Package 'HiCBricks'

October 16, 2019

Title Framework for Storing and Accessing Hi-C Data Through HDF Files

Version 1.2.0

Description A flexible framework for storing and accessing high-resolution Hi-C data through HDF files. HiCBricks allows import of Hi-C data through various formats such as the 2D matrix format or a generalized n-column table formats. In terms of access, HiCBricks offers functions to retrieve values from genomic loci separated by a certain distance, or the ability to fetch matrix subsets using word alike terms. HiCBricks will at a later point offer the ability to fetch multiple matrix subsets using fewer calls. It offers the capacity to store GenomicRanges that may be associated to a particular Hi-C experiment, to do basic ranges overlap (any, within) with the Hi-C experiment associated Ranges object and also to store any metadata that users may think to be relevant for their Hi-C experiment. Finally, you can do TAD calls with LSD and create pretty heatmaps.

Date 2019-02-03

Type Package

Maintainer Koustav Pal <koustav.pal@ifom.eu>

License MIT + file LICENSE

Depends R (>= 3.5), utils, curl, rhdf5, R6, grid

Imports ggplot2, viridis, RColorBrewer, scales, reshape2, stringr, data.table, GenomeInfoDb, GenomicRanges, stats, IRanges, grDevices, S4Vectors, digest

Suggests BiocStyle, knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 6.1.1

biocViews DataImport, Infrastructure, Software, Technology, Sequencing, HiC

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

git_branch RELEASE_3_9

git_last_commit e2a588c

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

Author Koustav Pal [aut, cre],
Carmen Livi [ctb],
Ilario Tagliaferri [ctb]

R topics documented:

Brick_add_ranges . . . 2
Brick_fetch_range_index . . . 4
Brick_fetch_row_vector . . . 5
Brick_get_bintable . . . 6
Brick_get_chrominfo . . . 7
Brick_get_matrix . . . 7
Brick_get_matrix_mcols . . . 8
Brick_get_matrix_within_coords . . . 9
Brick_get_ranges . . . 10
Brick_get_values_by_distance . . . 11
Brick_get_vector_values . . . 12
Brick_list_matrices . . . 13
Brick_list_matrix_mcols . . . 14
Brick_list_mcool_normalisations . . . 14
Brick_list_mcool_resolutions . . . 15
Brick_list_rangekeys . . . 15
Brick_list_ranges_mcols . . . 16
Brick_load_cis_matrix_till_distance . . . 16
Brick_load_data_from_mcool . . . 18
Brick_load_matrix . . . 19
Brick_local_score_differentiator . . . 20
Brick_make_ranges . . . 22
Brick_matrix_dimensions . . . 23
Brick_matrix_exists . . . 24
Brick_matrix_filename . . . 25
Brick_matrix_isdone . . . 25
Brick_matrix_issparse . . . 26
Brick_matrix_maxdist . . . 27
Brick_matrix_minmax . . . 27
Brick_mcool_normalisation_exists . . . 28
Brick_rangekey_exists . . . 29
Brick_return_region_position . . . 30
Brick_vizart_plot_heatmap . . . 31
CreateBrick . . . 34
CreateBrick_from_mcool . . . 36
HiCBricks . . . 38

Index **40**

`Brick_add_ranges` *Store a ranges object in the Brick store.*

Description

`Brick_add_ranges` loads a GRanges object into the Brick store.

Usage

```
Brick_add_ranges(Brick, ranges, rangekey, remove.existing = TRUE)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
ranges	**Required**. An object of class ranges specifying the ranges to store in the Brick.
rangekey	**Required**. The name to use for the ranges within the Brick store.
remove.existing	**Optional**. TRUE Will remove an existing Ranges by the same name and introduce the new one.

Details

With this function it is possible to associate other ranges objects with the Brick store. If metadata columns are present, the are also loaded into the Brick store. Although not explicitly asked for, the metadata columns should not be of type list as this may create complications down the line. We ask for ranges objects, so if the same ranges object is later retrieved two additional columns will be present. These are the strand and width columns that are obtained when a ranges is converted into a data.frame. Users can ignore these columns.

Value

Returns TRUE if completed successfully.

Examples

```
Bintable.path <- system.file("extdata",
"Bintable_40kb.txt", package = "HiCBricks")
Chromosomes <- "chr19"
Path_to_cached_file <- CreateBrick(ChromNames = Chromosomes,
BinTable = Bintable.path, bin.delim = " ",
Output.Filename = file.path(tempdir(),"test.hdf"), exec = "cat",
remove.existing = TRUE)

Chrom <- c("chrS","chrS","chrS","chrS","chrS")
Start <- c(10000,20000,40000,50000,60000)
End <- c(10001,20001,40001,50001,60001)
Test_ranges <- Brick_make_ranges(Chrom = Chrom, Start = Start, End = End)
Brick_add_ranges(Brick = Path_to_cached_file, ranges = Test_ranges,
rangekey = "test_ranges")

## Not run:
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Chrom <- c("chrS","chrS","chrS","chrS","chrS")
Start <- c(10000,20000,40000,50000,60000)
End <- c(10001,20001,40001,50001,60001)
Test_ranges <- Brick_make_ranges(Chrom = Chrom, Start = Start, End = End)
Brick_add_ranges(Brick = Brick.file, ranges = Test_ranges,
rangekey = "test_ranges", remove.existing = TRUE)


## End(Not run)
```

Brick_fetch_range_index *Returns the position of the supplied ranges in the binning table associated to the Hi-C experiment.*

Description

Brick_fetch_range_index constructs a ranges object using Brick_make_ranges, creates an overlap operation using GenomicRanges::findOverlaps, where the constructed ranges is the *subject* and the Hi-C experiment associated binning table is the *query*. The return of this object is a list of ranges with their corresponding indices in the binning table.

Usage

```
Brick_fetch_range_index(Brick, chr, start, end, names = NULL,
    type = "any")
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr	**Required**. A character vector of length N specifying the chromosomes to select from the ranges.
start	**Required**. A numeric vector of length N specifying the start positions in the chromosome
end	**Required**. A numeric vector of length N specifying the end positions in the chromosome
names	**Optional**. A character vector of length N specifying the names of the chromosomes. If absent, names will take the form chr:start:end.
type	**Optional**. Default any Type of overlap operation to do. It should be one of two, any or within. any considers any overlap (atleast 1 bp) between the provided ranges and the binning table.

Value

Returns a GenomicRanges object of same length as the chr, start, end vectors provided. The object is returned with an additional column, Indexes. Indexes is a column of class IRanges::IntegerList, which is part of the larger IRanges::AtomicList superset. This "Indexes" column can be accessed like a normal GRanges column with the additional list accessor [[]] in place of the normal vector accessor [].

Examples

```
Chrom <- c("chr19","chr19")
Start <- c(1,40000)
End <- c(1000000,2000000)
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Test_Run <- Brick_fetch_range_index(Brick = Brick.file, chr = Chrom,
start = Start, end = End)
Test_Run$Indexes[[1]]
```

Brick_fetch_row_vector *Return row or col vectors.*

Description

Brick_fetch_row_vector will fetch any given rows from a matrix. If required, the rows can be subsetted on the columns and transformations applied. Vice versa is also true, wherein columns can be retrieved and rows subsetted.

Usage

```
Brick_fetch_row_vector(Brick, chr1, chr2, by = c("position", "ranges"),
    vector, regions = NULL, force = FALSE, flip = FALSE, FUN = NULL)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix
by	**Required**. One of two possible values, "position" or "ranges". A one-dimensional numeric vector of length 1 specifying one of either position or ranges.
vector	**Required**. If by is position, a 1 dimensional numeric vector containing the rows to be extracted is expected. If by is ranges, a 1 dimensional character vector containing the names of the bintable is expected. This function does not do overlaps. Rather it returns any given row or column based on their position or names in the bintable.
regions	**Optional**. Default NULL A character vector of length vector is expected. Each element must be of the form chr:start:end. These regions will be converted back to their original positions and the corresponding rows will be subsetted by the corresponding region element. If the length of regions does not match, the subset operation will not be done and all elements from the rows will be returned.
force	**Optional**. Default FALSE If true, will force the retrieval operation when matrix contains loaded data until a certain distance.
flip	**Optional**. Default FALSE If present, will flip everything. This is equivalent to selecting columns, and subsetting on the rows.
FUN	**Optional**. Default NULL If provided a data transformation with FUN will be applied before the matrix is returned.

Value

Returns a list of length vector. Each list element will be of length chr2 binned length or if regions is present the corresponding region length. This may differ based on the operations with FUN.

See Also

Brick_get_matrix_within_coords to get matrix by using matrix genomic coordinates, Brick_get_values_by_dista to get values separated at a certain distance, Brick_fetch_row_vector to get values in a certain row/col and subset them, Brick_get_matrix to get matrix by using matrix coordinates.

Examples

```
Coordinate <- c("chr19:1:40000","chr19:40001:80000")
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Test_Run <- Brick_fetch_row_vector(Brick = Brick.file,
chr1 = "chr19", chr2 = "chr19", by = "ranges", vector = Coordinate,
regions = c("chr19:1:1000000", "chr19:40001:2000000"))
```

Brick_get_bintable *Returns the binning table associated to the Hi-C experiment.*

Description

Brick_get_bintable makes a call to Brick_get_ranges to retrieve the binning table of the associated Brick store. This is equivalent to passing the argument rangekey = "bintable" in Brick_get_ranges

Usage

```
Brick_get_bintable(Brick, chr = NULL)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr	**Optional**. A chr string specifying the chromosome to select from the ranges.

Value

Returns a GRanges object containing the binning table associated to the Brick store.

See Also

Brick_get_ranges

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_bintable(Brick = Brick.file)
```

Brick_get_chrominfo *Get the chrominfo for the Hi-C experiment.*

Description

Brick_get_chrominfo fetches the associated chrominfo table for the Brick it is associated to.

Usage

```
Brick_get_chrominfo(Brick)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.

Value

A three column data.frame containing chromosomes, nrows and length.

chromosomes corresponds to all chromosomes in the provided bintable.

nrows corresponds to the number of entries in the bintable or dimension for that chromosome in a Hi-C matrix.

Length is the total bp length of the same chromosome (max value for that chromosome in the bintable).

Examples

```
Brick.file = system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_chrominfo(Brick = Brick.file)
```

Brick_get_matrix *Return a matrix subset.*

Description

Brick_get_matrix will fetch a matrix subset between row values ranging from min(x.vector) to max(x.vector) and column values ranging from min(x.vector) to max(x.vector)

Usage

```
Brick_get_matrix(Brick, chr1, chr2, x.vector, y.vector, force = FALSE,
    FUN = NULL)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix
x.vector	**Required**. A one-dimensional numeric vector specifying the rows to subset.
y.vector	**Required**. A one-dimensional numeric vector specifying the columns to subset.
force	**Optional**. Default FALSE If true, will force the retrieval operation when matrix contains loaded data until a certain distance.
FUN	**Optional**. If provided a data transformation with FUN will be applied before the matrix is returned.

Value

Returns a matrix of dimension x.vector length by y.vector length. This may differ based on the operations with FUN.

See Also

Brick_get_matrix_within_coords to get matrix by using matrix genomic coordinates, Brick_get_values_by_dista to get values separated at a certain distance, Brick_fetch_row_vector to getvalues in a certain row/col and subset them, Brick_get_vector_values to get values using matrix coordinates.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_matrix(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19",
x.vector = c(1:10), y.vector = c(1:10))
```

Brick_get_matrix_mcols

Get the matrix metadata columns in the Brick store.

Description

Brick_get_matrix_mcols will get the specified matrix metadata column.

Usage

```
Brick_get_matrix_mcols(Brick, chr1, chr2, what)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix
what	**Required** A character vector of length 1 specifying the matrix metric to retrieve

Value

Returns a 1xN dimensional vector containing the specified matrix metric

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_matrix_mcols(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19",
what = "bin.coverage")
```

Brick_get_matrix_within_coords

Return a matrix subset between two regions.

Description

Brick_get_matrix_within_coords will fetch a matrix subset after creating an overlap operation between both regions and the bintable associated to the Brick store. This function calls Brick_get_matrix.

Usage

```
Brick_get_matrix_within_coords(Brick, x.coords, y.coords, force = FALSE,
    FUN = NULL)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
x.coords	**Required**. A string specifying the region to subset on the rows. It takes the form chr:start:end. An overlap operation with the associated bintable will be done to identify the bins to subset on the row
y.coords	**Required**. A string specifying the region to subset on the rows. It takes the form chr:start:end. An overlap operation with the associated bintable will be done to identify the bins to subset on the column
force	**Optional**. Default FALSE If true, will force the retrieval operation when matrix contains loaded data until a certain distance.
FUN	**Optional**. If provided a data transformation with FUN will be applied before the matrix is returned.

Value

Returns a matrix of dimension x.coords binned length by y.coords binned length. This may differ based on FUN.

See Also

Brick_get_matrix to get matrix by using matrix coordinates, Brick_get_values_by_distance to get values separated at a certain distance, Brick_fetch_row_vector to get values in a certain row/col and subset them, Brick_get_vector_values to get values using matrix coordinates.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_matrix_within_coords(Brick = Brick.file,
x.coords = "chr19:40000:2000000",
y.coords = "chr19:40000:2000000")

Brick_get_matrix_within_coords(Brick = Brick.file,
x.coords = "chr19:40000:2000000",
y.coords = "chr19:40000:2000000",
FUN = mean)

Brick_get_matrix_within_coords(Brick = Brick.file,
x.coords = "chr19:40000:2000000",
y.coords = "chr19:40000:2000000",
FUN = median)
```

Brick_get_ranges *Fetch the ranges associated to a rangekey or chromosome.*

Description

Brick_get_ranges will get a ranges object if present in the Brick store and return a GRanges object.

Usage

```
Brick_get_ranges(Brick, chr = NULL, rangekey)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with Create-Brick.
chr	**Optional**. A chr string specifying the chromosome to select from the ranges.
rangekey	**Required**. A string specifying the name of the ranges.

Details

If a rangekey is present, the ranges will be retrieve and a GRanges constructed. Metadata columns will also be added. If these are rangekeys other than "Bintable", and had been added using Brick_add_ranges the width and Strand columns may appear as metadata columns. These will most likely be artifacts from converting the original ranges object to a data.frame.

Value

Returns a GRanges object with the associated metadata columns that may have been present in the Ranges object.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_ranges(Brick = Brick.file, chr = "chr19", rangekey = "Bintable")
```

Brick_get_values_by_distance

Return values separated by a certain distance.

Description

Brick_get_values_by_distance can fetch values with or without transformation or subsetted by a certain distance. Please note, this module is not an iterable module.

Usage

```
Brick_get_values_by_distance(Brick, chr, distance,
    constrain.region = NULL, batch.size = 500, FUN = NULL)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with Create-Brick.
chr	**Required**. A string specifying the chromosome for the cis Hi-C matrix from which values will be retrieved at a certain distance.
distance	**Required**. 0 based. Fetch values separated by distance.
constrain.region	**Optional**. A character vector of length 1 with the form chr:start:end specifying the region for which the distance values must be retrieved.
batch.size	**Optional**. Default 500 A numeric vector of length 1 specifying the size of the chunk to retrieve for diagonal selection.
FUN	**Optional**. If provided a data transformation with FUN will be applied before values are returned.

Value

Returns a numeric vector of length N depending on the presence of constrain.region, FUN and distance from the main diagonal.

See Also

Brick_get_matrix_within_coords to get matrix by using matrix coordinates, Brick_fetch_row_vector to get values in a certain row/col and subset them, Brick_get_vector_values to get values using matrix coordinates, Brick_get_matrix to get matrix by using matrix coordinates.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_values_by_distance(Brick = Brick.file, chr = "chr19",
distance = 0)

Failsafe_median <- function(x){
     x[is.nan(x) | is.infinite(x) | is.na(x)] <- 0
     return(median(x))
}

Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_values_by_distance(Brick = Brick.file, chr = "chr19",
distance = 4, FUN = Failsafe_median)
```

Brick_get_vector_values

Return a N dimensional vector selection.

Description

`Brick_get_vector_values` is the base function being used by all other matrix retrieval functions.

Usage

```
Brick_get_vector_values(Brick, chr1, chr2, xaxis, yaxis, FUN = NULL,
    force = FALSE)
```

Arguments

`Brick`	**Required**. A string specifying the path to the Brick store created with CreateBrick.
`chr1`	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
`chr2`	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix
`xaxis`	**Required**. A 1 dimensional vector containing the rows to retrieve. Gaps in this vector may result in unexpected behaviour as the values which are considered are min(xaxis) and max(xaxis) for retrieval.
`yaxis`	**Required**. A 1 dimensional vector containing the columns to retrieve. Gaps in this vector may result in unexpected behaviour as the values which are considered are min(yaxis) and max(yaxis) for retrieval.
`FUN`	**Optional**. Default NULL If provided a data transformation with FUN will be applied before the vector is returned.
`force`	**Optional**. Default FALSE If true, will force the retrieval operation when matrix contains loaded data until a certain distance.

Value

Returns a vector of length yaxis if length of xaxis is 1. Else returns a matrix of dimension xaxis length by yaxis length.

Note

Whatever the length of xaxis or yaxis may be, the coordinates under consideration will range from min(xaxis) to max(xaxis) on the rows or min(yaxis) to max(yaxis) on the columns.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_get_vector_values(Brick = Brick.file, chr1 = "chr19",
chr2 = "chr19", xaxis = c(1:10), yaxis = c(1:10))
```

Brick_list_matrices *List the matrix pairs present in the Brick store.*

Description

Brick_list_matrices will list all chromosomal pair matrices from the Brick store, with their associated filename, value range, done status and sparse

Usage

```
Brick_list_matrices(Brick)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.

Value

Returns a data.frame object with columns chr1, chr2 corresponding to chromosome pairs, and the associated attributes. filename corresponds to the name of the file that was loaded for the pair. min and max specify the minimum and maximum values in the matrix, done is a logical value specifying if a matrix has been loaded and sparsity specifies if a matrix is defined as a sparse matrix.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_list_matrices(Brick = Brick.file)
```

Brick_list_matrix_mcols

List the matrix metadata columns in the Brick store.

Description

Brick_get_matrix_mcols will list the names of all matrix metadata columns.

Usage

```
Brick_list_matrix_mcols(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Value

Returns a vector containing the names of all matrix metadata columns

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_list_matrix_mcols(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")
```

Brick_list_mcool_normalisations

Get all available normalisations in an mcool file.

Description

Brick_list_mcool_normalisations lists the names available for accessing the various normalisation factors in an mcool file. Please note, this only lists the mapping of the columns to their respective names. It does not check for the availability of that particular column in the mcool file

Usage

```
Brick_list_mcool_normalisations(names.only = FALSE)
```

Arguments

names.only	**Optional**. Default FALSE A parameter specifying whether to list only the human readable names without their respective column names in the mcool file.

Value

A named vector listing all possible normalisation factors.

Examples

```
Brick_list_mcool_normalisations()
```

Brick_list_mcool_resolutions

Get all available normalisations in an mcool file.

Description

`Brick_list_mcool_resolutions` lists all available resolutions in the mcool file.

Usage

```
Brick_list_mcool_resolutions(mcool)
```

Arguments

`mcool`	**Required**. A parameter specifying the name of an mcool file

Value

A named vector listing all possible resolutions in the file.

Brick_list_rangekeys

List the ranges tables stored within the Brick.

Description

`Brick_list_rangekeys` lists the names of all ranges associated to a Brick.

Usage

```
Brick_list_rangekeys(Brick)
```

Arguments

`Brick`	**Required**. A string specifying the path to the Brick store created with CreateBrick.

Value

A one dimensional character vector of length x specifying the names of all ranges currently present in the file.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_list_rangekeys(Brick = Brick.file)
```

Brick_list_ranges_mcols

Find out what metadata columns are associated to a ranges with a certain name

---

Description

`Brick_list_ranges_mcols` will list the metadata columns of the specified ranges if it is present in the Brick store.

Usage

```
Brick_list_ranges_mcols(Brick, rangekey = NULL)
```

Arguments

`Brick`	**Required**. A string specifying the path to the Brick store created with CreateBrick.
`rangekey`	**Optional**. A string specifying the name of the ranges. If not present, the metadata columns of all ranges will be listed.

Value

if no metadata columns are present, NA. If metadata columns are present, a data.frame object containing the name of the ranges and the associated metadata column name.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_list_ranges_mcols(Brick = Brick.file, rangekey = "test_ranges")
```

Brick_load_cis_matrix_till_distance

Load a NxN dimensional sub-distance cis *matrix into the Brick store.*

Description

Load a NxN dimensional sub-distance *cis* matrix into the Brick store.

Usage

```
Brick_load_cis_matrix_till_distance(Brick, chr, matrix.file, delim = " ",
    distance, remove.prior = FALSE, num.rows = 2000, is.sparse = FALSE,
    sparsity.bins = 100)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows and cols of the matrix
matrix.file	**Required**. A character vector of length 1 specifying the name of the file to load as a matrix into the Brick store.
delim	**Optional**. Default " " The delimiter of the matrix file.
distance	**Required**. Default NULL. For very high-resolution matrices, read times can become extremely slow and it does not make sense to load the entire matrix into the data structure, as after a certain distance, the matrix will become extremely sparse. This ensures that only interactions upto a certain distance from the main diagonal will be loaded into the data structure.
remove.prior	**Optional**. Default FALSE If a matrix was loaded before, it will not be replaced. Use remove.prior to override and replace the existing matrix.
num.rows	**Optional**. Default 2000 Number of rows to insert per write operation in the HDF file.
is.sparse	**Optional**. Default FALSE If true, designates the matrix as being a sparse matrix, and computes the sparsity.index. The sparsity index measures the proportion of non-zero rows or columns at a certain distance from the diagonal (100) in cis interaction matrices.
sparsity.bins	**Optional**. Default 100 With regards to computing the sparsity.index, this parameter decides the number of bins to scan from the diagonal.

Value

Returns TRUE if all went well.

Examples

```
Bintable.path <- system.file("extdata",
"Bintable_40kb.txt", package = "HiCBricks")
Chromosomes <- "chr19"
Path_to_cached_file <- CreateBrick(ChromNames = Chromosomes,
BinTable = Bintable.path, bin.delim = " ",
Output.Filename = file.path(tempdir(),"test.hdf"), exec = "cat",
remove.existing = TRUE)

Test.mat <- matrix(runif(800*800),nrow = 800, ncol = 800)
Matrix.file <- file.path(tempdir(),"Test_matrix.txt")
write.table(x = Test.mat, file = Matrix.file, sep = " ", quote = FALSE,
row.names = FALSE, col.names = FALSE)
Brick_load_cis_matrix_till_distance(Brick = Path_to_cached_file,
chr = "chr19", matrix.file = Matrix.file, delim = " ",
distance = 200, remove.prior = TRUE)
```

`Brick_load_data_from_mcool` *Load a NxN dimensional matrix into the Brick store from an mcool file.*

Description

Read an mcool contact matrix coming out of 4D nucleome projects into a Brick store.

Usage

```
Brick_load_data_from_mcool(Brick, mcool, chr1, chr2, binsize = NULL,
    cooler.batch.size = 1e+06, matrix.chunk = 2000,
    dont.look.for.chr2 = FALSE, remove.prior = FALSE,
    norm.factor = "Iterative-Correction")
```

Arguments

`Brick`	**Required**. A string specifying the path to the Brick store created with CreateBrick.
`mcool`	**Required**. Path to an mcool file.
`chr1`	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
`chr2`	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix
`binsize`	**Optional**. The binsize to select from an mcool file.
`cooler.batch.size`	**Optional**. Default 1000000. The number of values to read per iteration through a mcool file.
`matrix.chunk`	**Optional**. Default 2000. The nxn matrix square to fill per iteration in a mcool file.
`dont.look.for.chr2`	**Required**. At startup, the function will attempt to search for the first occurence of a chr2 contact value. This is done to avoid the reading of all chr1 values for every chunk processed. If chr1 and chr2 are equivalent, consider setting it to FALSE.
`remove.prior`	**Optional**. Default FALSE If a matrix was loaded before, it will not be replaced. Use remove.prior to override and replace the existing matrix.
`norm.factor`	**Optional**. Default "Iterative-Correction". The normalization factor to use for normalization from an mcool file. norm.factor currently accepts one of "Iterative-Correction", "Knight-Ruitz", "Vanilla-coverage", "Vanilla-coverage-square-root" and NULL. If NULL, the function will load only counts from the mcool file.

Value

Returns TRUE if all went well.

See Also

CreateBrick_from_mcool to create matrix from an mcool file, Brick_list_mcool_resolutions to list available resolutions in an mcool file, Brick_list_mcool_normalisations to list available normalisation factors in the mcool file.

Examples

```
## Not run:

require(curl)
curl_download(url = paste("https://data.4dnucleome.org/"
"files-processed/4DNFI7JNCNFB/"
"@download/4DNFI7JNCNFB.mcool",sep = ""),
destfile = file.path(tempdir(),"H1-hESC-HiC-4DNFI7JNCNFB.mcool"))

Output.brick <- file.path(tempdir(),
 "H1-hESC-HiC-4DNFI7JNCNFB-10000-ICE-normalised-chr1.brick")
mcool <- file.path(tempdir(),"H1-hESC-HiC-4DNFI7JNCNFB.mcool")

CreateBrick_from_mcool(Brick = Output.brick,
mcool = mcool,
binsize = 10000,
chrs = "chr1")

Brick_load_data_from_mcool(Brick = Output.brick, mcool = mcool,
chr1 = "chr1", chr2 = "chr1", binsize = 10000,
cooler.batch.size = 1000000, matrix.chunk = 2000,
dont.look.for.chr2 = TRUE, remove.prior = TRUE,
norm.factor = "Iterative-Correction")


## End(Not run)
```

Brick_load_matrix *Load a NxM dimensional matrix into the Brick store.*

Description

Load a NxM dimensional matrix into the Brick store.

Usage

```
Brick_load_matrix(Brick, chr1, chr2, matrix.file, delim = " ", exec,
    remove.prior = FALSE, num.rows = 2000, is.sparse = FALSE,
    sparsity.bins = 100)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with Create-Brick.

chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix
matrix.file	**Required**. A character vector of length 1 specifying the name of the file to load as a matrix into the Brick store.
delim	**Optional**. Default " " The delimiter of the matrix file.
exec	**Required**. A string specifying the program to use for reading the file. Use cat for txt files, for bz2 files use bzcat and for gz files zcat.
remove.prior	**Optional**. Default FALSE If a matrix was loaded before, it will not be replaced. Use remove.prior to override and replace the existing matrix.
num.rows	**Optional**. Default 2000 Number of rows to read, in each chunk.
is.sparse	**Optional**. Default FALSE If true, designates the matrix as being a sparse matrix, and computes the sparsity.index. The sparsity index measures the proportion of non-zero rows or columns at a certain distance from the diagonal (100) in cis interaction matrices.
sparsity.bins	**Optional**. Default 100 With regards to computing the sparsity.index, this parameter decides the number of bins to scan from the diagonal.

Value

Returns TRUE if all went well.

Examples

```
Bintable.path <- system.file("extdata",
"Bintable_40kb.txt", package = "HiCBricks")
Chromosomes <- "chr19"
Path_to_cached_file <- CreateBrick(ChromNames = Chromosomes,
BinTable = Bintable.path, bin.delim = " ",
Output.Filename = file.path(tempdir(),"test.hdf"), exec = "cat",
remove.existing = TRUE)

Test.mat <- matrix(runif(800*800),nrow = 800, ncol = 800)
Matrix.file <- file.path(tempdir(),"Test_matrix.txt")
write.table(x = Test.mat, file = Matrix.file, sep = " ", quote = FALSE,
row.names = FALSE, col.names = FALSE)
Brick_load_matrix(Brick = Path_to_cached_file, chr1 = "chr19",
chr2 = "chr19", matrix.file = Matrix.file, delim = " ", exec = "cat",
remove.prior = TRUE)
```

Brick_local_score_differentiator

Do TAD Calls with Local Score Differentiator on a Hi-C matrix

---

Description

Local_score_differentiator calls topologically associated domains on Hi-C matrices. Local score differentiator at the most fundamental level is a change point detector, which detects change points in the directionality index using various thresholds defined on a local directionality index distributions. The directionality index (DI) is calculated as defined by Dixon et al., 2012 Nature. Next, the difference of DI is calculated between neighbouring bins to get the change in DI distribution in each bin. When a DI value goes from a highly negative value to a highly positive one as expected to occur at domain boundaries, the ensuing DI difference distribution becomes a very flat distribution interjected by very large peaks signifying regions where such a change may take place. We use two difference vectors, one is the difference vector between a bin and its adjacent downstream bin and another is the difference between a bin and its adjacent upstream bin. Using these vectors, and the original directionality index, we define domain borders as outliers.

Usage

```
Brick_local_score_differentiator(Brick, chrs = NULL, min.sum = -1,
    di.window = 200L, lookup.window = 200L, tukeys.constant = 1.5,
    strict = TRUE, fill.gaps = TRUE, ignore.sparse = TRUE,
    sparsity.threshold = 0.8, remove.empty = NULL, chunk.size = 500,
    force.retrieve = TRUE)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chrs	**Optional**. Default NULL If present, only TAD calls for elements in *chrs* will be done.
min.sum	**Optional**. Default -1 Process bins in the matrix with row.sums greater than *min.sum*.
di.window	**Optional**. Default 200 Use *di.window* to define the directionality index.
lookup.window	**Optional**. Default 200 Use *lookup.window* local window to call borders. At smaller *di.window* values we recommend setting this to 2**di.window*
tukeys.constant	**Optional**. Default 1.5 *tukeys.constant**IQR (inter-quartile range) defines the lower and upper fence values.
strict	**Optional**. Default TRUE If TRUE, *strict* creates an additional filter on the directionality index requiring it to be either greater than or less than 0 on the right tail or left tail respectively.
fill.gaps	**Optional**. Default TRUE If TRUE, this will affect the TAD stiching process. All Border starts are stiched to the next downstream border ends. Therefore, at times border ends remain unassociated to a border start. These border ends are stiched to the adjacent downstream bin from their upstream border end when *fill.gaps* is true. TADs inferred in this way will be annotated with two metadata columns in the GRanges object. *gap.fill* will hold a value of 1 and *level* will hold a value 1. TADs which were not filled in will hold a gap.fill value of 0 and a level value of 2.
ignore.sparse	**Optional**. Default TRUE If TRUE, a matrix which has been defined as sparse during the matrix loading process will be treated as a dense matrix. The *sparsity.threshold* filter will not be applied. Please note, that if a matrix is defined as sparse and fill.gaps is TRUE, fill.gaps will be turned off.

`sparsity.threshold`
: **Optional**. Default 0.8 Sparsity threshold relates to the sparsity index, which is computed as the number of non-zero bins at a certain distance from the diagonal. If a matrix is sparse and ignore.sparse is FALSE, bins which have a sparsity index value below this threshold will be discarded from DI computation.

`remove.empty`
: Not implemented. After implementation, this will ensure that the presence of centromeric regions is accounted for.

`chunk.size`
: **Optional**. Default 500 The size of the matrix chunk to process. This value should be larger than 2x di.window.

`force.retrieve`
: **Optional**. Default TRUE If TRUE, this will force the retrieval of a matrix chunk even when the retrieval includes interaction points which were not loaded into a Brick store (larger chunks). Please note, that this does not mean that DI can be computed at distances larger than max distance. Rather, this is meant to aid faster computation.

Details

To define an outlier, fences are first defined. The fences are defined using tukeys.constant x inter-quartile range of the directionality index. The upper fence used for detecting domain starts is the 75th quartile + (IQR x tukeys.constant), while the lower fence is the 25th quartile - (IQR x tukeys.constant). For domain starts the DI difference must be greater than or equal to the upper fence, it must be greater than the DI and the DI must be a finite real value. If strict is TRUE, DI will also be required to be greater than 0. Similarly, for domain ends the DI difference must be lower than or equal to the lower fence, it must be lower than the DI and the DI must be a finite real value. If strict is TRUE, DI will also be required to be lower than 0.

After defining outliers, each domain start will be associated to its nearest downstream domain end. If *fill.gaps* is defined as TRUE and there are domain ends which remain unassociated to a domain start, These domain ends will be associated to the bin adjacent to their nearest upstream domain end. This associations will be marked by metadata columns, gap.fill= 1 and level = 1.

This function provides the capability to call very accurante TAD definitions in a very fast way.

Value

A ranges object containing domain definitions. The starts and ends of the ranges coincide with the starts and ends of their contained bins from the bintable.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
TAD_ranges <- Brick_local_score_differentiator(Brick = Brick.file,
chrs = "chr19", di.window = 10, lookup.window = 30, strict = TRUE,
fill.gaps = TRUE, chunk.size = 500)
```

Brick_make_ranges *Creates a ranges object from provided vectors.*

Description

`Brick_make_ranges` creates a GRanges object from the provided arguments

Usage

```
Brick_make_ranges(Chrom, Start, End, Strand = NULL, Names = NULL)
```

Arguments

Chrom	**Required**. A 1 dimensional character vector of size N specifying the chromosomes in the ranges.
Start	**Required**. A 1 dimensional numeric vector of size N specifying the start positions in the ranges.
End	**Required**. A 1 dimensional numeric vector of size N specifying the end positions in the ranges. Must be less than Start.
Strand	**Optional**. A 1 dimensional character vector of size N specifying the strand of the ranges. If not provided, this will be set to the default *.
Names	**Optional**. A 1 dimensional character vector of size N specifying the names of the ranges. If not provided, this will be set to the default chr:start:end.

Value

A GenomicRanges object with the previous sort order being preserved

Examples

```
Chrom <- c("chrS","chrS","chrS","chrS","chrS")
Start <- c(10000,20000,40000,50000,60000)
End <- c(10001,20001,40001,50001,60001)
Test_ranges <- Brick_make_ranges(Chrom = Chrom, Start = Start, End = End)
```

Brick_matrix_dimensions

Return the dimensions of a matrix

Description

Return the dimensions of a matrix

Usage

```
Brick_matrix_dimensions(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Value

Returns the dimensions of a Hi-C matrix for any given chromosome pair.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_dimensions(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")
```

Brick_matrix_exists *Check if a chromosome pair exists.*

Description

Matrices are created when the bintable is loaded and the chromosome names are provided. If a user is in doubt regarding whether a matrix is present or not it is useful to check this function. If the Bintable did not contain a particular chromosome, any matrices for that chromosome would not be present in the file

Usage

```
Brick_matrix_exists(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Value

Returns a logical vector of length 1, specifying if the matrix exists or not.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_exists(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")

Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_exists(Brick = Brick.file, chr1 = "chr19", chr2 = "chr20")
```

Brick_matrix_filename *Return the filename of the loaded matrix*

Description

Return the filename of the loaded matrix

Usage

```
Brick_matrix_filename(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Value

Returns a character vector of length 1 specifying the filename of the currently loaded matrix.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_filename(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")
```

Brick_matrix_isdone *Check if a matrix has been loaded for a chromosome pair.*

Description

Check if a matrix has been loaded for a chromosome pair.

Usage

```
Brick_matrix_isdone(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Value

Returns a logical vector of length 1, specifying if a matrix has been loaded or not.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_isdone(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")
```

Brick_matrix_issparse *Check if a matrix for a chromosome pair is sparse.*

Description

Check if a matrix for a chromosome pair is sparse.

Usage

```
Brick_matrix_issparse(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Value

Returns a logical vector of length 1, specifying if a matrix was loaded as a sparse matrix.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_issparse(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")
```

Brick_matrix_maxdist *Get the maximum loaded distance from the diagonal of any matrix.*

Description

If values beyond a certain distance were not loaded in the matrix, this distance parameter is useful. This package by default will check this param to make sure that it is not returning non-existent data.

Usage

```
Brick_matrix_maxdist(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Details

Brick_matrix_maxdist will return this parameter.

Value

Returns an integer vector of length 1, specifying the maximum distance loaded for that matrix

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_maxdist(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")
```

Brick_matrix_minmax *Return the value range of the matrix*

Description

Return the value range of the matrix

Usage

```
Brick_matrix_minmax(Brick, chr1, chr2)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
chr1	**Required**. A character vector of length 1 specifying the chromosome corresponding to the rows of the matrix
chr2	**Required**. A character vector of length 1 specifying the chromosome corresponding to the columns of the matrix

Value

Returns a numeric vector of length 2, specifying the minimum and maximum finite real values in the matrix.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_matrix_minmax(Brick = Brick.file, chr1 = "chr19", chr2 = "chr19")
```

Brick_mcool_normalisation_exists

Check if a normalisation exists in an mcool file.

Description

`Brick_mcool_normalisation_exists` checks if a particular normalisation exists in an mcool file.

Usage

```
Brick_mcool_normalisation_exists(mcool, norm.factor = NULL,
    binsize = NULL)
```

Arguments

mcool	**Required**. Path to an mcool file.
norm.factor	**Required**. The normalization factor to use for normalization from an mcool file. norm.factor currently accepts one of "Iterative-Correction", "Knight-Ruitz", "Vanilla-coverage", "Vanilla-coverage-square-root".
binsize	**Optional**. The binsize to select from an mcool file.

Value

A boolean vector of length 1

Examples

```
## Not run:

require(curl)
curl_download(url = paste("https://data.4dnucleome.org/"
"files-processed/4DNFI7JNCNFB/"
"@download/4DNFI7JNCNFB.mcool",sep = ""),
destfile = "./H1-hESC-HiC-4DNFI7JNCNFB.mcool")

mcool <- "./H1-hESC-HiC-4DNFI7JNCNFB.mcool"
Brick_mcool_normalisation_exists(mcool = mcool,
norm.factor = "Iterative-Correction",
binsize = 10000)


## End(Not run)
```

Brick_rangekey_exists *Check to see if the Brick contains a ranges with a certain name.*

Description

Brick_rangekey_exists checks for the presence of a particular ranges with a certain name.

Usage

```
Brick_rangekey_exists(Brick, rangekey)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with Create-Brick.
rangekey	**Required**. A string specifying the name of the ranges to check for.

Value

A logical vector of length 1 with either TRUE or FALSE values.

Examples

```
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_rangekey_exists(Brick = Brick.file, rangekey = "Bintable")
```

`Brick_return_region_position` *Provides the overlapping position (within) from the bintable.*

Description

`Brick_return_region_position` takes as input a human-readable coordinate format of the form chr:start:end and outputs the overlapping bintable positions. This module does a "within" operation. So only bins which overlap completely with the region will be returned. This is not an iterable module, so the user has to make iterative calls to the module itself.

Usage

```
Brick_return_region_position(Brick, region)
```

Arguments

`Brick`	**Required**. A string specifying the path to the Brick store created with CreateBrick.
`region`	**Required**. A character vector of length 1 specifying the region to overlap. It must take the form chr:start:end.

Value

Returns a 1 dimensional vector containing the position of the overlapping regions in the bintable associated the Brick store.

Design choice

This may seem to be a poor design choice at first glance, but I do not think this to be the case. By not being iterable, this function circumvents the problem of how to structure the data for the user. If one more element was accepted, the return object would have become a list, which increases the data structure complexity significantly for users who are just starting out with R. Therefore this problem is left for the users themselves to deal with.

Examples

```
Coordinate <- "chr19:1:1000000"
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Test_Run <- Brick_return_region_position(Brick = Brick.file,
region = Coordinate)

## Not run:

Coordinate <- c("chr19:1:1000000","chr19:40000:2000000")
Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Test_Run <- Brick_return_region_position(Brick = Brick.file,
region = Coordinate)

This will generate an error because the module itself expects as input a
vector of length 1.
```

```
## End(Not run)
```

`Brick_vizart_plot_heatmap` *Create the entire HDF5 structure and load the bintable*

Description

`Brick_vizart_plot_heatmap` creates various heatmaps and plots TADs.

Usage

```
Brick_vizart_plot_heatmap(File, Bricks, x.coords, y.coords, FUN = NULL,
    value.cap = NULL, distance = NULL, rotate = FALSE, x.axis = TRUE,
    x.axis.title = NULL, y.axis = TRUE, y.axis.title = NULL,
    title = NULL, legend.title = NULL, return.object = FALSE,
    x.axis.num.breaks = 5, y.axis.num.breaks = 5, palette,
    col.direction = 1, extrapolate.on = NULL, x.axis.text.size = 10,
    y.axis.text.size = 10, text.size = 10, legend.title.text.size = 8,
    legend.text.size = 8, title.size = 10, tad.ranges = NULL,
    group.col = NULL, tad.colour.col = NULL, colours = NULL,
    colours.names = NULL, cut.corners = FALSE, highlight.points = NULL,
    width = 10, height = 6, line.width = 0.5, units = "cm",
    legend.key.width = unit(3, "cm"), legend.key.height = unit(0.5,
    "cm"))
```

Arguments

`File`	**Required** A character vector containing the output filename to write.
`Bricks`	**Required** A character vector of length 1 (in case of one sample heatmaps) or 2 (in case of two sample heatmaps) specifying the names of the Brick stores from where to fetch the data.
`x.coords`	**Required** A character vector of length 1 specifying the coordinates from where to fetch the data.
`y.coords`	**Required** A character vector of length 1 specifying the coordinates from where to fetch the data.
`FUN`	**Optional**. Default NULL If any sort of transformations should be applied to the data before plotting. Such as, log10 or log2 transformations.
`value.cap`	**Optional**. Default NULL If present, values beyond a certain quantile will be capped to that quantile. In Hi-C this helps to emphasize structural information. Please note, if this parameter is present the greatest value will have a greater than sign append- -ed to them.
`distance`	**Optional**. Default NULL If present, values beyond this distance will be filtered out. Please note, that if a Brick store matrix was loaded until a certain distance, this parameter will result in an error if it is greater than the loaded distance.
`rotate`	**Optional**. Default FALSE If TRUE, will rotate the heatmap by 90 degrees.

`x.axis` **Optional**. Default TRUE If FALSE, the x-axis will be removed (ticks, x-axis labels and title).

`x.axis.title` **Optional**. Default NULL If present, will be the *x-axis* title. Else defaults to the provided x.coords

`y.axis` **Optional**. Default TRUE If FALSE, the y-axis will be removed (ticks, y-axis labels and title).

`y.axis.title` **Optional**. Default NULL If present, will be the *y-axis* title. Else defaults to the provided y.coords

`title` **Optional**. Default NULL If present, will be the *plot* title. Else defaults to the provided x.coords vs y.coords

`legend.title` **Optional**. Default NULL If present will be the title of the legend. Else defaults to "Signal".

`return.object` **Optional**. Default FALSE If present the ggplot object will be returned

`x.axis.num.breaks` **Optional**. Default 5 Number of ticks on the x axis

`y.axis.num.breaks` **Optional**. Default 5 Number of ticks on the y axis

`palette` **Required**. Default NULL One of the RColorbrewer or viridis colour palettes

`col.direction` **Optional**. Default 1 If -1, the colour scale will be reversed.

`extrapolate.on` **Optional**. Default NULL If present, colours from the palette will be extrapolated between lightest and darkest to create the gradient. This value cannot be more than 100.

`x.axis.text.size` **Optional**. Default 10 x-axis text size

`y.axis.text.size` **Optional**. Default 10 y-axis text size

`text.size` **Optional**. Default 10 text size of text elements in the plot.

`legend.title.text.size` **Optional**. Default 8 text size of the legend title

`legend.text.size` **Optional**. Default 8 text size of the legend text

`title.size` **Optional**. Default 10 text size of the title

`tad.ranges` **Optional**. Default NULL A GenomicRanges object specifying the start and end coordinates of TADs to be plotted on the heatmap.

`group.col` **Optional**. Default NULL Name of the column which will be used to categorize TADs as belonging to either the first or the second Brick stores. This must be a numeric value ranging from 1 to 2. If NULL, TADs will be plotted on both Hi-C maps.

`tad.colour.col` **Optional**. Default NULL tad.colour.col takes as value the column name in the tad.ranges object corresponding to the column which should be used to define different TAD categories.

`colours` **Optional**. Default NULL If tad.ranges is present, colours expects a hexcode value of length 1. But, if tad.colour.col is specified, it expects colours of the same length as unique tad.ranges$tad.colour.col.

`colours.names` **Optional**. Default NULL If present, will be assigned to colours. Else, will inherit unique tad.colour.col. If tad.colour.col is also absent, will revert to a placehold column name.

`cut.corners` **Optional**. Default FALSE if cut.corners is TRUE, TAD borders will not be truncated, and they will span until the end of visible heatmap.

`highlight.points` **Optional**. Not yet implemented.

`width` **Optional**. Default 10cm Width of the output file units.

`height` **Optional**. Default 6cm Height of the output file in units.

`line.width` **Optional**. Default 0.5 When plotting TADs set the width of the plotted lines

`units` **Optional**. Default cm Defines the units of the output file width and height.

`legend.key.width` **Optional**. Default unit(3,"cm") Defines the legend key width.

`legend.key.height` **Optional**. Default unit(0.5,"cm") Defines the legend key height.

Details

This function provides the capability to plot various types of heatmaps from Hi-C data.

- One sample heatmap.
- Two sample heatmap (One sample on upper and other on lower).
- All of the above with 90 degree rotation.
- All of the above but with signal capped at a certain value.
- All of the above but filtered by distance.
- All of the above with TADs/TAD borders plotted on top.

Value

If return.object is set to TRUE, the constructed ggplot2 object will be returned. Else TRUE.

Examples

```
FailSafe_log10 <- function(x){
     x[is.na(x) | is.nan(x) | is.infinite(x)] <- 0
     return(log10(x+1))
}

Brick.file <- system.file("extdata", "test.hdf", package = "HiCBricks")
Brick_vizart_plot_heatmap(File = "./chr19-5000000-10000000.pdf",
Bricks = Brick.file, x.coords = "chr19:5000000:10000000", palette = "Reds",
y.coords = "chr19:5000000:10000000", FUN = FailSafe_log10,
value.cap = 0.99, width = 10, height = 11, legend.key.width = unit(3,"mm"),
legend.key.height = unit(0.3,"cm"))
```

`CreateBrick` *Create the entire HDF5 structure and load the bintable*

Description

`CreateBrick` creates the complete HDF5 on-disk data structure

Usage

```
CreateBrick(ChromNames, BinTable, bin.delim = "\t", col.index = c(1,
    2, 3), impose.discontinuity = TRUE, ChunkSize = NULL,
    Output.Filename, exec = "cat", remove.existing = FALSE)
```

Arguments

`ChromNames`	**Required** A character vector containing the chromosomes to be considered for the dataset. This string is used to verify the presence of all chromosomes in the provided bitable.
`BinTable`	**Required** A string containing the path to the file to load as the binning table for the Hi-C experiment. The number of entries per chromosome defines the dimension of the associated Hi-C data matrices. For example, if chr1 contains 250 entries in the binning table, the *cis* Hi-C data matrix for chr1 will be expected to contain 250 rows and 250 cols. Similary, if the same binning table contained 150 entries for chr2, the *trans* Hi-C matrices for chr1,chr2 will be a matrix with dimension 250 rows and 150 cols. There are no constraints on the bintable format. As long as the table is in a delimited format, the corresponding table columns can be outlined with the associated parameters. The columns of importance are chr, start and end. It is recommended to always use binning tables where the end and start of consecutive ranges are not the same. If they are the same, this may lead to **unexpected behaviour** when using the GenomicRanges "any" overlap function.
`bin.delim`	**Optional**. Defaults to tabs. A character vector of length 1 specifying the delimiter used in the file containing the binning table.
`col.index`	**Optional**. Default "c(1,2,3)". A character vector of length 3 containing the indexes of the required columns in the binning table. the first index, corresponds to the chr column, the second to the start column and the third to the end column.
`impose.discontinuity`	**Optional**. Default TRUE. If TRUE, this parameter ensures a check to make sure that required the end and start coordinates of consecutive entries are not the same per chromosome.
`ChunkSize`	**Optional**. A numeric vector of length 1. If provided, the HDF dataset will use this value as the chunk size, for all matrices. By default, the ChunkSize is set to matrix dimensions/100.
`Output.Filename`	**Required** A string specifying the location and name of the HDF file to create. If path is not provided, it will be created in the Bioc File cache. Otherwise, it will be created in the specified directory and tracked via Bioc File Cache.
`exec`	**Optional**. Default cat. A string specifying the program or expression to use for reading the file. For bz2 files, use bzcat and for gunzipped files use zcat.

`remove.existing`
: **Optional**. Default FALSE. If TRUE, will remove the HDF file with the same name and create a new one. By default, it will not replace existing files.

Details

This function creates the complete HDF data structure, loads the binning table associated to the Hi-C experiment and creates (for now) a 2D matrix layout for all chromosome pairs. **Please note**, the binning table must be a discontinuous one (first range end != secode range start), as ranges overlaps using the "any" form will routinely identify adjacent ranges with the same end and start to be in the overlap. Therefore, this criteria is enforced as default behaviour.

The structure of the HDF file is as follows: The structure contains three major groups which are then hierarchically nested with other groups to finally lead to the corresponding datasets.

- Base.matrices - **group** For storing Hi-C matrices
  - chromosome - **group**
  - chromosome - **group**
    - attributes - **attribute**
      - Filename - Name of the file
      - Min - min value of Hi-C matrix
      - Max - max value of Hi-C matrix
      - sparsity - specifies if this is a sparse matrix
      - distance - max distance of data from main diagonal
      - Done - specifies if a matrix has been loaded
    - matrix - **dataset** - contains the matrix
    - bin.coverage - **dataset** - proportion of cells with values greater than 0
    - row.sums - **dataset** - total sum of all values in a row
    - sparsity - **dataset** - proportion of non-zero cells near the diagonal
- Base.ranges - **group**, Ranges tables for quick and easy access. Additional ranges tables are added here under separate group names.
  - Bintable - **group** - The main binning table associated to a Brick.
    - ranges - **dataset** - Contains the three main columns chr, start and end.
    - offsets - **dataset** - first occurence of any given chromosome in the ranges dataset.
    - lengths - **dataset** - Number of occurences of that chromosome
    - chr.names - **dataset** - What chromosomes are present in the given ranges table.
- Base.metadata - **group**, A place to store metadata info
  - chromosomes - **dataset** - Metadata information specifying the chromosomes present in this particular Brick file.
  - other metadata tables.

Value

This function will generate the target Brick file. Upon completion, the function will provide the path to the created/tracked HDF file.

Examples

```
Bintable.path <- system.file("extdata",
"Bintable_40kb.txt", package = "HiCBricks")
Chromosomes <- "chr19"
Path_to_cached_file <- CreateBrick(ChromNames = Chromosomes,
  BinTable = Bintable.path, bin.delim = " ",
  Output.Filename = file.path(tempdir(),"test.hdf"), exec = "cat",
  remove.existing = TRUE)

## Not run:
Bintable.path <- system.file("extdata",
"Bintable_40kb.txt", package = "HiCBricks")
Chromosomes <- c("chr19", "chr20", "chr22", "chr21")
Path_to_cached_file <- CreateBrick(ChromNames = Chromosomes,
BinTable = Bintable.path, impose.discontinuity=TRUE,
col.index = c(1,2,3), Output.Filename = file.path(tempdir(),"test.hdf"),
exec = "cat", remove.existing = TRUE)

This will cause an error as the file located at Bintable.path,
contains coordinates for only chromosome 19. For this code to work, either
all other chromosomes need to be removed from the Chromosomes variable or
coordinate information for the other chromosomes need to be provided.

Similarly vice-versa is also true. If the Bintable contains data for other
chromosomes, but they were not listed in ChromNames, this will cause an
error.

Keep in mind that if the end coordinates and start coordinates of adjacent
ranges are not separated by at least a value of 1, then
impose.discontinuity = TRUE will likely cause an error to occur.
This may seem obnoxious, but GenomicRanges by default will consider an
overlap of 1 bp as an overlap. Therefore, to be certain that ranges which
should not be, are not being targeted during retrieval operations, a check
is initiated to make sure that adjacent ends and starts are not
overlapping.
To load continuous ranges, use impose.discontinuity = FALSE.

Also note, that col.index determines which columns to use for chr, start
and end. Therefore, the original binning table may have 10 or 20 columns,
but it only requires the first three in order of chr, start and end.

## End(Not run)
```

CreateBrick_from_mcool

Create the entire HDF5 structure and load the bintable from a mcool file

---

Description

CreateBrick_from_mcool is a wrapper on CreateBrick which creates the Brick data structure from an mcool file.

Usage

```
CreateBrick_from_mcool(Brick, mcool, binsize = NULL, chrs = NULL,
    remove.existing = FALSE)
```

Arguments

Brick	**Required**. A string specifying the path to the Brick store created with CreateBrick.
mcool	**Required**. Path to an mcool file.
binsize	**Optional**. The binsize to select from an mcool file.
chrs	**Optional**. If provided will only create a Brick for these chromosomes (both cis & trans).
remove.existing	**Optional**. Default FALSE. If TRUE, will remove the HDF file with the same name and create a new one. By default, it will not replace existing files.

Details

mcool are a standard 4D nucleome data structure for Hi-C data. Read more about the 4D nucleome project here.

Value

This function will generate the target Brick file. Upon completion, the function will provide the path to the created/tracked HDF file.

See Also

Brick_load_data_from_mcool to load data from the mcool to a Brick store.

Examples

```
## Not run:
require(curl)
curl_download(url = paste("https://data.4dnucleome.org/"
"files-processed/4DNFI7JNCNFB/"
"@download/4DNFI7JNCNFB.mcool",sep = ""),
destfile = file.path(temp.dir(),"H1-hESC-HiC-4DNFI7JNCNFB.mcool"))

Output.brick <- file.path(tempdir(),
"H1-hESC-HiC-4DNFI7JNCNFB-10000-ICE-normalised-chr1.brick")
mcool <- file.path(temp.dir(),"H1-hESC-HiC-4DNFI7JNCNFB.mcool")

CreateBrick_from_mcool(Brick = Output.brick,
mcool = mcool,
binsize = 10000,
chrs = "chr1")


## End(Not run)
```

HiCBricks *A package for storing, accessing and plotting Hi-C data*

Description

HiCBricks is a package allowing users to flexibly import and work with Hi-C data

Details

Using HiCBricks users are able to import Hi-C matrices stored in various formats into an HDF structure. This is the Brick file. You can then access the Hi-C data using accessor functions. Since the data is stored in an HDF file, if you have the Brick (HDF) file, you can keep on accessing the same file an infinite number of times.

Users can also associate different ranges objects with the HDF file.

The HDF file must have the same structure as followed by HiCBricks

Users can then move forward and create analysis pipelines and statistical methods based on HiCBricks HDF files without worrying about the underlying data structure. To showcase this, Local score differentiator (LSD) our novel TAD calling procedure comes packaged with HiCBricks.

You are also able to plot Hi-C data using HiCBricks functions. There are a few types. You can create,

- a square heatmap
- a rotated heatmap
- two group square/rotated heatmaps
- both heatmaps until a certain distance
- plot TADs on both heatmaps

Brick creation

- `CreateBrick` - Create the complete HDF data structure. We refer to the HDF files as Brick
- `CreateBrick_from_mcool` - Create the complete Brick data structure from an mcool file.

Matrix loaders

- `Brick_load_matrix` - Load a complete nxm dimensional matrix.
- `Brick_load_cis_matrix_till_distance` - Load a sam chromosome nxn dimensional matrix until a certain distance.
- `Brick_load_data_from_mcool` - Load parts of the data from the 4DN consortium generated mcool files.

Matrix Accessors

- `Brick_get_matrix_within_coords` - Fetches a matrix within the provided genomic coordinates.
- `Brick_get_matrix` - Fetches a matrix within the provided x and y coordinates.
- `Brick_get_values_by_distance` - Fetch all values corresponding to interactions between genomic loci separated by the corresponding value.
- `Brick_fetch_row_vector` - Fetch all values at a given row or column.

All of the functions above can be subsetted and contain further value transformations.

Ranges operators

- `Brick_get_bintable` - All HiCBricks Brick files contain a binning table containing the coordinate information of the matrix. This fetches the associated binning table.
- `Brick_add_ranges` - Add a ranges object to the Brick file.
- `Brick_get_ranges` - Get a ranges object associated to a Brick file.
- `Brick_fetch_range_index` - Provided a set of coordinate vectors, get the corresponding rows/cols overlapping with those coordinates.
- `Brick_make_ranges` - Create a granges object from provided vectors.
- `Brick_return_region_position` - Get the row/col number corresponding to coordinates spelled out in human readable format.

Other functions

- `Brick_local_score_differentiator` - Use the LSD TAD calling procedure to do some TAD calls.
- `Brick_vizart_plot_heatmap` - Plot pretty heatmaps.

Utility functions

- `Brick_get_chrominfo` - Get the basic information regarding the Brick file. Which chromosomes are present, dimension of the matrix and the total length of the chromosome.
- `Brick_get_matrix_mcols` - Get the matrix metadata information. Such as, row sums, coverage information and how sparse regions near the diagonal are.
- `Brick_list_matrices` - List all the matrices present in the Brick file. Alongside, also provide information such as if the matrix has been loaded or not, min max values, e.t.c
- `Brick_list_rangekeys` - List the names of the ranges present in the Brick file.
- `Brick_rangekey_exists` - Answers the question, is this rangekey present in the Brick file?
- `Brick_list_ranges_mcols` - List the names of metadata columns associated to a ranges object in the Brick file.
- `Brick_matrix_dimensions` - Get the dimensions of a given matrix.
- `Brick_matrix_exists` - Answers the question, has a matrix been created for this Brick store?
- `Brick_matrix_filename` - Answers the question, what is the name of the file used to load this particular matrix?
- `Brick_matrix_isdone` - Answers the question, has this matrix been loaded already?
- `Brick_matrix_issparse` - Answers the question, was this matrix defined as a sparse matrix while loading?
- `Brick_matrix_maxdist` - If `Brick_load_cis_matrix_till_distance` was used for loading data, then this function will tell you until what distance data was loaded.
- `Brick_matrix_minmax` - Outputs the value range of the matrix.

mcool utility functions

- `Brick_list_mcool_normalisations` - List the names of normalisation vectors that can be present in a mcool file.
- `Brick_mcool_normalisation_exists` - Check if a specific normalisation vector exists in an mcool file.
- `Brick_list_mcool_resolutions` - List the resolutions present in an mcool file.

Index

Brick_add_ranges, 2, *39*
Brick_fetch_range_index, 4, *39*
Brick_fetch_row_vector, 5, *6*, *8*, *10, 11*, *38*
Brick_get_bintable, 6, *39*
Brick_get_chrominfo, 7, *39*
Brick_get_matrix, *6*, 7, *9–11*, *38*
Brick_get_matrix_mcols, 8, *39*
Brick_get_matrix_within_coords, *6*, *8*, 9, *11*, *38*
Brick_get_ranges, *6*, 10, *39*
Brick_get_values_by_distance, *6*, *8*, *10*, 11, *38*
Brick_get_vector_values, *8*, *10, 11*, 12
Brick_list_matrices, 13, *39*
Brick_list_matrix_mcols, 14
Brick_list_mcool_normalisations, 14, *19*, *39*
Brick_list_mcool_resolutions, 15, *19*, *39*
Brick_list_rangekeys, 15, *39*
Brick_list_ranges_mcols, 16, *39*
Brick_load_cis_matrix_till_distance, 16, *38, 39*
Brick_load_data_from_mcool, 18, *37, 38*
Brick_load_matrix, 19, *38*
Brick_local_score_differentiator, 20, *39*
Brick_make_ranges, *4*, 22, *39*
Brick_matrix_dimensions, 23, *39*
Brick_matrix_exists, 24, *39*
Brick_matrix_filename, 25, *39*
Brick_matrix_isdone, 25, *39*
Brick_matrix_issparse, 26, *39*
Brick_matrix_maxdist, 27, *39*
Brick_matrix_minmax, 27, *39*
Brick_mcool_normalisation_exists, 28, *39*
Brick_rangekey_exists, 29, *39*
Brick_return_region_position, 30, *39*
Brick_vizart_plot_heatmap, 31, *39*

CreateBrick, 34, *38*
CreateBrick_from_mcool, *19*, 36, *38*

HiCBricks, 38