Package 'slinky'

October 17, 2020

Type Package

Title Putting the fun in LINCS L1000 data analysis

Version 1.6.0

Date 2019-10-04

Author Eric J. Kort

Maintainer Eric J. Kort <eric.kort@vai.org>

Description Wrappers to query the L1000 metadata available via the clue.io REST API as well as helpers for dealing with LINCS gctx files, extracting data sets of interest, converting to SummarizedExperiment objects, and some facilities for performing streamlined differential expression analysis of these data sets.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5.0)

Imports SummarizedExperiment, curl, dplyr, foreach, httr, stats, utils, methods, readr, rhdf5, jsonlite, tidyr

Collate 'slinky.R' 'slinky_clue.R' 'slinky_sumexp.R' 'slinky_download.R' 'slinky_info.R' 'slinky_gctx.R' 'slinky_scoring.R' 'slinky_chdir.R' 'slinky_ks.R'

Suggests GeoDE, doParallel, testthat, knitr, rmarkdown, ggplot2, Rtsne, Biobase, BiocStyle

VignetteBuilder knitr

RoxygenNote 6.1.1

LazyData true

biocViews DataImport, ThirdPartyClient, GeneExpression, DifferentialExpression, GeneSetEnrichment, PatternLogic

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

git_branch RELEASE_3_11

git_last_commit 0d50bdb

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

R topics documented:

`.zs` *.zs*

Description

The .zs function provides method for calculating robust z-scores

Usage

```
.zs(treat, control)
```

Arguments

`treat`	a matrix of values for the treated samples
`control`	a matrix of values for the control samples

Value

A vector of z-scores.

chDir *chDir*

Description

Convenience wrapper to calculate Characteristic Direction Unity Vector based on two datasets. There are a few steps involved in getting the data formatted for the 'chdirAnalysis' function. This function takes care of that for you. Also, the chdirSig function is not exported from the package GeoDE so we copy it here to be able to circumvent plotting (which may not be desired for the high throughput applications targeted by this package).

Usage

```
chDir(x, treated, control)
```

Arguments

x	A Slinky object.
treated	Expression data for treated samples, as 'data.frame', 'matrix', or 'SummarizedExperiment'
control	Expression data for control samples, as 'data.frame', 'matrix', or 'SummarizedExperiment'

Value

Column matrix of characteristic direction scores for each gene.

closeAll *closeAll*

Description

Close any open HDF5 (gctx) file connections.

Usage

```
closeAll(x)

## S4 method for signature 'Slinky'
closeAll(x)
```

Arguments

x	a Slinky Object

Value

None. Called for side effect of closing connections.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
closeAll(sl)
```

---

clue *clue*

Description

Wrapper for clue.io REST calls

Usage

```
clue(x, endpoint = c("sigs", "cells", "genes", "perts", "plates",
  "profiles", "rep_drugs", "rep_drug_indications", "pcls"), fields = "",
  where_clause = NULL, ids = NULL, limit = 0, count = FALSE,
  unpack_sigs = TRUE, poscon = c("omit", "keep"), cl = NULL,
  verbose = FALSE)

## S4 method for signature 'Slinky'
clue(x, endpoint = c("sigs", "cells", "genes",
  "perts", "plates", "profiles", "rep_drugs", "rep_drug_indications",
  "pcls"), fields = "", where_clause = NULL, ids = NULL, limit = 0,
  count = FALSE, unpack_sigs = TRUE, poscon = c("omit", "keep"),
  cl = NULL, verbose = FALSE)
```

Arguments

x	a Slinky Object
endpoint	The endpoint to query. Default is 'sigs'.
fields	Optional vector of fields to return.
where_clause	Optional where_clause clause. Must be named list (e.g. list(field='value')
ids	Optional vector of ids to fetch for sigs or profiles endpoints. Should not be used together with where_clause or count.
limit	Optional limit to number of instances (samples to return
count	Should we just return the count of intances satisfying the query rather than the data? Default is FALSE.
unpack_sigs	The sigs endpoint returns multiple distil_ids per row. Should we unpack these to one per row?
poscon	Instances of type `trt_poscon` are recoded as `trt_cp` in clue.io's `sigs` endpoint. This can lead to unexpected results downstream. To keep these instances, specify `poscon='keep'`

cl	Optional cluster object to parallelize this operation. If verbose is TRUE, use this pattern in order for progress bar to update: cl <-parallel::makeCluster(4,outfile=\"\")
verbose	Do you want to know how things are going? Default is false.

Value

Data returned by Slinky.api as a data.frame

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
amox <- clue(sl, where_clause = list("pert_iname" = "amoxicillin",
                                     "cell_id" = "MCF7",
                                     "is_gold" = TRUE))
```

---

clueCount *clueCount*

Description

Wrapper for Slinky.io REST calls to retrieve record counts

Usage

```
clueCount(x, endpoint = c("sigs", "cells", "genes", "perts", "plates",
  "profiles", "rep_drugs", "rep_drug_indications", "pcls"),
  where_clause = "")

## S4 method for signature 'Slinky'
clueCount(x, endpoint = c("sigs", "cells", "genes",
  "perts", "plates", "profiles", "rep_drugs", "rep_drug_indications",
  "pcls"), where_clause = "")
```

Arguments

x	a Slinky Object
endpoint	The endpoint to query, default is 'sigs'.
where_clause	Optional where_clause clause. Must be named list (e.g. list(field='value')

Value

Count of records satisfying query

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
amox_count <- clueCount(sl, where_clause = list("pert_iname" = "amoxicillin",
                                    "cell_id" = "MCF7",
                                    "is_gold" = TRUE))
```

---

clueInstances *clueInstances*

Description

Convenience wrapper to query function to retrieve instance ids meeting specified criteria.

Usage

```
clueInstances(x, where_clause = NULL, verbose = FALSE,
  poscon = c("omit", "keep"))

## S4 method for signature 'Slinky'
clueInstances(x, where_clause = NULL,
  verbose = FALSE, poscon = c("omit", "keep"))
```

Arguments

x	a Slinky Object
where_clause	Filter terms, as a list of terms, e.g. `list(pert_type='trt_cp','is_gold'=TRUE)`. Terms will be joined by AND logic.
verbose	Do you want to know how things are going? Default is false.
poscon	Instances of type `trt_poscon` are recoded as `trt_cp` in clue.io's `sigs` endpoint. This can lead to unexpected results downstream. To keep these instances, specify `poscon='keep'`

Value

Vector of ids matching criteria. This is a convenience wrapper to the signature API which queries clue.io and unwraps response.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
amox_ids <- clueInstances(sl, where_clause = list("pert_iname" = "amoxicillin",
                                    "cell_id" = "MCF7",
                                    "is_gold" = TRUE))
```

---

clueVehicle *clueVehicle*

Description

Fetch the vehicle control applicable to given ids (distil_id). Expects that perturbagen is of type trt_cp.

Usage

```
clueVehicle(x, ids, verbose = FALSE)

## S4 method for signature 'Slinky'
clueVehicle(x, ids, verbose = FALSE)
```

Arguments

x	a Slinky Object
ids	The distil_id(s) to lookup.
verbose	Do you want to know how things are going? Default is FALSE.

Value

The name of the vehicle control for the queried perturbagen(s). This is a convenience wrapper to the profiles API which queries clue.io and unwraps response.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
```

```
amox <- clue(sl, where_clause = list("pert_iname" = "amoxicillin",
                                     "cell_id" = "MCF7",
                                     "is_gold" = TRUE))
amox.ctrl <- clueVehicle(sl, amox$distil_id)
```

---

coerce *as("Slinky", "SummarizedExperiment")*

Description

Create SummarizedExperiment object from Slinky object. Data will be loaded from the GCTX file, combined with metadata from the info file, and wrapped in a SummarizedExperiment object. Note that this may take a long time for the entire data set from the L1000 project. For most use cases, a subset will be desired (e.g. `as(sl[1:987,1:50],SummarizedExperiment)`. See the loadL1K method for a more flexible way to create a SummarizedExperiment object based on various query parameters.

See Also

loadL1K

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
sumex <- as(sl[, 1:20], "SummarizedExperiment")
```

---

colnames *colnames*

Description

Retrieve column names from LINCS gctx datafile

Usage

```
colnames(x, do.NULL = TRUE, prefix = "col")

## S4 method for signature 'Slinky'
colnames(x)
```

Arguments

x	a Slinky Object
do.NULL	Ignored (see ?base::colnames)
prefix	Ignored (see ?base::colnames)

Value

Names of columns from gctx file The gctx file is an HDF5 formatted file with several sections (groups) containing the column and row level metadata as well as the expression data itself. Note that for best performance, if a subset of colnames is desired, subset the slinky object itself, not the colnames, to avoid loading the entire set of colnames from the the gctx file. That is, `names <-colnames(x[,1:50])` will be considerably faster than `names <-colnames(x)[1:50]`. The `do.NULL` and `prefix` arguments from `base::colnames` do not apply here (as the slinky object will always have column names), and will be silently ignored if provided.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
colnames(sl[,1:5])
```

---

controls *controls #' Fetch the same plate control samples applicable for given ids (distil_id). Expects that the specified ids have pert_type of trt_sh or trt_cp.*

Description

controls #' Fetch the same plate control samples applicable for given ids (distil_id). Expects that the specified ids have pert_type of trt_sh or trt_cp.

Usage

```
controls(x, ids, verbose = FALSE, cl = NULL)

## S4 method for signature 'Slinky'
controls(x, ids, verbose = FALSE, cl = NULL)
```

Arguments

x	A slinky object
ids	The distil_id(s) to lookup.
verbose	Do you want to know how things are going? Default is FALSE.
cl	Optional cluster object to parallelize this operation. If verbose is TRUE, use this pattern in order for progress bar to update: `cl <-parallel::makeCluster(4,outfile=\"\")`

Value

The name of the vehicle control for the queried perturbagen(s). \For a given set of distil_ids, this function finds the distil_ids for the corresponding control samples based on the the pert_type and (for trt_cp) the specified vehicle. The returned dataframe can be used, among other things, to create a control dataset for differential expression or other analysis. See also diffexp.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
amox_gold <- clueInstances(sl, where_clause = list("pert_type" = "trt_cp",
   "pert_iname" = "amoxicillin",
   "cell_id" = "MCF7",
   "is_gold" = TRUE),
   poscon = "omit")
colnames(sl[,1:5])
rownames(sl[1:5,1:5])
ids.ctrl <- controls(sl, ids = amox_gold)$distil_id
```

`diffexp` *diffexp*

Description

Calculate differential expression scores, subsetting by plate.

Usage

```
diffexp(x, treat, control = "auto", method = "cd",
  split_by_plate = FALSE, where_clause = list(), gold = TRUE,
  inferred = TRUE, verbose = FALSE, ...)

## S4 method for signature 'Slinky'
diffexp(x, treat, control = "auto", method = "cd",
  split_by_plate = FALSE, where_clause = list(), gold = TRUE,
  inferred = TRUE, verbose = FALSE, ...)
```

Arguments

`x`	An object of class Slinky
`treat`	A SummarizedExperiment containing the treated samples, or the pert_iname of desired perturbagen. See details.
`control`	A SummarizedExperiment containing the control samples, or the pert_iname of desired controls. Default is 'auto'. See details.

method	Scoring method to use. Only cd and ks are presently supported.
split_by_plate	Should the analysis be split by plate? This is one way to control for batch effects, but requires at least two treated sample and two control samples on each plate in the dataset. Default is FALSE. Not supported for method = 'ks'.
where_clause	If treat is a pert_iname, further query terms may be specified here (e.g. pert_type=\"trt_sh\").
gold	Restrict analysis to gold instances as defined by LINCS. Ignored if treat and control are SummarizedExperiments.
inferred	Should the inferred (non-landmark) genes be included in the analysis? Default is TRUE.
verbose	Do you want to know how things are going? Default is FALSE.
...	Additional arguments for method.

Value

Vectors of scores, one per subset (plate). This function looks for rna_plate in colData(treat) and colData(control) to slice the data into subsets, and then performs differential expression analysis on the subsets. If a perturbation identifier is provided instead of an SummarizedExperiment, the necessary SummarizedExperiment is constructed by calling this package's toSummarizedExperiment function (which requires that you have initialized this class with appropriate clue.io key and location of gctx file). Note that the control dataset can be automatically generated by the default option of control=\"auto\". In this case, appropriate same-plate controls are identified for the samples in the treat dataset and loaded. For more complex queries, you can create the requisite SummarizedExperiments yourself with toSummarizedExperiment, or create a SummarizedExperiment by any other methods, ensuring that treat and control contain the rna_plate metadata variable for subsetting. Note that this function assumes that each plate represented in treat is also represented in control

Examples

```
#'
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
scores <- diffexp(sl, sl[,1:5], sl[,18:22])
head(scores)

# for build/demo only.  You MUST use your own key when using the slinky
# package.
## Not run:
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
cd_vector <- diffexp(sl,
```

```
                    treat = "amoxicillin",
                    split_by_plate = FALSE,
                    verbose = FALSE)

## End(Not run)
```

---

download *download*

Description

Convenience function to retrieve LINCS L1000 data and metadata files.

Usage

```
download(x, type = c("expression", "info", "readme"), level = 3,
  phase = 1, prompt = FALSE, verbose = FALSE)

## S4 method for signature 'Slinky'
download(x, type = c("expression", "info", "readme"),
  level = 3, phase = 1, prompt = FALSE, verbose = FALSE)
```

Arguments

x	A Slinky object
type	Type of file to retrieve: expression, info (instance level), or readme file.
level	Level of data desired (if type is expression): 2, 3 (default), 4, or 5.
phase	What phase of data is desired, 1 of 2? Currently only 1 is supported. Phase 2 files may be used with this package, but must be downloaded manually. Phase 2 support planned for next update.
prompt	Warn before downloading huge files? Default is FALSE.
verbose	Do you want to know how things are going? Default is FALSE.

Value

None

Note

Most of these files are very large and may take many minutes to several hours to download. A fast and reliable connection is highly recommended.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
sl <- new("Slinky")
## Not run:
download(sl, type = "info")

## End(Not run)
```

get_metadata *get_metadata*

Description

The accessor function retrieves metadata from Slinky object.

Usage

```
get_metadata(x)

## S4 method for signature 'Slinky'
get_metadata(x)
```

Arguments

x	a Slinky object

Details

As it turns out ‘metadata‘ was a poor choice for the accessor function because it can be masked if the user loads the ‘SummarizedExperiment‘ package after slinky. So this provides an alternative. Eventually ‘slinky::metadata‘ should be deprecated.

Value

The accessor function returns a `data.frame` containing the metadata.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
md <- get_metadata(sl[, 1:10])
```

ks *ks*

Description

Function to calculate KS based enrichment score based on robust z-scores

Usage

```
ks(x, data)
```

Arguments

x	A slinky object
data	a matrix of scores with genes as rows. Typically normalized, e.g. robust z scores (see rzs)

Value

Vector of KS scores for each gene. The the original CMAP paper by Lamb et al. used a KS random walk based statistic for calculating enrichment. Here we use the same metric to identify differentially expressed genes. The question arises how to summarize accross replicates. We use a \"rank of ranks\" approach. With this method, each sample is ranked, and then the entire matrix is ranked to create a single vector. The KS statistic is then calculated for each gene based on the position of their replicate values in the vectorized matrix. In this way, genes that consistently have high ranks within each sample will have a higher KS score than those that are inconsistently ranked.

See Also

rzs

---

loadL1K *loadL1K*

Description

Convert data from gctx file to SummarizedExperiment, pulling metadata from various sources. Specifying the subset of data you want can be done is various ways. The simplest example is with explicit subsetting (e.g. `data <-loadL1K(sl[1:978,1:10])`, which is equivalent to `data <-as(sl[1:978,1:10],"SummarizedExperiment")`). However, more sophisticated data loading can be achieved, for example by specifing a specific perturbagen (`pert`), or even an explicit `where_clause` which will be passed directly to the clue.io API. Thus, this function supports users with varying degrees of familiarity with the structure and content of the L1000 metadata. All arguments are optional.

Usage

```
loadL1K(x, ids = character(), pert = character(),
  cell_line = character(), type = NULL, where_clause = character(),
  gold = TRUE, inferred = TRUE, fields = character(),
  controls = FALSE, cl = NULL, verbose = FALSE)

## S4 method for signature 'Slinky'
loadL1K(x, ids = character(), pert = character(),
  cell_line = character(), type = NULL, where_clause = character(),
  gold = FALSE, inferred = TRUE, fields = character(),
  controls = FALSE, cl = NULL, verbose = FALSE)
```

Arguments

x	A slinky object
ids	distil_ids to include in the Expression Set.
pert	name (pert_iname) of perturbation for which data is desired. Supercedes value for pert_iname specified in where_clause (if any).
cell_line	name (cell_id) of cell line for which data is desired. Supercedes value for cell_id specified in where_clause (if any).
type	Optional type (pert_type) of perturbation for which data is desired, should be one of c("trt_cp","trt_sh","trt_oe") or NULL (default)
where_clause	Rather than specifying above terms, an explicit where_clause may be provided to identify the data to be loaded from the gctx file. This will be passed directly the the /pert endpoint of the clue.io API, and full documentation of the query options can be reviewed at the above link.
gold	Should we limit to instances classified as "gold" by the L1000 project (by virtue of their low inter-replicate variability)? Default is TRUE.
inferred	Should the inferred (non-landmark) genes be included in the analysis? Default is TRUE. Ignored if index is specified.
fields	Fields to include in the expression set's phenodata. Default is all available.
controls	Should same-plate controls be identified and included? Default is FALSE.
cl	Optional cluster object to speed up data retrieval from clue.io. Please use caution...a large cluster might produce requests to the API at an obnoxious rate.
verbose	Do you want to know how things are going? Default is FALSE

Value

Object of type SummarizedExperiment containing expression and meta data. #' @name loadL1K

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
amox_gold <- clueInstances(sl, where_clause = list("pert_type" = "trt_cp",
                                                   "pert_iname" = "amoxicillin",
                                                   "cell_id" = "MCF7",
                                                   "is_gold" = TRUE),
                           poscon = "omit")
ids.ctrl <- controls(sl, ids = amox_gold)$distil_id
amox_and_control <- loadL1K(sl, ids = c(amox_gold,
                                        ids.ctrl))
str(SummarizedExperiment::assays(amox_and_control)[[1]])
```

metadata *metadata*

Description

The accessor function retrieves metadata from Slinky object.

Usage

```
metadata(x)

## S4 method for signature 'Slinky'
metadata(x)
```

Arguments

x	a Slinky object

Value

The accessor function returns a `data.frame` containing the metadata.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
md <- metadata(sl[, 1:10])
```

nrow *Slinky object dimensions*

Description

Get the number of rows and columns in L1000 data represented by Slinky object.

Usage

```
nrow(x)

## S4 method for signature 'Slinky'
nrow(x)

ncol(x)
```

```
## S4 method for signature 'Slinky'
ncol(x)
```

Arguments

x	an object of class Slinky

Value

number of rows or columns of current (possibly subsetted) L1000 data set.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
ncol(sl)
nrow(sl)
```

---

readGCTX *readGCTX*

Description

Read portions of data matrix from LINCS gctx datafile

Usage

```
readGCTX(x)

## S4 method for signature 'Slinky'
readGCTX(x)
```

Arguments

x	a Slinky Object

Value

Matrix of expression data with rownames and colnames appropriately set. If a subset of the data is desired, subset the slinky object itself, not the resulting data matrix. That is, `data <-readGCTX(x[1:50,1:500])` will be MUCH faster than `ndata <-readGCTX(x)[1:50,1:500]`.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
data <- readGCTX(sl[1:20,1:5])
```

`rownames` *rownames*

Description

Retrieve row names from LINCS gctx datafile

Usage

```
rownames(x, do.NULL = TRUE, prefix = "row")

## S4 method for signature 'Slinky'
rownames(x)
```

Arguments

`x`	a Slinky Object
`do.NULL`	Ignored (see `?base::rownames`)
`prefix`	Ignored (see `?base::rownames`)

Value

Names of rows from gctx file The gctx file is an HDF5 formatted file with several sections (groups) containing the column and row level metadata as well as the expression data itself. Note that for best performance, if a subset of rownames is desired, subset the slinky object itself, not the rownames, to avoid loading the entire set of rownames from the the gctx file. That is, `names <-rownames(x[,1:50])` will be faster than `names <-rownames(x)[1:50]`. The `do.NULL` and `prefix` arguments from `base::rownames` do not apply here (as the slinky object will always have row names), and will be silently ignored if provided.

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
```

```
                    system.file('extdata', 'demo_inst_info.txt',
                        package = 'slinky'))
rownames(sl[1:5,])
```

---

rzs *rzs*

Description

Convert each sample in `treat` to robust zscore.

Usage

```
rzs(x, treat, control = "auto", where_clause = list(), gold = TRUE,
  inferred = TRUE, byplate = TRUE, verbose = FALSE, ...)

## S4 method for signature 'Slinky'
rzs(x, treat, control = "auto",
  where_clause = list(), gold = TRUE, inferred = TRUE,
  byplate = TRUE, verbose = FALSE, ...)
```

Arguments

`x`	An object of class Slinky
`treat`	A SummarizedExperiment containing the treated samples, or the pert_iname of desired perturbagen. See details.
`control`	An SummarizedExperiment containing the control samples, or the pert_iname of desired controls. Default is 'auto'. See details.
`where_clause`	If treat is a pert_iname, further query terms may be specified here (e.g. `pert_type=\"trt_sh\"`).
`gold`	Restrict analysis to gold instances as defined by LINCS. Ignored if treat and control are SummarizedExperiments.
`inferred`	Should the inferred (non-landmark) genes be included in the analysis? Default is TRUE.
`byplate`	Do you want to split the scores by plate? This is usually wise, unless you have already subsetted `treat` and `control` samples in such a way that plate can safely be ignored, or if `treat` and `control` must come from different plates for some reason. Default is TRUE.
`verbose`	Do you want to know how things are going? Default is FALSE.
`...`	Additional arguments for `method`.

Value

Matrix of zscore of same dimension as `treat` (or the expression matrix resulting from querying for `treat` if a pert_iname is specified). This function identifes same-plate controls for each treated sample, then converts each treated sample to robust z-score by subtracting the median control values and dividing by the (scaled) median absolute deviations.

Examples

```
#'
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
scores <- rzs(sl, "amoxicillin")
head(scores)
```

`Slinky-class` *An S4 class for working with LINCS data*

Description

The `Slinky` class encapsulates details about location of the LINCS L1000 data files as well as access credentials for the clue.io API (if desired). It provides methods for querying and loading data from these resources. The helper function `Slinky` is a simpler way to construct an object of this class

Usage

```
Slinky(user_key = character(), gctx = character(),
  info = character())
```

Arguments

`user_key`	clue.io API key
`gctx`	gctx containing expression data (optional)
`info`	info file containing metadata (optional)

Value

A Slinky object.

Slots

`.index` internal slot for mapping object to file data

`base` Base url for clue.io API.

`gctx` gctx containing expression data (optional)

`info` info file containing metadata (optional)

`metadata` internal slot for storing metadata from info file, mapped to gctx file and current index.

`user_key` clue.io API key (required unless CLUE_API_KEY env variable is set)

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                 system.file('extdata', 'demo.gctx',
                      package='slinky'),
                 system.file('extdata', 'demo_inst_info.txt',
                     package = 'slinky'))
amox_gold <- clueInstances(sl, where_clause = list('pert_type' = 'trt_cp',
                 'pert_iname' = 'amoxicillin',
                 'cell_id' = 'MCF7',
                 'is_gold' = TRUE), poscon = 'omit')
amox_gold_sumexp <- loadL1K(sl, ids = amox_gold)
```

[,Slinky,numeric,numeric,ANY-method

Subsetting Slinky objects

Description

Subsetting Slinky objects

Usage

```
## S4 method for signature 'Slinky,numeric,numeric,ANY'
x[i, j]

## S4 method for signature 'Slinky,missing,numeric,ANY'
x[i, j]

## S4 method for signature 'Slinky,missing,missing,ANY'
x[i, j]

## S4 method for signature 'Slinky,numeric,missing,ANY'
x[i, j]
```

Arguments

x	A Slinky Object
i	row index
j	column indes Subsets a Slinky object. This does not touch the data on file, it simply adjusts the index slots in the resulting Slinky object to speed up subsequent data operations.

Value

The subsetted Slinky object

Examples

```
# for build/demo only.  You MUST use your own key when using the slinky
# package.
user_key <- httr::content(httr::GET('https://api.clue.io/temp_api_key'),
                          as='parsed')$user_key
sl <- Slinky(user_key,
                  system.file('extdata', 'demo.gctx',
                       package='slinky'),
                  system.file('extdata', 'demo_inst_info.txt',
                      package = 'slinky'))
colnames(sl[,1:5])
rownames(sl[1:5,1:5])
```

Index