Package ‘LOLA’

May 27, 2024

Version 1.34.0
Date 2021-11-20
Title Locus overlap analysis for enrichment of genomic ranges
Description Provides functions for testing overlap of sets of genomic regions with public and custom region set (genomic ranges) databases. This makes it possible to do automated enrichment analysis for genomic region sets, thus facilitating interpretation of functional genomics and epigenomics data.

Author Nathan Sheffield <http://www.databio.org> [aut, cre], Christoph Bock [ctb]
Maintainer Nathan Sheffield <nathan@code.databio.org>
Depends R (>= 2.10)
Imports BiocGenerics, S4Vectors, IRanges, GenomicRanges, data.table, reshape2, utils, stats, methods
Suggests parallel, testthat, knitr, BiocStyle, rmarkdown
Enhances simpleCache, qvalue, ggplot2

VignetteBuilder knitr
License GPL-3

biocViews GeneSetEnrichment, GeneRegulation, GenomeAnnotation, SystemsBiology, FunctionalGenomics, ChIPSeq, MethylSeq, Sequencing

URL http://code.databio.org/LOLA
BugReports http://github.com/nsheff/LOLA
RoxygenNote 7.1.0
git_url https://git.bioconductor.org/packages/LOLA
git_branch RELEASE_3_19
git_last_commit ef2bf76
git_last_commit_date 2024-04-30
Repository Bioconductor 3.19
Date/Publication 2024-05-27
If you want to test for differential enrichment within your user sets, you can restrict the universe to only regions that are covered in at least one of your sets. This function helps you build just such a restricted universe.
**checkUniverse Appropriateness**

**Description**
If you want to test for differential enrichment within your usersets, you can restrict the universe to only regions that are covered in at least one of your sets. This function helps you build just such a restricted universe.

**Usage**
```
buildRestrictedUniverse(userSets)
```

**Arguments**
- `userSets`: The userSets you will pass to the enrichment calculation.

**Value**
A restricted universe

**Examples**
```
data("sample_input", package="LOLA") # load userSets
restrictedUniverse = buildRestrictedUniverse(userSets)
```

---

**checkUniverse Appropriateness**

*Check universe appropriateness*

**Description**
Checks to see if the universe is appropriate for the userSets. Anything in the userSets should be present in the universe. In addition, 2 different regions in the userSets should not overlap the same region in the universe.

**Usage**
```
checkUniverse Appropriateness(userSets, userUniverse, cores = 1, fast = FALSE)
```

**Arguments**
- `userSets`: Regions of interest
- `userUniverse`: Regions tested for inclusion in userSets
- `cores`: Number of processors
- `fast`: Skip the (slow) test for many-to-many relationships

**Value**
No return value.
Examples

data("sample_input", package="LOLA") # load userSet
data("sample_universe", package="LOLA") # load userUniverse
checkUniverseAppropriateness(userSets, userUniverse)

cleanws

Description

`cleanws` takes multi-line, code formatted strings and just formats them as simple strings.

Usage

`cleanws(string)`

Arguments

- `string`: string to clean

Value

A string with all consecutive whitespace characters, including tabs and newlines, merged into a single space.

countOverlapsAnyRev

Description

`countOverlapsAnyRev` just a reverser. Reverses the order of arguments and passes them untouched to `countOverlapsAny` – so you can use it with `lapply`.

Usage

`countOverlapsAnyRev(subj, quer)`

Arguments

- `subj`: Subject
- `quer`: Query

Value

Results from `countOverlapsAny`
Given a single row from an enrichment table calculation, finds the set of overlaps between the user set and the test set. You can then use these, for example, to get sequences for those regions.

Usage

\[
\text{extractEnrichmentOverlaps}(\text{locResult}, \text{userSets}, \text{regionDB})
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locResult</td>
<td>Results from runLOLA function</td>
</tr>
<tr>
<td>userSets</td>
<td>User sets passed to the runLOLA function</td>
</tr>
<tr>
<td>regionDB</td>
<td>Region database used</td>
</tr>
</tbody>
</table>

Value

userSets overlapping the supplied database entry.

Examples

```r
dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation=dbPath)
data("sample_universe", package="LOLA")
data("sample_input", package="LOLA")

getRegionSet(regionDB, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionSet(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionFile(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")

res = runLOLA(userSets, userUniverse, regionDB, cores=1)
locResult = res[2,]
extractEnrichmentOverlaps(locResult, userSets, regionDB)
writeCombinedEnrichment(locResult, "temp_outfolder")

userSetsRedefined = redefineUserSets(userSets, userUniverse)
resRedefined = runLOLA(userSetsRedefined, userUniverse, regionDB, cores=1)

g = plotTopLOLAEnrichments(resRedefined)
```
getRegionFile  

*Grab the filename for a single region set from a database specified by filename.*

**Description**

Like `getRegionSet` but returns a filename instead of a GRanges object. Given a local filename, returns a complete absolute path so you can read that file in.

**Usage**

```
getRegionFile(dbLocation, filenames, collections = NULL)
```

**Arguments**

- `dbLocation`: folder of regionDB
- `filenames`: Filename(s) of a particular region set to grab.
- `collections`: (optional) subset of collections to list

**Value**

A filename the specified file in the regionDB.

**Examples**

```r
dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation=dbPath)
data("sample_universe", package="LOLA")
data("sample_input", package="LOLA")

generateTestCollections()
generateTestUserSets()
getRegionSet(regionDB, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionSet(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionFile(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")

res = runLOLA(userSets, userUniverse, regionDB, cores=1)
locResult = res[2,]
extractEnrichmentOverlaps(locResult, userSets, regionDB)
writeCombinedEnrichment(locResult, "temp_outfolder")

userSetsRedefined = redefineUserSets(userSets, userUniverse)
resRedefined = runLOLA(userSetsRedefined, userUniverse, regionDB, cores=1)

g = plotTopLOLAEEnrichments(resRedefined)
```
getRegionSet

Grab a single region set from a database, specified by filename.

Description

If you want to work with a LOLA regionDB region set individually, this function can help you. It can extract individual (or subsets of) region sets from either loaded regionDBs, loaded with loadRegionDB(), or from a database on disk, where only the region sets of interest will be loaded.

Usage

getRegionSet(regionDB, filenames, collections = NULL)

Arguments

regionDB A region database loaded with loadRegionDB().
filenames Filename(s) of a particular region set to grab.
collections (optional) subset of collections to list

Value

A GRanges object derived from the specified file in the regionDB.

Examples

dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation=dbPath)
data("sample_universe", package="LOLA")
data("sample_input", package="LOLA")

getRegionSet(regionDB, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionSet(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionFile(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")

res = runLOLA(userSets, userUniverse, regionDB, cores=1)
locResult = res[2,]
extractEnrichmentOverlaps(locResult, userSets, regionDB)
writeCombinedEnrichment(locResult, "temp_outfolder")

userSetsRedefined = redefineUserSets(userSets, userUniverse)
resRedefined = runLOLA(userSetsRedefined, userUniverse, regionDB, cores=1)

g = plotTopLOLAEnrichments(resRedefined)
lapplyAlias

Function to run lapply or mclapply, depending on the option set in `getOption("mc.cores")`, which can be set with `setLapplyAlias()`.

Description

Function to run lapply or mclapply, depending on the option set in `getOption("mc.cores")`, which can be set with `setLapplyAlias()`.

Usage

lapplyAlias(..., mc.preschedule = TRUE)

Arguments

... Arguments passed lapply() or mclapply()
mc.preschedule Argument passed to mclapply

Value

Result from lapply or parallel::mclapply

listRegionSets

Lists the region sets for given collection(s) in a region database on disk.

Description

Lists the region sets for given collection(s) in a region database on disk.

Usage

listRegionSets(regionDB, collections = NULL)

Arguments

regionDB File path to region database
collections (optional) subset of collections to list

Value

a list of files in the given collections

Examples

dbPath = system.file("extdata", "hg19", package="LOLA")
listRegionSets(dbPath)
listToGRangesList  

converts a list of GRanges into a GRangesList; strips all metadata.

Description  

converts a list of GRanges into a GRangesList; strips all metadata.

Usage  

listToGRangesList(lst)

Arguments  

lst  
a list of GRanges objects

Value  
a GRangesList object

loadRegionDB  

Helper function to annotate and load a regionDB, a folder with subfolder collections of regions.

Description  

Helper function to annotate and load a regionDB, a folder with subfolder collections of regions.

Usage  

loadRegionDB(dbLocation, useCache = TRUE, limit = NULL, collections = NULL)

Arguments  

dbLocation  
folder where your regionDB is stored, or list of such folders

useCache  
uses simpleCache to cache and load the results

limit  
You can limit the number of regions for testing. Default: NULL (no limit)

collections  
Restrict the database loading to this list of collections

Value  

regionDB list containing database location, region and collection annotations, and regions GRangesList

Examples  

dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation=dbPath)
mergeRegionDBs

**LOLA**  
*Genome locus overlap analysis.*

**Description**

Run, Lola!

**Author(s)**

Nathan Sheffield

**References**

[http://github.com/sheffien](http://github.com/sheffien)

**mergeRegionDBs**  
*Given two regionDBs, (lists returned from loadRegionDB()), This function will combine them into a single regionDB. This will enable you to combine, for example, LOLA Core databases with custom databases into a single analysis.*

**Description**

Given two regionDBs, (lists returned from loadRegionDB()), This function will combine them into a single regionDB. This will enable you to combine, for example, LOLA Core databases with custom databases into a single analysis.

**Usage**

```r
mergeRegionDBs(dbA, dbB)
```

**Arguments**

- **dbA**  
  First regionDB database.

- **dbB**  
  Second regionDB database.

**Value**

A combined regionDB.

**Examples**

```r
dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbPath)
combinedRegionDB = mergeRegionDBs(regionDB, regionDB)
```
**nlist**

*Named list function.*

**Description**

This function is a drop-in replacement for the base list() function, which automatically names your list according to the names of the variables used to construct it. It seamlessly handles lists with some names and others absent, not overwriting specified names while naming any unnamed parameters. Took me awhile to figure this out.

**Usage**

```r
nlist(...)```

**Arguments**

`...` arguments passed to list()

**Value**

A named list object.

---

**plotTopLOLAEnrichments**

*Given some results (you grab the top ones on your own), this plots a barplot visualizing their odds ratios.*

**Description**

Given some results (you grab the top ones on your own), this plots a barplot visualizing their odds ratios.

**Usage**

```r
plotTopLOLAEnrichments(data)```

**Arguments**

`data` A results table returned from runLOLA()

**Value**

Returns a ggplot2 plot object.
readBed

Imports bed files and creates GRanges objects, using the fread() function from data.table.

Description
Imports bed files and creates GRanges objects, using the fread() function from data.table.

Usage
readBed(file)

Arguments
file File name of bed file.

Value
GRanges Object

Examples
a = readBed(system.file("extdata", "examples/combined_regions.bed", package="LOLA"))
**readCollection**

Given a bunch of region set files, read in all those flat (bed) files and create a GRangesList object holding all the region sets. This function is used by readRegionGRL to process annotation objects.

**Description**

Given a bunch of region set files, read in all those flat (bed) files and create a GRangesList object holding all the region sets. This function is used by readRegionGRL to process annotation objects.

**Usage**

```r
readCollection(filesToRead, limit = NULL)
```

**Arguments**

- `filesToRead` a vector containing bed files
- `limit` for testing purposes, limit the number of files read. NULL for no limit (default).

**Value**

A GRangesList with the GRanges in the filesToRead.

**Examples**

```r
files = list.files(system.file("extdata", "hg19/ucsc_example/regions", package="LOLA"), pattern="*.bed")
regionAnno = readCollection(files)
```

---

**readCollectionAnnotation**

*Read collection annotation*

**Description**

Read collection annotation

**Usage**

```r
readCollectionAnnotation(dbLocation, collections = NULL)
```

**Arguments**

- `dbLocation` Location of the database
- `collections` Restrict the database loading to this list of collections. Leave NULL to load the entire database (Default).
# readCollectionFiles

Given a database and a collection, this will create the region annotation data.table; either giving a generic table based on file names, or by reading in the annotation data.

## Description

Given a database and a collection, this will create the region annotation data.table; either giving a generic table based on file names, or by reading in the annotation data.

## Usage

```r
readCollectionFiles(dbLocation, collection, refreshSizes = FALSE)
```

## Arguments

- **dbLocation**: folder where your regionDB is stored.
- **collection**: Collection folder to load
- **refreshSizes**: should I recreate the sizes files documenting how many regions (lines) are in each region set?

## Value

A data.table annotating the regions in the collections.

## Examples

```r
dbPath = system.file("extdata", "hg19", package="LOLA")
collectionAnno = readCollectionAnnotation(dbLocation=dbPath)

readCollectionFiles = function(dbLocation, collection, refreshSizes = FALSE) {
  # Implementation
}
```
readRegionGRL

This function takes a region annotation object and reads in the regions, returning a GRangesList object of the regions.

Description

This function takes a region annotation object and reads in the regions, returning a GRangesList object of the regions.

Usage

```r
readRegionGRL(
  dbLocation,
  annoDT,
  refreshCaches = FALSE,
  useCache = TRUE,
  limit = NULL
)
```

Arguments

- `dbLocation`: folder of regionDB
- `annoDT`: output of readRegionSetAnnotation().
- `refreshCaches`: should I recreate the caches?
- `useCache`: uses simpleCache to cache and load the results
- `limit`: for testing purposes, limit the number of files read. NULL for no limit (default).

Value

GRangesList object

Examples

```r
dbPath = system.file("extdata", "hg19", package="LOLA")
regionAnno = readRegionSetAnnotation(dbLocation=dbPath)
regionGRL = readRegionGRL(dbLocation= dbPath, regionAnno, useCache=FALSE)
```
readRegionSetAnnotation

Given a folder containing region collections in subfolders, this function will either read the annotation file if one exists, or create a generic annotation file.

Description

Given a folder containing region collections in subfolders, this function will either read the annotation file if one exists, or create a generic annotation file.

Usage

readRegionSetAnnotation(
  dbLocation,
  collections = NULL,
  refreshCaches = FALSE,
  refreshSizes = TRUE,
  useCache = TRUE
)

Arguments

dbLocation  folder where your regionDB is stored.
collections  Restrict the database loading to this list of collections Leave NULL to load the entire database (Default).
refreshCaches  should I recreate the caches? Default: FALSE
refreshSizes  should I refresh the size files? Default: TRUE
useCache  Use simpleCache to store results and load them?

Value

Region set annotation (data.table)

Examples

dbPath = system.file("extdata", "hg19", package="LOLA")
regionAnno = readRegionSetAnnotation(dbLocation=dbPath)
redefineUserSets

This function will take the user sets, overlap with the universe, and redefine the user sets as the set of regions in the user universe that overlap at least one region in user sets. This makes for a more appropriate statistical enrichment comparison, as the user sets are actually exactly the same regions found in the universe otherwise, you can get some weird artifacts from the many-to-many relationship between user set regions and universe regions.

Description

This function will take the user sets, overlap with the universe, and redefine the user sets as the set of regions in the user universe that overlap at least one region in user sets. This makes for a more appropriate statistical enrichment comparison, as the user sets are actually exactly the same regions found in the universe otherwise, you can get some weird artifacts from the many-to-many relationship between user set regions and universe regions.

Usage

redefineUserSets(userSets, userUniverse, cores = 1)

Arguments

userSets Regions of interest
userUniverse Regions tested for inclusion in userSets
cores Number of processors

Value

userSets redefined in terms of userUniverse

Examples

dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation=dbPath)
data("sample_universe", package="LOLA")
data("sample_input", package="LOLA")

getRegionSet(regionDB, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionSet(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionFile(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")

res = runLOLA(userSets, userUniverse, regionDB, cores=1)
locResult = res[2,]
extactEnrichmentOverlaps(locResult, userSets, regionDB)
writeCombinedEnrichment(locResult, "temp_outfolder")
userSetsRedefined = redefineUserSets(userSets, userUniverse)
resRedefined = runLOLA(userSetsRedefined, userUniverse, regionDB, cores=1)

g = plotTopLOLAEnrichments(resRedefined)

replaceFileExtension  
This will change the string in filename to have a new extension

Description
This will change the string in filename to have a new extension

Usage
replaceFileExtension(filename, extension)

Arguments
filename  string to convert
extension  new extension

Value
Filename with original extension deleted, replaced by provided extension

runLOLA   Enrichment Calculation

Description
Workhorse function that calculates overlaps between userSets, and then uses a fisher’s exact test
rank them by significance of the overlap.

Usage
runLOLA(
    userSets, 
    userUniverse, 
    regionDB, 
    minOverlap = 1, 
    cores = 1, 
    redefineUserSets = FALSE,  
    direction = "enrichment"  
)
Arguments

- **userSets**: Regions of interest
- **userUniverse**: Regions tested for inclusion in userSets
- **regionDB**: Region DB to check for overlap, from loadRegionDB()
- **minOverlap**: (Default: 1) Minimum bases required to count an overlap
- **cores**: Number of processors
- **redefineUserSets**: run redefineUserSets() on your userSets?
- **direction**: Defaults to "enrichment", but may also accept "depletion", which will swap the direction of the fisher test (use 'greater' or 'less' value passed to the 'alternative' option of fisher.test)

Value

Data.table with enrichment results. Rows correspond to individual pairwise fisher’s tests comparing a single userSet with a single databaseSet. The columns in this data.table are: userSet and dbSet: index into their respective input region sets. pvalueLog: -log10(pvalue) from the fisher’s exact result; oddsRatio: result from the fisher’s exact test; support: number of regions in userSet overlapping databaseSet; rnkPV, rnkOR, rnkSup: rank in this table of p-value, oddsRatio, and Support respectively. The –value is the negative natural log of the p-value returned from a one-sided fisher’s exact test. maxRnk, meanRnk: max and mean of the 3 previous ranks, providing a combined ranking system. b, c, d: 3 other values completing the 2x2 contingency table (with support). The remaining columns describe the dbSet for the row.

If you have the qvalue package installed from bioconductor, runLOLA will add a q-value transformation to provide FDR scores automatically.

Examples

```r
dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation=dbPath)
data("sample_universe", package="LOLA")
data("sample_input", package="LOLA")

getRegionSet(regionDB, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionSet(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionFile(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")

res = runLOLA(userSets, userUniverse, regionDB, cores=1)
locResult = res[2,]
extractEnrichmentOverlaps(locResult, userSets, regionDB)
writeCombinedEnrichment(locResult, "temp_outfolder")

userSetsRedefined = redefineUserSets(userSets, userUniverse)
resRedefined = runLOLA(userSetsRedefined, userUniverse, regionDB, cores=1)

g = plotTopLOLAEnrichments(resRedefined)
```
sampleGRL

Function to sample regions from a GRangesList object, in specified proportion

Description
Function to sample regions from a GRangesList object, in specified proportion

Usage
sampleGRL(GRL, prop)

Arguments
- GRL: GRangesList from which to sample
- prop: vector with same length as GRL, of values between 0-1, proportion of the list to select

Value
A sampled subset of original GRangesList object.

setLapplyAlias

To make parallel processing a possibility but not required, I use an lapply alias which can point at either the base lapply (for no multicore), or it can point to mclapply, and set the options for the number of cores (what mclapply uses). With no argument given, returns instead the number of cpus currently selected.

Description
To make parallel processing a possibility but not required, I use an lapply alias which can point at either the base lapply (for no multicore), or it can point to mclapply, and set the options for the number of cores (what mclapply uses). With no argument given, returns instead the number of cpus currently selected.

Usage
setLapplyAlias(cores = 0)

Arguments
- cores: Number of cpus

Value
None
setSharedDataDir

setSharedDataDir Sets global variable specifying the default data directory.

Description

setSharedDataDir Sets global variable specifying the default data directory.

Usage

setSharedDataDir(sharedDataDir)

Arguments

sharedDataDir directory where the shared data is stored.

Value

No return value.

Examples

setSharedDataDir("project/data")

splitDataTable

Efficiently split a data.table by a column in the table

Description

Efficiently split a data.table by a column in the table

Usage

splitDataTable(DT, splitFactor)

Arguments

DT Data.table to split

splitFactor Column to split, which can be a character vector or an integer.

Value

List of data.table objects, split by column
splitFileIntoCollection

This function will take a single large bed file that is annotated with a column grouping different sets of similar regions, and split it into separate files for use with the LOLA collection format.

Description

This function will take a single large bed file that is annotated with a column grouping different sets of similar regions, and split it into separate files for use with the LOLA collection format.

Usage

splitFileIntoCollection(
  filename,
  splitCol,
  collectionFolder = NULL,
  filenamePrepend = ""
)

Arguments

filename       the file to split
splitCol       factor column that groups the lines in the file by set. It should be an integer.
collectionFolder
name of folder to place the new split files.
filenamePrepend string to prepend to the filenames. Defaults to blank.

Value

No return value.

Examples

combFile = system.file("extdata", "examples/combined_regions.bed", package="LOLA")
splitFileIntoCollection(combFile, 4)
userSets

An example set of regions, sampled from the example database.

Description

A dataset containing a few sample regions.

Usage

data(sample_input)

Format

A GRangesList object

Value

No return value.

Examples

## Not run:
This is how I produced the sample data sets:
```
dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation= dbPath)
userSetA = reduce(do.call(c, (sampleGRL(regionDB$regionGRL,
prop=c(.1,.25,.05,.05,0)))))
userSetB = reduce(do.call(c, (sampleGRL(regionDB$regionGRL,
prop=c(.2,.05,.05,.05,0)))))
```
userSets = GRangesList(setA=userSetA, setB=userSetB)
userUniverse = reduce(do.call(c, regionDB$regionGRL))
save(userSets, file="sample_input.RData")
save(userUniverse, file="sample_universe.RData")

## End(Not run)

userUniverse

A reduced GRanges object from the example regionDB database

Description

A reduced GRanges object from the example regionDB database

Usage

data(sample_universe)
writeCombinedEnrichment

Function for writing output all at once: combinedResults is an table generated by "locationEnrichment()" or by rbinding category/location results. Writes all enrichments to a single file, and also spits out the same data divided into groups based on userSets, and Databases, just for convenience. disable this with an option.

writeCombinenEnrichment

Function for writing output all at once: combinedResults is an table generated by "locationEnrichment()" or by rbinding category/location results. Writes all enrichments to a single file, and also spits out the same data divided into groups based on userSets, and Databases, just for convenience. disable this with an option.

write.tsv

Wrapper of write.table that provides defaults to write a simple .tsv file. Passes additional arguments to write.table

Description

Wrapper of write.table that provides defaults to write a simple .tsv file. Passes additional arguments to write.table

Usage

write.tsv(...)
writeDataTableSplitByColumn

Usage

```
writeCombinedEnrichment(
  combinedResults,
  outFolder = NULL,
  includeSplits = TRUE
)
```

Arguments

- `combinedResults`: enrichment results object
- `outFolder`: location to write results on disk
- `includeSplits`: also include individual files for each user set and database?

Value

No return value.

Examples

```
dbPath = system.file("extdata", "hg19", package="LOLA")
regionDB = loadRegionDB(dbLocation=dbPath)
data("sample_universe", package="LOLA")
data("sample_input", package="LOLA")

getRegionSet(regionDB, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionSet(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")
getRegionFile(dbPath, collections="ucsc_example", filenames="vistaEnhancers.bed")

res = runLOLA(userSets, userUniverse, regionDB, cores=1)
locResult = res[2,]
extractEnrichmentOverlaps(locResult, userSets, regionDB)
writeCombinedEnrichment(locResult, "temp_outfolder")

userSetsRedefined = redefineUserSets(userSets, userUniverse)
resRedefined = runLOLA(userSetsRedefined, userUniverse, regionDB, cores=1)

g = plotTopLOLAEnrichments(resRedefined)
```

writeDataTableSplitByColumn

*Given a data table and a factor variable to split on, efficiently divides the table and then writes the different splits to separate files, named with filePrepend and numbered according to split.*
Description
Given a data table and a factor variable to split on, efficiently divides the table and then writes the different splits to separate files, named with filePrepend and numbered according to split.

Usage

```
writeDataTableSplitByColumn(
  DT,
  splitFactor,
  filePrepend = ",",
  orderColumn = NULL
)
```

Arguments

- **DT**: data.table to split
- **splitFactor**: column of DT to split on
- **filePrepend**: notation string to prepend to output files
- **orderColumn**: column of DT to order on (defaults to the first column)

Value

number of splits written
Index

* datasets
  userSets, 23
  userUniverse, 23
buildRestrictedUniverse, 2
checkUniverseAppropriateness, 3
cleanws, 4
countOverlapsAnyRev, 4
extractEnrichmentOverlaps, 5
getRegionFile, 6
getRegionSet, 7
lapplyAlias, 8
listRegionSets, 8
listToGRangesList, 9
loadRegionDB, 9
LOLA, 10
mergeRegionDBs, 10
nlist, 11
plotTopLOLAEnrichments, 11
readBed, 12
readCollection, 13
readCollectionAnnotation, 13
readCollectionFiles, 14
readRegionGRL, 15
readRegionSetAnnotation, 16
redefineUserSets, 17
replaceFileExtension, 18
runLOLA, 18
sampleGRL, 20
setLapplyAlias, 20
setSharedDataDir, 21
splitDataTable, 21
splitFileIntoCollection, 22
userSets, 23
userUniverse, 23
write.tsv, 24
writeCombinedEnrichment, 24
writeDataTableSplitByColumn, 25