropForFilt`
A number indicating the false discovery rate to be used for filtering: this will allow to choose the closest threshold below this number. Defaults to 0.05 (so fdr of 5%).

`lUseBestThrIfNoFdrThr`
For fdr calculation by motif: if no fdr-associated threshold can be retrieved for one motif, return the strongest threshold identified for any other motif if lUseBestThrIfNoFdrThr is TRUE; if lUseBestThrIfNoFdrThr is FALSE, return the max value for the threshold (so every modification in that motif will be filtered out automatically). Defaults to TRUE.

Examples

```
library(Biostrings)
myGenome <- readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)
```

```
# Preparing a gposPacBioCSV dataset with sequences
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGrangesBaseCSV <- as(myGposPacBioCSV[myGposPacBioCSV$base == "A"], "GRanges")
myGrangesBaseCSVWithSeq <- GetGRangesWindowSeqandParam(
  grangesData = myGrangesBaseCSV,
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1
)

# FDR estimation by motif associated to modifications
myFdr_score_per_motif_list <-
  GetFdrEstListByThresh(
    grangesDataWithSeq = myGrangesBaseCSVWithSeq,
    cNameParamToTest = "score",
    nRoundDigits = 1,
    cModMotifsAsForeground = c("AG", "AT")
  )

GetFdrBasedThreshLimit(
  listFdrEstByThr = myFdr_score_per_motif_list,
  nFdrPropForFilt = 0.05,
  lUseBestThrIfNoFdrThr = TRUE
)
```

GetFdrEstListByThresh *GetFdrEstListByThresh Function (FDREst)*

Description

Return a list with the false discovery rate estimated by threshold on the parameter provided with the GRanges object(s). The thresholds to test are determined by all the possible values of the parameter provided.

Usage

```
GetFdrEstListByThresh(
  grangesDataWithSeq,
  grangesDataWithSeqControl = NULL,
  cNameParamToTest,
  nRoundDigits = 1,
  cModMotifsAsForeground = NULL
)
```

Arguments

`grangesDataWithSeq`
: A GRanges-like object containing, in the extra columns, the parameter to be tested.

`grangesDataWithSeqControl`
: A GRanges-like object containing, in the extra columns, the parameter to be used as a control (usually a non-methylated sample (example: Whole-Genome Amplified/PCR Amplified)). If not NULL, false discovery rate estimation will be calculated using parameter from grangesDataWithSeq and grangesDataWithSeqControl. For example, with "b" as any adenine and with "param" as the parameter to be tested, and for each threshold:

  - foreground = proportion, in the sample, of modified "b" among total "b" = (number of "b" with "param" >= threshold) / total number of "b" in the grangesDataWithSeq
  - background = proportion, in the control, of modified "b" among total "b" = (number of "b" with "param" >= threshold) / total number of "b" in the grangesDataWithSeqControl
  - fdr estimation: background / foreground

  If NULL, only the parameter from grangesDataWithSeq will be used: here the background and foreground will be estimated using motifs associated to modifications (provided with cModMotifsAsForeground) versus other motifs. For example, with "m" as one motif associated to modifications, with "b" as any adenine and with "param" as the parameter to be tested, for each "m" (provided with cModMotifsAsForeground) and for each threshold:

  - foreground = proportion of modified "b", in "m", among total "b" = (number of "b", in "m", with "param" >= threshold) / total number of "b" in "m"
  - background = proportion of modified "b", not in "m", among total "b" = (number of "b", not in "m", with "param" >= threshold) / total number of "b" not in "m"
  - fdr estimation: background / foreground

  Defaults to NULL.

`cNameParamToTest`
: The name of the column containing the parameter to be tested.

`nRoundDigits`
: The number of digits for the thresholds on the parameter. A value of 0 would mean no value to be rounded: this could results in errors for a continuous variable. Defaults to 1.

`cModMotifsAsForeground`
: A character vector of motifs associated to the modifications. If grangesDataWithSeqControl is NULL, this vector must be given. This vector is not used if grangesDataWithSeqControl is not NULL. Defaults to NULL.

Value

A list composed of x dataframes: if grangesDataWithSeqControl is NULL, 1 dataframe; if grangesDataWithSeqControl is not NULL, 1 dataframe by motif provided with cModMotifsAsForeground. In each dataframe:

- fdr: The false discovery rate estimated for this threshold.
- threshold: The threshold on the parameter.
- fdr_cummin: The minimum false discovery rate estimated for this threshold and less stringent thresholds (adjusted false discovery rate).

Examples

```
library(Biostrings)
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a gposPacBioCSV dataset with sequences
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base", "score",
      "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGrangesBaseCSV <- as(myGposPacBioCSV[myGposPacBioCSV$base == "A"], "GRanges")
myGrangesBaseCSVWithSeq <- GetGRangesWindowSeqandParam(
  grangesData = myGrangesBaseCSV,
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1
)

# FDR estimation by motif associated to modifications
myFdr_score_per_motif_list <-
  GetFdrEstListByThresh(
    grangesDataWithSeq = myGrangesBaseCSVWithSeq,
    cNameParamToTest = "score",
    nRoundDigits = 1,
    cModMotifsAsForeground = c("AG", "AT")
  )
myFdr_score_per_motif_list

## NOT RUN!
## FDR estimation versus granges control
# myFdr_score_vsCtrl_list <-
#  GetFdrEstListByThresh(grangesDataWithSeq = myGrangesBaseCSVWithSeq,
#                        grangesDataWithSeqControl = myGrangesBaseCSVWithSeq_control,
#                        cNameParamToTest = "score",
#                        nRoundDigits = 1)
# myFdr_score_vsCtrl_list
```

GetGenomeGRanges *GetGenomeGRanges Function (DaLoad)*

Description

Return a GRanges object from a DNAStringSet object with ranges from 1 to the sequences width (for both strands).

Usage

```
GetGenomeGRanges(dnastringsetGenome)
```

Arguments

`dnastringsetGenome`
A DNAStringSet object to convert as a GRanges object. If sequences have no names, these will be named with "seq" and a number (corresponding to the order in the DNAStringSet object) in the GRanges object.

Examples

```
mySeqs <- Biostrings::DNAStringSet(c("ACCATTGATTAT", "AATATCGACTA", "GACTAT"))
myRanges <- GetGenomeGRanges(mySeqs)
myRanges
```

GetGposCenterFromGRanges *GetGposCenterFromGRanges Function (DaLoad)*

Description

Retrieve, in a GPos object, the positions of the center of the ranges from a GRanges object.

Usage

```
GetGposCenterFromGRanges(grangesData)
```

Arguments

`grangesData` A GRanges object to be converted as a GPos object. The center of each range will be processed in the GPos object output.

Examples

```
myRanges <- as(c("chrI:300-500:+", "chrI:308-680:+", "chrII:30-550:-"), "GRanges")
myCenterPos <- GetGposCenterFromGRanges(grangesData = myRanges)
myCenterPos
```

GetGRangesWindowSeqandParam

GetGRangesWindowSeqandParam Function (GloModAn)

Description

Return the GRanges object provided with the sequence associated to each position (and can also retrieve the sequence around each position).

Usage

```
GetGRangesWindowSeqandParam(
  grangesData,
  grangesGenome,
  dnastringsetGenome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 0
)
```

Arguments

grangesData
: A GRanges object containing Modifications Positions data to be extracted with the sequence.

grangesGenome
: A GRanges object containing the width of each contig.

dnastringsetGenome
: A DNAStringSet object containing the sequence for each contig.

nUpstreamBpToAdd
: Number of base pairs to add upstream of the range from the GRanges object provided to obtain some sequence upstream of range. If some new ranges do not fit in the ranges of the contigs (provided with grangesGenome), those new ranges will be removed. New windows with gaps are also removed. Defaults to 0.

nDownstreamBpToAdd
: Number of base pairs to add downstream of the range from the GRanges object provided to obtain some sequence downstream of range. If some new ranges do not fit in the ranges of the contigs (provided with grangesGenome), those new ranges will be removed. New windows with gaps are also removed. Defaults to 0.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a grangesPacBioGFF datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )

# Retrieve GRanges with sequence
myPositions_Mod_Granges_wSeq <- GetGRangesWindowSeqandParam(
  grangesData = myGrangesPacBioGFF,
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  nUpstreamBpToAdd = 5,
  nDownstreamBpToAdd = 5
)
myPositions_Mod_Granges_wSeq
```

GetListCountsByDist *GetListCountsByDist Function (ModAnnot)*

Description

Return, in dataframes via a list, the counts (or proportion) of provided "Positions" by distance from feature positions. If the input list contains 2 GRanges, 2 dataframes ("Position" vs featureStart; "Position" vs featureEnd) will be exported in the output instead of 1 dataframe ("Position" vs featureStart).

Usage

```
GetListCountsByDist(
  listGRangesDist,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  lGetPropInsteadOfCounts = TRUE
)
```

Arguments

`listGRangesDist`
: A GRangesList with 1 or 2 GRanges objects containing ranges of given "Positions" with their distance to feature positions.

`lAddCorrectedDistFrom5pTo3p`
: If TRUE, the distance will be corrected to reflect 5' to 3' direction and will be stored in a new column (dist_5to3). Defaults to TRUE.

`lGetPropInsteadOfCounts`
: If TRUE, return the proportion of given "Positions" near feature position: counts / sum of counts. If listGRangesDist contains 4 GRanges, the proportion of given "Positions" is calculated near both feature positions: counts / (sum of counts near feature1 + sum of counts near feature2). Defaults to TRUE.

Value

A list with 1 or 2 dataframe(s) containing "Positions" counts by distance to feature positions:

- If 1 GRanges are provided in listGRangesDist, 1 dataframe is provided ("Position" vs featureStart).
- If 2 GRanges are provided in listGRangesDist, 2 dataframes are provided ("Position" vs featureStart; "Position" vs featureEnd).

If a given "Position" is within nWindowSizeAroundFeaturePos base pairs of x different feature positions: this given "Position" will then reported x times with the distance to each feature position.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Retrieve, in a list, dataframes of ModBase counts per Distance values from feature positions
myModDistGRangesList <- GetDistFromFeaturePos(
  grangesAnnotations = myAnnotations,
  cSelectFeature = "gene",
  grangesData = myGrangesPacBioGFF,
  lGetGRangesInsteadOfListCounts = TRUE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
myModDistCountsList <- GetListCountsByDist(
  listGRangesDist = myModDistGRangesList,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  lGetPropInsteadOfCounts = TRUE
)
myModDistCountsList
```

GetMeanParamByContig *GetMeanParamByContig Function (GloModAn)*

Description

Return a list with the mean by strand of the parameter provided for all scaffolds of genome assembly provided.

Usage

```
GetMeanParamByContig(grangesData, dnastringsetGenome, cParamName)
```

Arguments

`grangesData`	A GRanges-like object containing, in the extra columns, the parameter to be analysed.
`dnastringsetGenome`	A DNAStringSet object containing the sequence for each contig.
`cParamName`	The name of the column containing the parameter to be analysed.

Value

A list composed of 3 dataframes: 1 dataframe for each strand and 1 dataframe with both strands. In each dataframe:

- refName: The names of each contig.
- strand: The strand of each contig.
- width: The width of each contig.
- nb_sequenced: The number of bases sequenced by strand for each contig.
- seqPct: The percentage of bases sequenced by strand for each contig (percentage of sequencing).

Examples

```
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# Preparing a gposPacBioCSV dataset
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

myMean_cov_list <- GetMeanParamByContig(
  grangesData = myGposPacBioCSV,
  dnastringsetGenome = myGenome,
  cParamName = "coverage"
)
myMean_cov_list
```

GetModBaseCountsByFeature

GetModBaseCountsByFeature Function (ModAnnot)

Description

Return the Annotation provided with the counts (and counts per KiloBase pairs (kbp)) of the base modified (Mod) and the base letter of the modified base (Base). Example: for Mod="6mA", Base="A"; for Mod="5mC", Base="C".

Usage

```
GetModBaseCountsByFeature(
  grangesAnnotations,
  grangesModPos,
  gposModTargetBasePos,
  lIgnoreStrand = FALSE
)
```

Arguments

`grangesAnnotations`

A GRanges object containing the annotation for the genome assembly corresponding to the grangesModPos and gposModTargetBasePos provided. The Genomic features categories must be in a column named "type".

`grangesModPos` A GRanges object containing Modifications Positions data to be counted.

`gposModTargetBasePos`
A GRanges or GPos object containing Base Positions (which can be targeted by the modification) to be counted.

`lIgnoreStrand` If TRUE, Mod and Base will be counted independently of the strand of each feature. If FALSE, only Mod and Base on the same strand as the feature will be counted. Defaults to FALSE.

Value

A GRanges object based on grangesAnnotations with the counts:

- Modcount: The number of "Mod" within this feature.
- Modcount_perkbp: (The number of "Mod" within this feature divided by the size of the feature)*1000.
- Basecount: The number of "Base" within this feature.
- Basecount_perkbp: (The number of "Base" within this feature divided by the size of the feature)*1000.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

# Retrieve annotations with "Mod" and "Base" counts (and counts per kbp)
myAnn_ModBase_counts <- GetModBaseCountsByFeature(
  grangesAnnotations = myAnnotations,
  grangesModPos = myGrangesPacBioGFF,
  gposModTargetBasePos = myGposPacBioCSV
)
myAnn_ModBase_counts
```

GetModBaseCountsWithinFeature

GetModBaseCountsWithinFeature Function (ModAnnot)

Description

Cut the Annotation provided into x windows of relative size and return, for each window, the counts (and counts per KiloBase pairs (kbp)) of the base modified (Mod) and the base letter of the modified base (Base). Example: for Mod="6mA", Base="A"; for Mod="5mC", Base="C".

Usage

```
GetModBaseCountsWithinFeature(
  grangesAnnotations,
  nWindowsNb = 20,
  grangesModPos,
  gposModTargetBasePos,
  lIgnoreStrand = FALSE
)
```

Arguments

grangesAnnotations
: A GRanges object containing the annotation for the genome assembly corresponding to the grangesModPos and gposModTargetBasePos provided. The Genomic features categories must be in a column named "type".

nWindowsNb
: The number of output windows by feature. The annotation provided (with input ranges) will be cut into this number of output windows. Each output window will have the same size as the other output windows from the same input range. Defaults to 20.

grangesModPos
: A GRanges object containing Modifications Positions data to be counted.

gposModTargetBasePos
: A GRanges or GPos object containing Base Positions (which can be targeted by the modification) to be counted.

lIgnoreStrand
: If TRUE, Mod and Base will be counted independently of the strand of each feature. If FALSE, only Mod and Base on the same strand as the feature will be counted. Defaults to FALSE.

Value

A GRanges object based on grangesAnnotations with the counts:

- Modcount: The number of "Mod" within this window of this feature.
- ModcountSum: Total number of "Mod" within all windows of this feature.
- Modprop: (Modcount / ModcountSum)*100
- Basecount: The number of "Base" within this window of this feature.
- BasecountSum: Total number of "Base" within all windows of this feature.
- Baseprop: (Basecount/BasecountSum)*100

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# loading annotation
library(rtracklayer)
myAnnotations <- readGFFAsGRanges(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
))

# Preparing a grangesPacBioGFF and a grangesPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Retrieve annotations with "Mod" and "Base" counts (and counts per kbp)
myAnn_ModBase_counts_by_window <-
  GetModBaseCountsWithinFeature(
    grangesAnnotations = myAnnotations[myAnnotations$type == "gene", ],
    grangesModPos = myGrangesPacBioGFF,
    gposModTargetBasePos = myGposPacBioCSV,
    nWindowsNb = 20
  )
myAnn_ModBase_counts_by_window
```

GetModRatioByContig *GetModRatioByContig Function (GloModAn)*

Description

Return a list with the Modification ratio (Mod ratio) by strand for all scaffolds of genome assembly provided. For "b" as the base that can be modified, Mod ratio = Number of modified "b" / Total number of "b".

Usage

```
GetModRatioByContig(
  grangesModPos,
  gposModTargetBasePos,
  dnastringsetGenome,
  cBaseLetterForMod
)
```

Arguments

`grangesModPos` A GRanges object containing Modifications Positions data.

`gposModTargetBasePos` A GPos object containing Base Positions that can be targeted by the modification.

`dnastringsetGenome` A DNAStringSet object containing the sequence for each contig.

`cBaseLetterForMod` The name of the base letter of the modified base.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))

# Preparing a grangesPacBioGFF and gposPacBioCSV datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )

myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Mod report
myMod_ratio_list <- GetModRatioByContig(
  grangesModPos = myGrangesPacBioGFF,
  gposModTargetBasePos = myGposPacBioCSV,
  dnastringsetGenome = myGenome,
  cBaseLetterForMod = "A"
)
myMod_ratio_list
```

GetModReportDeepSignal

GetModReportDeepSignal Function (GloModAn)

Description

Return a report with global characteristics of DNA modifications (Mod) distribution in the genome assembly provided. (adapted to data from DeepSignal software)

Usage

```
GetModReportDeepSignal(
  dnastringsetGenome,
  grangesGenome,
  gposDeepSignalMod,
  gposDeepSignalModBase,
  cOrgAssemblyName,
  cBaseLetterForMod,
  cModNameInOutput
)
```

Arguments

`dnastringsetGenome`
A DNAStringSet object containing the sequence for each contig.

`grangesGenome` A GRanges object containing the width of each contig.

`gposDeepSignalMod`
: An UnStitched GPos object containing DeepSignal modified sites data.

`gposDeepSignalModBase`
: An UnStitched GPos object containing DeepSignal modification target sites data.

`cOrgAssemblyName`
: The name of the genome assembly provided.

`cBaseLetterForMod`
: The name of the base letter of the modified base.

`cModNameInOutput`
: Name for the modification in the output.

Examples

```
# preparing genome (simulated)
myGenome <- Biostrings::DNAStringSet(paste0(rep("ATCG", 100000), collapse = ""))
names(myGenome) <- "NC_000001.11"
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Loading Nanopore data
myDeepSignalModPath <- system.file(
  package = "DNAModAnnot", "extdata",
  "FAB39088-288418386-Chr1.CpG.call_mods.frequency.tsv"
)
mygposDeepSignalModBase <- ImportDeepSignalModFrequency(
  cDeepSignalModPath = myDeepSignalModPath,
  lSortGPos = TRUE,
  cContigToBeAnalyzed = "all"
)

# Filtering
mygposDeepSignalMod <- FiltDeepSignal(
  gposDeepSignalModBase = mygposDeepSignalModBase,
  cParamNameForFilter = "frac",
  lFiltParam = TRUE,
  nFiltParamLoBoundaries = 0,
  nFiltParamUpBoundaries = 1,
  cFiltParamBoundariesToInclude = "upperOnly"
)$Mod

# Mod report
myReport_Mod <- GetModReportDeepSignal(
  dnastringsetGenome = myGenome,
  grangesGenome = myGrangesGenome,
  gposDeepSignalMod = as(mygposDeepSignalMod, "GRanges"),
  gposDeepSignalModBase = as(mygposDeepSignalModBase, "GRanges"),
  cOrgAssemblyName = "Test_function",
  cBaseLetterForMod = "C", cModNameInOutput = "5mC"
)
myReport_Mod
```

GetModReportPacBio *GetModReportPacBio Function (GloModAn)*

Description

Return a report with global characteristics of DNA modifications (Mod) distribution in the genome assembly provided. (adapted to PacBio data)

Usage

```
GetModReportPacBio(
  dnastringsetGenome,
  grangesGenome,
  grangesPacBioGFF,
  gposPacBioCSVBase,
  cOrgAssemblyName,
  cBaseLetterForMod,
  cModNameInOutput
)
```

Arguments

dnastringsetGenome	A DNAStringSet object containing the sequence for each contig.
grangesGenome	A GRanges object containing the width of each contig.
grangesPacBioGFF	A GRanges object containing PacBio GFF data.
gposPacBioCSVBase	A GPos object containing PacBio CSV data for sites that can be targeted by the modification only.
cOrgAssemblyName	The name of the genome assembly provided.
cBaseLetterForMod	The name of the base letter of the modified base.
cModNameInOutput	Name for the modification in the output.

Examples

```
# loading genome
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a grangesPacBioGFF datasets
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )

myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGposPacBioCSV <- myGposPacBioCSV[myGposPacBioCSV$base == "A"]

# Mod report
myReport_Mod <- GetModReportPacBio(
  grangesGenome = myGrangesGenome,
  dnastringsetGenome = myGenome,
  grangesPacBioGFF = myGrangesPacBioGFF,
  gposPacBioCSVBase = myGposPacBioCSV,
  cOrgAssemblyName = "ptetraurelia_mac_51",
  cBaseLetterForMod = "A",
  cModNameInOutput = "6mA"
)
myReport_Mod
```

GetSeqPctByContig *GetSeqPctByContig Function (SeQual)*

Description

Return a list with the percentage of sequencing by strand for all scaffolds of genome assembly provided. This function is not adapted for data from DeepSignal.

Usage

```
GetSeqPctByContig(gposPacBioCSV, grangesGenome)
```

Arguments

`gposPacBioCSV` An UnStitched GPos object containing PacBio CSV data to be analysed.

`grangesGenome` A GRanges object containing the width of each contig.

Value

A list composed of 3 dataframes: 1 dataframe by strand and 1 dataframe with both strands. In each dataframe:

- refName: The names of each contig.
- strand: The strand of each contig.
- width: The width of each contig.
- nb_sequenced: The number of bases sequenced by strand for each contig.
- seqPct: The percentage of bases sequenced for each strand for each contig (percentage of sequencing).

Examples

```
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Preparing a gposPacBioCSV dataset
myGposPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE, lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )

myPct_seq_csv <- GetSeqPctByContig(myGposPacBioCSV, grangesGenome = myGrangesGenome)
myPct_seq_csv
```

ImportDeepSignalModFrequency

ImportDeepSignalModFrequency Function (DaLoad)

Description

Import DeepSignal call_modification_frequency.py output file and convert it as an UnStitched GPos object.

Usage

```
ImportDeepSignalModFrequency(
  cDeepSignalModPath,
  cColumnNames = c("chrom", "pos", "strand", "pos_in_strand", "prob_0_sum",
    "prob_1_sum", "count_modified", "count_unmodified", "coverage",
    "modification_frequency", "k_mer"),
  cSelectColumnsToExtract = c("chrom", "pos", "strand", "prob_0_sum", "prob_1_sum",
   "count_modified", "count_unmodified", "coverage", "modification_frequency", "k_mer"),
  lSortGPos = TRUE,
  cContigToBeAnalyzed,
  lKeepSequence = TRUE
)
```

Arguments

`cDeepSignalModPath`
Path to a DeepSignal call_modification_frequency.py output file containing data from all target sites.

`cColumnNames` Names for each column in the DeepSignal call_modification_frequency.py output file. Should not be changed unless some columns are missing in the file to be imported. Defaults to c("chrom", "pos", "strand", "pos_in_strand", "prob_0_sum", "prob_1_sum", "count_modified", "count_unmodified", "coverage", "modification_frequency", "k_mer")

`cSelectColumnsToExtract`
Names of columns to extract from DeepSignal call_modification_frequency.py output file. Less there are columns, faster the file will be loaded. The columns "chrom", "pos" and "strand" are mandatory to convert to a GPos object. Defaults to c("chrom", "pos", "strand", "prob_0_sum", "prob_1_sum", "count_modified", "count_unmodified", "coverage", "modification_frequency", "k_mer")

`lSortGPos` If TRUE, the GPos object will be sorted before being returned: the function will take a longer time to proceed but the GPos Object will require less memory.

`cContigToBeAnalyzed`
Names of contigs for which the data will be kept. If NULL, data from all contigs available will be imported. Defaults to NULL.

`lKeepSequence` If TRUE, the sequence of the base will be retained in one column. Otherwise, it will be discarded to reduce object size. Defaults to TRUE.

Examples

```
# Loading Nanopore data
myDeepSignalModPath <- system.file(
  package = "DNAModAnnot", "extdata",
  "FAB39088-288418386-Chr1.CpG.call_mods.frequency.tsv"
)
mygposDeepSignalModBase <- ImportDeepSignalModFrequency(
  cDeepSignalModPath = myDeepSignalModPath,
  lSortGPos = TRUE,
  cContigToBeAnalyzed = "all"
)
mygposDeepSignalModBase
```

ImportPacBioCSV *ImportPacBioCSV Function (DaLoad)*

Description

Import PacBio CSV file and convert it as an UnStitched GPos object.

Usage

```
ImportPacBioCSV(
  cPacBioCSVPath,
  cSelectColumnsToExtract = c("refName", "tpl", "strand", "base", "score", "ipdRatio",
    "coverage"),
  lKeepExtraColumnsInGPos = TRUE,
  lSortGPos = TRUE,
  cContigToBeAnalyzed = NULL,
  lKeepSequence = TRUE
)
```

Arguments

`cPacBioCSVPath` Path to a PacBio CSV file containing data from all bases sequenced.

`cSelectColumnsToExtract`
Names of columns to extract from PacBio CSV file. Less there are columns, faster the file will be loaded. The columns "refName", "tpl" and "strand" are mandatory to convert to a GPos object. Defaults to c("refName", "tpl", "strand", "base", "score", "ipdRatio", "coverage")

`lKeepExtraColumnsInGPos`
If FALSE, only the contig names, start/end positions and strand will be displayed in the resulting GPos object. Defaults to TRUE.

`lSortGPos` If TRUE, the GPos object will be sorted before being returned: the function will take a longer time to proceed but the GPos Object will require less memory.

`cContigToBeAnalyzed`
Names of contigs for which the data will be kept. If NULL, data from all contigs available will be imported. Defaults to NULL.

`lKeepSequence` If TRUE, the sequence of the base will be retained in one column. Otherwise, it will be discarded to reduce object size. Defaults to TRUE.

Examples

```
# Loading genome data
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
names(myGenome)

# Loading PacBio data
myGrangesPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c("refName", "tpl", "strand", "base", "score", "ipdRatio", "coverage"),
    lKeepExtraColumnsInGPos = TRUE,
    lSortGPos = TRUE,
    cContigToBeAnalyzed = names(myGenome)
  )
myGrangesPacBioCSV
```

```
# Loading PacBio data for 2 scaffolds only
myGrangesPacBioCSV <-
  ImportPacBioCSV(
    cPacBioCSVPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.bases.sca171819.csv"
    ),
    cSelectColumnsToExtract = c(
      "refName", "tpl", "strand", "base",
      "score", "ipdRatio", "coverage"
    ),
    lKeepExtraColumnsInGPos = TRUE,
    lSortGPos = TRUE,
    cContigToBeAnalyzed = c("scaffold51_18", "scaffold51_19")
  )
myGrangesPacBioCSV
```

ImportPacBioGFF *ImportPacBioGFF Function (DaLoad)*

Description

Import PacBio GFF file, extract one modification, rename this modification and convert it as a GRanges object with new colnames similar to PacBio CSV file containing data from all bases sequenced.

Usage

```
ImportPacBioGFF(
  cPacBioGFFPath,
  cNameModToExtract,
  cModNameInOutput,
  cContigToBeAnalyzed = NULL,
  lKeepSequence = TRUE
)
```

Arguments

cPacBioGFFPath	Path to a PacBio GFF file containing modification detection data.
cNameModToExtract	Name of modification to be extracted.
cModNameInOutput	Name for the extracted modification in the output.
cContigToBeAnalyzed	Names of contigs for which the data will be kept. If NULL, data from all contigs available will be imported. Defaults to NULL.
lKeepSequence	If TRUE, the sequence of the base will be retained in one column. Otherwise, it will be discarded to reduce object size. Defaults to TRUE.

Examples

```
# Loading genome data
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
names(myGenome)

# Loading PacBio data
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = names(myGenome)
  )
myGrangesPacBioGFF

# Loading PacBio data for 2 scaffolds only
myGrangesPacBioGFF <-
  ImportPacBioGFF(
    cPacBioGFFPath = system.file(
      package = "DNAModAnnot", "extdata",
      "ptetraurelia.modifications.sca171819.gff"
    ),
    cNameModToExtract = "m6A",
    cModNameInOutput = "6mA",
    cContigToBeAnalyzed = c("scaffold51_18", "scaffold51_19")
  )
myGrangesPacBioGFF
```

PredictMissingAnnotation

PredictMissingAnnotation Function (DaLoad)

Description

Complete annotation with features, such as "intergenic", "antisense_strand_of_gene" or "exon"|"intron", using available features in the annotation.

Usage

```
PredictMissingAnnotation(
  grangesAnnotations,
  grangesGenome,
  cFeaturesColName = "type",
  cGeneCategories = c("gene"),
  lAddIntronRangesUsingExon = FALSE
)
```

Arguments

`grangesAnnotations`
: A GRanges object with the annotation to be completed.

`grangesGenome`
: A GRanges object with number and width of contigs (both strands).

`cFeaturesColName`
: The name of the column containing feature type annotation ("gene", "exon", "mRNA"...). Defaults to "type".

`cGeneCategories`
: The name of the categories considered as genes in the column containing feature type annotation. Defaults to c("gene").

`lAddIntronRangesUsingExon`
: If TRUE, uses "exon" and "mRNA" categories to add "intron" if "intron" is missing, or uses "intron" and "mRNA" categories to add "exon" if "exon" is missing. This will return an error if 2 categories among "mRNA", "exon" and "intron" are missing. Defaults to FALSE.

Examples

```
# Loading genome data
myGenome <- Biostrings::readDNAStringSet(system.file(
  package = "DNAModAnnot", "extdata",
  "ptetraurelia_mac_51_sca171819.fa"
))
myGrangesGenome <- GetGenomeGRanges(myGenome)

# Loading annotation data
myAnnotations <-
  rtracklayer::readGFFAsGRanges(system.file(
    package = "DNAModAnnot", "extdata",
    "ptetraurelia_mac_51_annotation_v2.0_sca171819.gff3"
  ))

# Completing annotation data
levels(myAnnotations$type)
myAnnotations <- PredictMissingAnnotation(
  grangesAnnotations = myAnnotations,
  grangesGenome = myGrangesGenome,
  cFeaturesColName = "type",
  cGeneCategories = c("gene"),
  lAddIntronRangesUsingExon = TRUE
)
levels(myAnnotations$type)
```

Index