

# Package ‘compartmap’

October 18, 2022

**Type** Package

**Title** Higher-order chromatin domain inference in single cells from scRNA-seq and scATAC-seq

**Description** Compartmap performs direct inference of higher-order chromatin from scRNA-seq and scATAC-seq. This package implements a James-Stein estimator for computing single-cell level higher-order chromatin domains. Further, we utilize random matrix theory as a method to de-noise correlation matrices to achieve a similar “plaid-like” patterning as observed in Hi-C and scHi-C data.

**Version** 1.14.0

**Date** 2021-05-03

**URL** <https://github.com/biobenkj/compartmap>

**BugReports** <https://github.com/biobenkj/compartmap/issues>

**Encoding** UTF-8

**License** GPL-3 + file LICENSE

**biocViews** Genetics, Epigenetics, ATACSeq, RNASeq, SingleCell

**Depends** R (>= 4.1.0), SummarizedExperiment, RaggedExperiment, BiocSingular, HDF5Array

**Imports** GenomicRanges, parallel, grid, ggplot2, reshape2, scales, DelayedArray, rtracklayer, DelayedMatrixStats, Matrix, RMTstat

**Suggests** covr, testthat, knitr, Rcpp, rmarkdown, markdown

**RoxygenNote** 7.1.1

**Roxygen** list(markdown = TRUE)

**VignetteBuilder** knitr

**git\_url** <https://git.bioconductor.org/packages/compartmap>

**git\_branch** RELEASE\_3\_15

**git\_last\_commit** e33c33b

**git\_last\_commit\_date** 2022-04-26

**Date/Publication** 2022-10-18

**Author** Benjamin Johnson [aut, cre],  
 Tim Triche [aut],  
 Hui Shen [aut],  
 Kasper Hansen [aut],  
 Jean-Philippe Fortin [aut]

**Maintainer** Benjamin Johnson <ben.johnson@vai.org>

## R topics documented:

.n_approx	3
.p_approx	3
.z	4
agrestiCoullCI	4
bootstrapCompartments	5
checkAssayType	6
condenseRE	7
condenseSE	7
estRMT	8
extractOpenClosed	9
fexpit	10
filterCompartments	10
fisherZ	11
fixCompartments	12
flogit	12
getABSignal	13
getAssayNames	14
getATACABsignal	14
getBinMatrix	16
getChrs	17
getCorMatrix	18
getDenoisedMatrix	19
getDomainInflections	20
getGlobalMeans	21
getMatrixBlocks	21
getSeqLengths	22
getShrinkageTargets	23
getSVD	23
hdf5TFIDF	24
hg19.gr	25
hg38.gr	25
ifisherZ	26
importBigWig	26
k562_scatac_chr14	27
k562_scrna_chr14	28
k562_scrna_se_chr14	28
meanSmoother	29
mm10.gr	29

`.n_approx` 3

<code>mm9.gr</code>	30
<code>plotAB</code>	30
<code>plotCorMatrix</code>	32
<code>precomputeBootstrapMeans</code>	33
<code>removeEmptyBoots</code>	34
<code>scCompartments</code>	34
<code>shrinkBins</code>	35
<code>sparseToDenseMatrix</code>	36
<code>ss3_umi_sce</code>	37
<code>summarizeBootstraps</code>	38
<code>transformTFIDF</code>	38

**Index** 40

---

`.n_approx`                      *n\_tilde in AC*

---

**Description**

*n\_tilde in AC*

**Usage**

`.n_approx(n1, n0, q)`

**Arguments**

`n1`                      number of successes/ones  
`n0`                      number of failures/zeroes  
`q`                        quantile for eventual CI (e.g. 0.95 for a 95 percent binomial CI)

**Value**

the effective sample size for smoothed CIs

---

`.p_approx`                      *p\_tilde in AC*

---

**Description**

*p\_tilde in AC*

**Usage**

`.p_approx(n1, n0, q)`

**Arguments**

n1            number of successes/ones  
 n0            number of failures/zeroes  
 q             quantile for eventual CI (e.g. 0.95 for a 95 percent binomial CI)

**Value**

the approximate success probability for a smoothed CIs

---

.z                            *Normal alpha/2 quantile*

---

**Description**

Normal alpha/2 quantile

**Usage**

.z(q)

**Arguments**

q             the quantile at which to extract Z

**Value**

Z

---

agrestiCoullCI            *Agresti-Coull confidence interval for a binomial proportion*

---

**Description**

Agresti-Coull confidence interval for a binomial proportion

**Usage**

agrestiCoullCI(n1, n0, q)

**Arguments**

n1            number of successes/ones  
 n0            number of failures/zeroes  
 q             quantile for eventual CI (e.g. 0.95 for a 95 percent binomial CI)

**Value**

the approximate  $(q \times 100)$  percent confidence interval for  $(p|n1,n0,q)$

**Examples**

```
binom.ci <- agrestiCoullCI(10, 3, 0.95)
```

---

bootstrapCompartments *Non-parametric bootstrapping of compartments and summarization of bootstraps/compute confidence intervals*

---

**Description**

Non-parametric bootstrapping of compartments and summarization of bootstraps/compute confidence intervals

**Usage**

```
bootstrapCompartments(  
  obj,  
  original.obj,  
  bootstrap.samples = 1000,  
  chr = "chr14",  
  assay = c("rna", "atac"),  
  parallel = TRUE,  
  cores = 2,  
  targets = NULL,  
  res = 1e+06,  
  genome = c("hg19", "hg38", "mm9", "mm10"),  
  q = 0.95,  
  svd = NULL,  
  group = FALSE,  
  bootstrap.means = NULL  
)
```

**Arguments**

obj	List object of computed compartments for a sample with 'pc' and 'gr' as elements
original.obj	The original, full input SummarizedExperiment of all samples/cells
bootstrap.samples	How many bootstraps to run
chr	Which chromosome to operate on
assay	What sort of assay are we working on
parallel	Whether to run the bootstrapping in parallel

cores	How many cores to use for parallel processing
targets	Targets to shrink towards
res	The compartment resolution
genome	What genome are we working on
q	What sort of confidence intervals are we computing (e.g. 0.95 for 95 percentCI)
svd	The original compartment calls as a GRanges object
group	Whether this is for group-level inference
bootstrap.means	Pre-computed bootstrap means matrix

**Value**

Compartment estimates with summarized bootstraps and confidence intervals

**Examples**

```
# this needs a good example
```

---

checkAssayType	<i>Check if the assay is a SummarizedExperiment</i>
----------------	---

---

**Description**

Check if the assay is a SummarizedExperiment

**Usage**

```
checkAssayType(obj)
```

**Arguments**

obj	Input object
-----	--------------

**Value**

Boolean

**Examples**

```
data("k562_scrna_chr14", package = "compartmap")
checkAssayType(k562_scrna_chr14)
```

---

condenseRE	<i>Condense a RaggedExperiment to a list of SummarizedExperiments</i>
------------	---

---

**Description**

Condense a RaggedExperiment to a list of SummarizedExperiments

**Usage**

```
condenseRE(obj)
```

**Arguments**

obj                    Input RaggedExperiment

**Value**

A list of SummarizedExperiments corresponding to the assays in the input

**Examples**

```
gr1 <- GRangesList(GRanges(c("A:1-5", "A:4-6", "A:10-15"), score=1:3),
GRanges(c("A:1-5", "B:1-3"), score=4:5))
names(gr1) <- c("A", "B")
x <- RaggedExperiment(gr1)
x.condense <- condenseRE(x)
```

---

condenseSE	<i>Condense the output of condenseRE to reconstruct per-sample GRanges objects to plot</i>
------------	--

---

**Description**

Condense the output of condenseRE to reconstruct per-sample GRanges objects to plot

**Usage**

```
condenseSE(obj, sample.name = NULL)
```

**Arguments**

obj                    Output of condenseRE or can be a RaggedExperiment  
sample.name            Vector of samples/cells to extract

**Value**

GRanges or list of per-sample GRanges to pass to plotAB or export

## Examples

```
gr1 <- GRangesList(GRanges(c("A:1-5", "A:4-6", "A:10-15"), score=1:3),
  GRanges(c("A:1-5", "B:1-3"), score=4:5))
names(gr1) <- c("A", "B")
x <- RaggedExperiment(gr1)
condense.x <- condenseSE(x, sample.name = "A")
```

---

 estRMT

*Denoising of Covariance matrix using Random Matrix Theory*


---

## Description

Denoising of Covariance matrix using Random Matrix Theory

## Usage

```
estRMT(
  R,
  Q = NA,
  cutoff = c("max", "each"),
  eigenTreat = c("average", "delete"),
  numEig = 1
)
```

## Arguments

R	input matrix
Q	ratio of rows/size. Can be supplied externally or fit using data
cutoff	takes two values max/each. If cutoff is max, Q is fitted and cutoff for eigenvalues is calculated. If cutoff is each, Q is set to row/size. Individual cutoff for each eigenvalue is calculated and used for filtration.
eigenTreat	takes 2 values, average/delete. If average then the noisy eigenvalues are averaged and each value is replaced by average. If delete then noisy eigenvalues are ignored and the diagonal entries of the correlation matrix are replaced with 1 to make the matrix psd.
numEig	number of eigenvalues that are known for variance calculation. Default is set to 1. If numEig = 0 then variance is assumed to be 1.

## Details

This method takes in data as a matrix object. It then fits a marchenko pastur density to eigenvalues of the correlation matrix. All eigenvalues above the cutoff are retained and ones below the cutoff are replaced such that the trace of the correlation matrix is 1 or non-significant eigenvalues are deleted and diagonal of correlation matrix is changed to 1. Finally, correlation matrix is converted to covariance matrix. This function was taken and modified from the covmat package (<https://github.com/cran/covmat>) which has since been deprecated on CRAN.



**Value**

A denoised RMT object

**Author(s)**

Rohit Arora

**Examples**

```
rand_cor_mat <- cor(matrix(rnorm(100), nrow = 10))
denoised_rand_cor_mat <- estRMT(rand_cor_mat)$cov
```

---

extractOpenClosed	<i>Get the open and closed compartment calls based on sign of singular values</i>
-------------------	---

---

**Description**

Get the open and closed compartment calls based on sign of singular values

**Usage**

```
extractOpenClosed(gr, cutoff = 0, assay = c("rna", "atac"))
```

**Arguments**

gr	Input GRanges with associated mcols that represent singular values
cutoff	Threshold to define open and closed states
assay	The type of assay we are working with

**Value**

A vector of binary/categorical compartment states

**Examples**

```
dummy <- matrix(rnorm(10000), ncol=25)
sing_vec <- getSVD(dummy, k = 1, sing.vec = "right")
```

---

fexpit	<i>Helper function: expanded expit</i>
--------	--

---

**Description**

Helper function: expanded expit

**Usage**

```
fexpit(x, sqz = 1e-06)
```

**Arguments**

x	a vector of values between -Inf and +Inf
sqz	the amount by which we 'squoze', default is .000001

**Value**

a vector of values between 0 and 1 inclusive

**Examples**

```
x <- rnorm(n=1000)
summary(x)

sqz <- 1 / (10**6)
p <- fexpit(x, sqz=sqz)
summary(p)

all( (abs(x - flogit(p)) / x) < sqz )
all( abs(x - flogit(fexpit(x))) < sqz )
```

---

filterCompartments	<i>Filter compartments using confidence estimates and eigenvalue thresholds</i>
--------------------	---

---

**Description**

Filter compartments using confidence estimates and eigenvalue thresholds

**Usage**

```
filterCompartments(obj, min.conf = 0.7, min.eigen = 0.02)
```

**Arguments**

obj	Output of condenseSE or fixCompartments
min.conf	Minimum confidence estimate to use when filtering
min.eigen	Minimum absolute eigenvalue to use when filtering

**Value**

A filtered/subset of the input object/list

---

fisherZ	<i>Fisher's Z transformation</i>
---------	----------------------------------

---

**Description**

fisherZ returns (squeezed) Fisher's Z transformed Pearson's r

**Usage**

```
fisherZ(cormat)
```

**Arguments**

cormat	Pearson correlation matrix
--------	----------------------------

**Details**

This function returns (squeezed) Fisher's Z transformed Pearson's r

**Value**

Fisher Z transformed Pearson correlations

**Examples**

```
#Generate a random binary (-1, 1) matrix
mat <- matrix(sample(c(1,-1), 10000, replace = TRUE), ncol = 100)

#Correct matrix diag
diag(mat) <- 1

#Transform
mat.transform <- fisherZ(mat)
```

---

fixCompartments	<i>Invert, or "fix", compartments that have a minimum confidence score (1-min.conf)</i>
-----------------	---

---

**Description**

Invert, or "fix", compartments that have a minimum confidence score (1-min.conf)

**Usage**

```
fixCompartments(obj, min.conf = 0.8, parallel = FALSE, cores = 1)
```

**Arguments**

obj	Input RaggedExperiment or output of condenseSE
min.conf	Minimum confidence score to use
parallel	Whether to run in parallel
cores	How many cores to use if running in parallel

**Value**

A "fixed" set of compartments

---

flogit	<i>Helper function: squeezed logit</i>
--------	--

---

**Description**

Helper function: squeezed logit

**Usage**

```
flogit(p, sqz = 1e-06)
```

**Arguments**

p	a vector of values between 0 and 1 inclusive
sqz	the amount by which to 'squeeze', default is .000001

**Value**

a vector of values between -Inf and +Inf

**Examples**

```

p <- runif(n=1000)
summary(p)

sqz <- 1 / (10**6)
x <- flogit(p, sqz=sqz)
summary(x)

all( abs(p - fexpit(x, sqz=sqz)) < sqz )
all( abs(p - fexpit(flogit(p, sqz=sqz), sqz=sqz)) < sqz )

```

---

getABSignal

*Calculate Pearson correlations of smoothed eigenvectors*


---

**Description**

This function is used to generate a list `x` to be passed to `getABSignal`

**Usage**

```
getABSignal(x, squeeze = FALSE, assay = c("rna", "atac"))
```

**Arguments**

<code>x</code>	A list object from <code>getCorMatrix</code>
<code>squeeze</code>	Whether squeezing was used (implies Fisher's Z transformation)
<code>assay</code>	What kind of assay are we working on ("array", "atac", "bisulfite")

**Value**

A list `x` to pass to `getABSignal`

**Examples**

```

library(SummarizedExperiment)
library(BiocSingular)

#Generate random genomic intervals of 1-1000 bp on chr1-22
#Modified from https://www.biostars.org/p/225520/
random_genomic_int <- data.frame(chr = rep("chr14", 100))
random_genomic_int$start <- apply(random_genomic_int, 1, function(x) { round(runif(1, 0, getSeqLengths(chr = x))[]
random_genomic_int$end <- random_genomic_int$start + runif(1, 1, 1000)
random_genomic_int$strand <- "*"

#Generate random counts
counts <- rnbinom(1000, 1.2, 0.4)

```

```

#Build random counts for 10 samples
count.mat <- matrix(sample(counts, nrow(random_genomic_int) * 10, replace = FALSE), ncol = 10)
colnames(count.mat) <- paste0("sample_", seq(1:10))

#Bin counts
bin.counts <- getBinMatrix(count.mat, makeGRangesFromDataFrame(random_genomic_int), chr = "chr14", genome = "hg19")

#Calculate correlations
bin.cor.counts <- getCorMatrix(bin.counts)

#Get A/B signal
absignal <- getABSignal(bin.cor.counts)

```

---

<code>getAssayNames</code>	<i>Get the assay names from a SummarizedExperiment object</i>
----------------------------	---

---

### Description

Get the assay names from a SummarizedExperiment object

### Usage

```
getAssayNames(se)
```

### Arguments

`se` Input SummarizedExperiment object

### Value

The names of the assays

### Examples

```

data("k562_scrna_chr14", package = "compartmap")
getAssayNames(k562_scrna_chr14)

```

---

<code>getATACABsignal</code>	<i>Estimate A/B compartments from ATAC-seq data</i>
------------------------------	---

---

### Description

`getATACABsignal` returns estimated A/B compartments from ATAC-seq data.

**Usage**

```

getATACABsignal(
  obj,
  res = 1e+06,
  parallel = FALSE,
  chr = NULL,
  targets = NULL,
  cores = 2,
  bootstrap = TRUE,
  num.bootstraps = 100,
  genome = c("hg19", "hg38", "mm9", "mm10"),
  other = NULL,
  group = FALSE,
  boot.parallel = FALSE,
  boot.cores = 2
)

```

```

getRNAABsignal(
  obj,
  res = 1e+06,
  parallel = FALSE,
  chr = NULL,
  targets = NULL,
  cores = 2,
  bootstrap = TRUE,
  num.bootstraps = 100,
  genome = c("hg19", "hg38", "mm9", "mm10"),
  other = NULL,
  group = FALSE,
  boot.parallel = FALSE,
  boot.cores = 2
)

```

**Arguments**

obj	Input SummarizedExperiment object
res	Compartment resolution in bp
parallel	Whether to run samples in parallel
chr	What chromosome to work on (leave as NULL to run on all chromosomes)
targets	Samples/cells to shrink towards
cores	How many cores to use when running samples in parallel
bootstrap	Whether we should perform bootstrapping of inferred compartments
num.bootstraps	How many bootstraps to run
genome	What genome to work on ("hg19", "hg38", "mm9", "mm10")
other	Another arbitrary genome to compute compartments on

group	Whether to treat this as a group set of samples
boot.parallel	Whether to run the bootstrapping in parallel
boot.cores	How many cores to use for the bootstrapping

**Value**

A RaggedExperiment of inferred compartments

**Functions**

- getRNAABsignal: Alias for getATACABsignal

**Examples**

```
data("k562_scatac_chr14", package = "compartmap")
atac_compartments <- getATACABsignal(k562_scatac_chr14, parallel=FALSE, chr="chr14", bootstrap=FALSE, genome="hg
```

---

getBinMatrix	<i>Generate bins for A/B compartment estimation</i>
--------------	---

---

**Description**

Generate bins across a user defined chromosome for A/B compartment estimation. A/B compartment estimation can be used for non-supported genomes if chr.end is set.

**Usage**

```
getBinMatrix(
  x,
  genloc,
  chr = "chr1",
  chr.start = 0,
  chr.end = NULL,
  res = 1e+05,
  FUN = sum,
  genome = c("hg19", "hg38", "mm9", "mm10")
)
```

**Arguments**

x	A p x n matrix where p (rows) = loci and n (columns) = samples/cells
genloc	GRanges object that contains corresponding genomic locations of the loci
chr	Chromosome to be analyzed
chr.start	Starting position (in bp) to be analyzed
chr.end	End position (in bp) to be analyzed
res	Binning resolution (in bp)
FUN	Function to be used to summarize information within a bin
genome	Genome corresponding to the input data ("hg19", "hg38", "mm9", "mm10")



**Details**

This function is used to generate a list object to be passed to getCorMatrix

**Value**

A list object to pass to getCorMatrix

**Examples**

```
library(GenomicRanges)

#Generate random genomic intervals of 1-1000 bp on chr1-22
#Modified from https://www.biostars.org/p/225520/
random_genomic_int <- data.frame(chr = rep("chr14", 100))
random_genomic_int$start <- apply(random_genomic_int, 1, function(x) { round(runif(1, 0, getSeqLengths(chr = x))[]
random_genomic_int$end <- random_genomic_int$start + runif(1, 1, 1000)
random_genomic_int$strand <- "*"

#Generate random counts
counts <- rnbinom(1000, 1.2, 0.4)

#Build random counts for 10 samples
count.mat <- matrix(sample(counts, nrow(random_genomic_int) * 10, replace = FALSE), ncol = 10)
colnames(count.mat) <- paste0("sample_", seq(1:10))

#Bin counts
bin.counts <- getBinMatrix(count.mat, makeGRangesFromDataFrame(random_genomic_int), chr = "chr14", genome = "hg19
```

---

getChrs

*Get the chromosomes from an object*

---

**Description**

Get the chromosomes from an object

**Usage**

```
getChrs(obj)
```

**Arguments**

obj                    Input SummarizedExperiment object

**Value**

A character vector of chromosomes present in an object

**Examples**

```
data("k562_scrna_chr14", package = "compartmap")
getChrs(k562_scrna_chr14)
```

---

```
getCorMatrix
```

```
Calculate Pearson correlations of a binned matrix
```

---

**Description**

This function is used to generate a list object to be passed to getABSignal

**Usage**

```
getCorMatrix(binmat, squeeze = FALSE)
```

**Arguments**

binmat	A binned matrix list object from getBinMatrix
squeeze	Whether to squeeze the matrix for Fisher's Z transformation

**Value**

A list object to pass to getABSignal

**Examples**

```
library(GenomicRanges)

#Generate random genomic intervals of 1-1000 bp on chr1-22
#Modified from https://www.biostars.org/p/225520/
random_genomic_int <- data.frame(chr = rep("chr14", 100))
random_genomic_int$start <- apply(random_genomic_int, 1, function(x) { round(runif(1, 0, getSeqLengths(chr = x))[]
random_genomic_int$end <- random_genomic_int$start + runif(1, 1, 1000)
random_genomic_int$strand <- "*"

#Generate random counts
counts <- rnbinom(1000, 1.2, 0.4)

#Build random counts for 10 samples
count.mat <- matrix(sample(counts, nrow(random_genomic_int) * 10, replace = FALSE), ncol = 10)
colnames(count.mat) <- paste0("sample_", seq(1:10))

#Bin counts
bin.counts <- getBinMatrix(count.mat, makeGRangesFromDataFrame(random_genomic_int), chr = "chr14", genome = "hg19")

#Calculate correlations
bin.cor.counts <- getCorMatrix(bin.counts)
```

---

getDenoisedMatrix	<i>Wrapper to denoise a correlation matrix using a Random Matrix Theory approach</i>
-------------------	--

---

## Description

Wrapper to denoise a correlation matrix using a Random Matrix Theory approach

## Usage

```
getDenoisedCorMatrix(  
  obj,  
  res = 1e+06,  
  chr = "chr14",  
  genome = c("hg19", "hg38", "mm9", "mm10"),  
  iter = 2,  
  targets = NULL,  
  prior.means = NULL,  
  assay = c("rna", "atac")  
)
```

## Arguments

obj	SummarizedExperiment object with rowRanges for each feature and colnames
res	The resolution desired (default is a megabase 1e6)
chr	Which chromosome to perform the denoising
genome	Which genome (default is hg19)
iter	How many iterations to perform denoising
targets	Samples/cells to shrink towards
prior.means	The means of the bin-level prior distribution (default will compute them for you)
assay	What assay type this is ("rna", "atac")

## Value

A denoised correlation matrix object for plotting with plotCorMatrix

## Examples

```
data("k562_scrna_chr14", package = "compartmap")  
denoised_cor_mat <- getDenoisedCorMatrix(k562_scrna_chr14, genome = "hg19", assay = "rna")
```

---

getDomainInflections *A wrapper function to generate a GRanges object of chromatin domain inflection points*

---

### Description

A wrapper function to generate a GRanges object of chromatin domain inflection points

### Usage

```
getDomainInflections(  
  gr,  
  what = "score",  
  res = 1e+06,  
  chrs = c(paste0("chr", 1:22), "chrX"),  
  genome = c("hg19", "hg38", "mm9", "mm10")  
)
```

### Arguments

gr	Input GRanges object with mcols column corresponding to chromatin domains
what	The name of the column containing the chromatin domain information
res	What resolution the domains were called
chrs	Which chromosomes to work on
genome	Which genome does the input data come from

### Value

A GRanges object of compartment inflection points

### Examples

```
data("k562_scrna_chr14", package = "compartmap")  
chr14_domains <- scCompartments(k562_scrna_chr14,  
                               res = 1e6, genome = "hg19",  
                               group = TRUE, bootstrap = FALSE)  
chr14_domain_inflections <- getDomainInflections(chr14_domains, what = "pc")
```

---

getGlobalMeans	<i>Get the global means of a matrix</i>
----------------	---

---

**Description**

Get the global means of a matrix

**Usage**

```
getGlobalMeans(obj, targets = NULL, assay = c("atac", "rna"))
```

**Arguments**

obj	Input SummarizedExperiment object
targets	Column names or indices to indicate which samples to shrink towards
assay	What type of assay the data are from

**Value**

A vector of global or targeted means

**Examples**

```
data("k562_scrna_chr14", package = "compartmap")
scrna.global.means <- getGlobalMeans(k562_scrna_chr14, assay = "rna")
```

---

getMatrixBlocks	<i>Get chunked sets of row-wise or column-wise indices of a matrix</i>
-----------------	--

---

**Description**

Get chunked sets of row-wise or column-wise indices of a matrix

**Usage**

```
getMatrixBlocks(mat, chunk.size = 1e+05, by.row = TRUE, by.col = FALSE)
```

**Arguments**

mat	Input matrix
chunk.size	The size of the chunks to use for coercion
by.row	Whether to chunk in a row-wise fashion
by.col	Whether to chunk in a column-wise fashion

**Value**

A set of chunked indices

**Examples**

```
#make a sparse binary matrix
library(Matrix)
m <- 100
n <- 1000
mat <- round(matrix(runif(m*n), m, n))
mat.sparse <- Matrix(mat, sparse = TRUE)

#get row-wise chunks of 10
chunks <- getMatrixBlocks(mat.sparse, chunk.size = 10)
```

---

getSeqLengths

*Get the seqlengths of a chromosome*

---

**Description**

The goal for this function is to eliminate the need to lug around large packages when we only want seqlengths for things.

**Usage**

```
getSeqLengths(genome = c("hg19", "hg38", "mm9", "mm10"), chr = "chr14")
```

**Arguments**

genome	The desired genome to use ("hg19", "hg38", "mm9", "mm10")
chr	What chromosome to extract the seqlengths of

**Value**

The seqlengths of a specific chromosome

**Examples**

```
hg19.chr14.seqlengths <- getSeqLengths(genome = "hg19", chr = "chr14")
```

---

getShrinkageTargets     *Get the specified samples to shrink towards instead of the global mean*

---

**Description**

Get the specified samples to shrink towards instead of the global mean

**Usage**

```
getShrinkageTargets(obj, group)
```

**Arguments**

obj	Input matrix
group	Samples/colnames to use for targeted shrinkage

**Value**

A matrix composed of samples to shrink towards

**Examples**

```
dummy <- matrix(rnorm(1000), ncol=25)
dummy.sub <- getShrinkageTargets(dummy, group = c(1,5,8,10))
```

---

getSVD     *Compute the SVD of a matrix using irlba*

---

**Description**

Compute the SVD of a matrix using irlba

**Usage**

```
getSVD(matrix, k = 1, sing.vec = c("left", "right"))
```

**Arguments**

matrix	A p x n input matrix
k	Number of singular vectors to return
sing.vec	Whether to return the right or left singular vector

**Value**

A singular vector or matrix with sign corresponding to positive values

**Examples**

```
dummy <- matrix(rnorm(10000), ncol=25)
sing_vec <- getSVD(dummy, k = 1, sing.vec = "right")
```

---

hdf5TFIDF	<i>Transform/normalize compartment calls using TF-IDF on HDF5-backed objects</i>
-----------	--

---

**Description**

Transform/normalize compartment calls using TF-IDF on HDF5-backed objects

**Usage**

```
hdf5TFIDF(h5, scale.factor = 1e+05, return.dense = FALSE, return.se = FALSE)
```

**Arguments**

h5	SummarizedExperiment object, DelayedMatrix, or a normal matrix
scale.factor	Scaling factor for the term-frequency (TF)
return.dense	Whether to return a dense, in memory matrix
return.se	Whether to return the TF-IDF matrix as a new assay in the SummarizedExperiment

**Value**

A TF-IDF transformed matrix of the same dimensions as the input

**Examples**

```
m <- 1000
n <- 100
mat <- round(matrix(runif(m*n), m, n))
#Input needs to be a tall matrix
tfidf <- hdf5TFIDF(mat)
```



---

`hg19.gr`*hg19 seqlengths as a GRanges object*

---

**Description**

This object was generated using the Homo.sapiens package. The script used for this object is found in the inst/scripts directory

**Usage**

```
data(hg19.gr, package = "compartmap")
```

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

---

`hg38.gr`*hg38 seqlengths as a GRanges object*

---

**Description**

This object was generated using the BSgenome.Hsapiens.UCSC.hg38 package. The script used for this object is found in the inst/scripts directory

**Usage**

```
data(hg38.gr, package = "compartmap")
```

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

---

ifisherZ	<i>Fisher's Z transformation</i>
----------	----------------------------------

---

**Description**

fisherZ returns the inverse (squeezed) Fisher's Z transformed Pearson's r. This will fail if a matrix is used as input instead of a vector.

**Usage**

```
ifisherZ(cormat)
```

**Arguments**

cormat            vector of Fisher's Z transformed Pearson correlations or an eigenvector

**Details**

This function returns the inverse (squeezed) Fisher's Z transformed Pearson's r

**Value**

Back transformed Fisher's Z

**Examples**

```
#Generate a random binary (-1, 1) matrix
mat <- matrix(sample(c(1,-1), 10000, replace = TRUE), ncol = 100)

#Correct matrix diag
diag(mat) <- 1

#Transform
mat.transform <- fisherZ(mat)

#Back transform
mat.transform.inverse <- apply(mat.transform, 1, ifisherZ)
```

---

importBigWig	<i>Import and optionally summarize a bigwig at a given resolution</i>
--------------	---

---

**Description**

Import and optionally summarize a bigwig at a given resolution

**Usage**

```
importBigWig(  
  bw,  
  bins = NULL,  
  summarize = FALSE,  
  genome = c("hg19", "hg38", "mm9", "mm10")  
)
```

**Arguments**

bw	Path a bigwig file
bins	Optional set of bins as a GRanges to summarize the bigwig to
summarize	Whether to perform mean summarization
genome	Which genome is the bigwig from ("hg19", "hg38", "mm9", "mm10")

**Value**

SummerizedExperiment object with rowRanges corresponding to summarized features

---

k562_scatac_chr14	<i>Example scATAC-seq data for compartmap</i>
-------------------	---

---

**Description**

This data was generated using the data from the reference via bwa mem and pre-processing the data using the csaw package.

**Usage**

```
data(k562_scatac_chr14, package = "compartmap")
```

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

**References**

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE99172>

---

k562\_scrna\_chr14      *Example scRNA-seq data for compartmap*

---

**Description**

This object was generated using the K562 data from the STORM-seq paper and pre-processed using the scran and scater packages and TF-IDF transformed.

**Usage**

```
data(k562_scrna_chr14, package = "compartmap")
```

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

---

k562\_scrna\_se\_chr14      *Example scRNA-seq data for compartmap*

---

**Description**

This object was generated using the K562 data from the STORM-seq paper and pre-processed using the scran and scater packages and are raw counts.

**Usage**

```
data(k562_scrna_raw, package = "compartmap")
```

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

---

meanSmoother	<i>Windowed mean smoother</i>
--------------	-------------------------------

---

**Description**

TODO: farm out to C++ and test, at least when there are no NAs

**Usage**

```
meanSmoother(x, k = 1, iter = 2, na.rm = TRUE, delta = 0, w = NULL)
```

**Arguments**

x	Input data matrix: samples are columns, regions/loci are rows
k	Number of windows to use (default k=1, i.e., 3 windows)
iter	Number of iterations to smooth (default is 2)
na.rm	Whether to remove NAs prior to smoothing (TRUE)
delta	Convergence threshold (overrides iter if > 0; default is 0)
w	Weights, if using any (NULL)

**Value**

Smoothed data matrix

**Examples**

```
dummy <- matrix(rnorm(10000), ncol=25)
smooth.dummy <- meanSmoother(dummy)
smooth.dummy <- meanSmoother(dummy, iter=3)
smooth.dummy <- meanSmoother(dummy, delta=1e-3)
```

---

mm10.gr	<i>mm10 seqlengths as a GRanges object</i>
---------	--

---

**Description**

This object was generated using the Mus.musculus package. The script used for this object is found in the inst/scripts directory

**Usage**

```
data(mm10.gr, package = "compartmap")
```

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

---

`mm9.gr`*mm9 seqlengths as a GRanges object*

---

**Description**

This object was generated using the `BSSgenome.Mmusculus.UCSC.mm9` package. The script used for this object is found in the `inst/scripts` directory

**Usage**

```
data(mm9.gr, package = "compartmap")
```

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

---

`plotAB`*Plots A/B compartment estimates on a per chromosome basis*

---

**Description**

Plot A/B compartments bins

**Usage**

```
plotAB(  
  x,  
  chr = NULL,  
  what = "score",  
  main = "",  
  ylim = c(-1, 1),  
  unitarize = FALSE,  
  reverse = FALSE,  
  top.col = "deeppink4",  
  bot.col = "grey50",  
  with.ci = FALSE,  
  filter = TRUE,  
  filter.min.eigen = 0.02,  
  median.conf = FALSE  
)
```

**Arguments**

x	The matrix object returned from getCompartments
chr	Chromosome to subset to for plotting
what	Which metadata column to plot
main	Title for the plot
ylim	Y-axis limits (default is -1 to 1)
unitarize	Should the data be unitarized?
reverse	Reverse the sign of the PC values?
top.col	Top (pos. PC values) chromatin color to be plotted
bot.col	Bottom (neg. PC values) chromatin color to be plotted
with.ci	Whether to plot confidence intervals
filter	Whether to filter eigenvalues close to zero (default: TRUE)
filter.min.eigen	Minimum absolute eigenvalue to include in the plot
median.conf	Plot the median confidence estimate across the chromosome?

**Value**

A plot of inferred A/B compartments

**Examples**

```
library(GenomicRanges)

#Generate random genomic intervals of 1-1000 bp on chr1-22
#Modified from https://www.biostars.org/p/225520/
random_genomic_int <- data.frame(chr = rep("chr14", 100))
random_genomic_int$start <- apply(random_genomic_int, 1, function(x) { round(runif(1, 0, getSeqLengths(chr = x))[]
random_genomic_int$end <- random_genomic_int$start + runif(1, 1, 1000)
random_genomic_int$strand <- "*"

#Generate random counts
counts <- rnbinom(1000, 1.2, 0.4)

#Build random counts for 10 samples
count.mat <- matrix(sample(counts, nrow(random_genomic_int) * 10, replace = FALSE), ncol = 10)
colnames(count.mat) <- paste0("sample_", seq(1:10))

#Bin counts
bin.counts <- getBinMatrix(count.mat, makeGRangesFromDataFrame(random_genomic_int), chr = "chr14", genome = "hg19")

#Calculate correlations
bin.cor.counts <- getCorMatrix(bin.counts)

#Get A/B signal
absignal <- getABSignal(bin.cor.counts)
```

```
#Plot the A/B signal
par(mar=c(1,1,1,1))
par(mfrow=c(1,1))
plotAB(absignal, what = "pc")
```

---

plotCorMatrix

*Plot a denoised correlation matrix*

---

## Description

Plot a denoised correlation matrix

## Usage

```
plotCorMatrix(
  denoised.cor.mat,
  midpoint = 0.3,
  return.plot.obj = FALSE,
  uppertri = FALSE,
  lowertri = FALSE
)
```

## Arguments

denoised.cor.mat	The denoised correlation matrix object from getDenoisedMatrix
midpoint	The midpoint for the coloring (default is 0.3)
return.plot.obj	Whether to return the ggplot object
uppertri	Whether to keep the upper triangle of the matrix
lowertri	Whether to keep the lower triangle of the matrix

## Value

Either a ggplot object or plot

## Examples

```
dummy <- matrix(rnorm(10000), ncol=25)
set.seed(1000)
my_plot <- plotCorMatrix(dummy, return.plot.obj = TRUE)
```



---

`precomputeBootstrapMeans`*Pre-compute the global means for bootstrapping compartments*

---

**Description**

Pre-compute the global means for bootstrapping compartments

**Usage**

```
precomputeBootstrapMeans(  
  obj,  
  targets = NULL,  
  num.bootstraps = 100,  
  assay = c("atac", "rna"),  
  parallel = FALSE,  
  num.cores = 1  
)
```

**Arguments**

<code>obj</code>	Input SummarizedExperiment object
<code>targets</code>	Optional targets to shrink towards
<code>num.bootstraps</code>	The number of bootstraps to compute
<code>assay</code>	What type of assay the data are from
<code>parallel</code>	Whether to run in parallel
<code>num.cores</code>	How many cores to use for parallel processing

**Value**

A matrix of bootstrapped global means

**Examples**

```
data("k562_scrna_chr14", package = "compartmap")  
scrna.bootstrap.global.means <- precomputeBootstrapMeans(k562_scrna_chr14, assay = "rna", num.bootstraps = 2)
```

---

removeEmptyBoots	<i>Remove bootstrap estimates that failed</i>
------------------	---

---

**Description**

Remove bootstrap estimates that failed

**Usage**

```
removeEmptyBoots(obj)
```

**Arguments**

obj                    Input list object with elements 'pc' and 'gr'

**Value**

A filtered list object

---

scCompartments	<i>Estimate A/B compartments from single-cell sequencing data</i>
----------------	---

---

**Description**

scCompartments returns estimated A/B compartments from sc-seq data.

**Usage**

```
scCompartments(  
  obj,  
  res = 1e+06,  
  parallel = FALSE,  
  chr = NULL,  
  targets = NULL,  
  cores = 2,  
  bootstrap = TRUE,  
  num.bootstraps = 100,  
  genome = c("hg19", "hg38", "mm9", "mm10"),  
  group = FALSE,  
  assay = c("atac", "rna")  
)
```

**Arguments**

obj	Input SummarizedExperiment object
res	Compartment resolution in bp
parallel	Whether to run samples in parallel
chr	What chromosome to work on (leave as NULL to run on all chromosomes)
targets	Samples/cells to shrink towards
cores	How many cores to use when running samples in parallel
bootstrap	Whether we should perform bootstrapping of inferred compartments
num.bootstraps	How many bootstraps to run
genome	What genome to work on ("hg19", "hg38", "mm9", "mm10")
group	Whether to treat this as a group set of samples
assay	What type of single-cell assay is the input data ("atac" or "rna")

**Value**

A RaggedExperiment of inferred compartments

**Examples**

```
data("k562_scrna_chr14", package = "compartmap")
sc_compartments <- scCompartments(k562_scrna_chr14, parallel=FALSE, chr="chr14", bootstrap=FALSE, genome="hg19")
```

---

shrinkBins	<i>Employ an eBayes shrinkage approach for bin-level estimates for A/B inference</i>
------------	--

---

**Description**

shrinkBins returns shrunken bin-level estimates

**Usage**

```
shrinkBins(
  x,
  original.x,
  prior.means = NULL,
  chr = NULL,
  res = 1e+06,
  targets = NULL,
  jse = TRUE,
  assay = c("rna", "atac"),
  genome = c("hg19", "hg38", "mm9", "mm10")
)
```

**Arguments**

<code>x</code>	Input SummarizedExperiment object
<code>original.x</code>	Full sample set SummarizedExperiment object
<code>prior.means</code>	The means of the bin-level prior distribution
<code>chr</code>	The chromosome to operate on
<code>res</code>	Resolution to perform the binning
<code>targets</code>	The column/sample/cell names to shrink towards
<code>jse</code>	Whether to use a James-Stein estimator (default is TRUE)
<code>assay</code>	What assay type this is ("rna", "atac")
<code>genome</code>	What genome are we working with ("hg19", "hg38", "mm9", "mm10")

**Details**

This function computes shrunken bin-level estimates using a James-Stein estimator, reformulated as an eBayes procedure

**Value**

A list object to pass to `getCorMatrix`

**Examples**

```
data("k562_scrna_chr14", package = "compartmap")
shrunken.bin.scrna <- shrinkBins(x = k562_scrna_chr14,
                                original.x = k562_scrna_chr14,
                                chr = "chr14", assay = "rna")
```

---

`sparseToDenseMatrix`    *Convert a sparse matrix to a dense matrix in a block-wise fashion*

---

**Description**

Convert a sparse matrix to a dense matrix in a block-wise fashion

**Usage**

```
sparseToDenseMatrix(
  mat,
  blockwise = TRUE,
  by.row = TRUE,
  by.col = FALSE,
  chunk.size = 1e+05,
  parallel = FALSE,
  cores = 2
)
```

**Arguments**

mat	Input sparse matrix
blockwise	Whether to do the coercion in a block-wise manner
by.row	Whether to chunk in a row-wise fashion
by.col	Whether to chunk in a column-wise fashion
chunk.size	The size of the chunks to use for coercion
parallel	Whether to perform the coercion in parallel
cores	The number of cores to use in the parallel coercion

**Value**

A dense matrix of the same dimensions as the input

**Examples**

```
#make a sparse binary matrix
library(Matrix)
m <- 100
n <- 1000
mat <- round(matrix(runif(m*n), m, n))
mat.sparse <- Matrix(mat, sparse = TRUE)

#coerce back
mat.dense <- sparseToDenseMatrix(mat.sparse, chunk.size = 10)

#make sure they are the same dimensions
dim(mat) == dim(mat.dense)

#make sure they are the same numerically
all(mat == mat.dense)
```

---

ss3\_umi\_sce

*Example SMART-seq3 scRNA-seq data for compartmap*

---

**Description**

Only keep chromosome 22 for the example

**Usage**

```
data(ss3_umi_sce, package = "compartmap")
```

**Details**

This object was generated using the HEK293T data from the SMART-seq3 paper

**Author(s)**

Benjamin K Johnson <ben.johnson@vai.org>

**References**

<https://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-8735/>

---

summarizeBootstraps	<i>Summarize the bootstrap compartment estimates and compute Agresti-Coull confidence intervals</i>
---------------------	---

---

**Description**

Summarize the bootstrap compartment estimates and compute Agresti-Coull confidence intervals

**Usage**

```
summarizeBootstraps(boot.list, est.ab, q = 0.95, assay = c("rna", "atac"))
```

**Arguments**

boot.list	List of bootstraps as GRanges objects
est.ab	The original compartment calls
q	Confidence interval to compute (0.95 for 95 percent CI)
assay	Type of assay we are working with

**Value**

A GRanges object with bootstraps summarized

---

transformTFIDF	<i>Transform/normalize compartment calls using TF-IDF</i>
----------------	---

---

**Description**

Transform/normalize compartment calls using TF-IDF

**Usage**

```
transformTFIDF(obj, scale.factor = 1e+05)
```

**Arguments**

obj	n x p input matrix (n = samples/cells; p = compartments)
scale.factor	Scaling factor for the term-frequency (TF)

**Value**

A TF-IDF transformed matrix of the same dimensions as the input

**Examples**

```
m <- 1000
n <- 100
mat <- round(matrix(runif(m*n), m, n))
#Input needs to be a tall matrix
tfidf <- transformTFIDF(mat)
```

# Index

## \* data

- hg19.gr, [25](#)
- hg38.gr, [25](#)
- k562\_scatac\_chr14, [27](#)
- k562\_scrna\_chr14, [28](#)
- k562\_scrna\_se\_chr14, [28](#)
- mm10.gr, [29](#)
- mm9.gr, [30](#)
- ss3\_umi\_sce, [37](#)
- .n\_approx, [3](#)
- .p\_approx, [3](#)
- .z, [4](#)
- agrestiCoulICI, [4](#)
- bootstrapCompartments, [5](#)
- checkAssayType, [6](#)
- condenseRE, [7](#)
- condenseSE, [7](#)
- estRMT, [8](#)
- extractOpenClosed, [9](#)
- fexpit, [10](#)
- filterCompartments, [10](#)
- fisherZ, [11](#)
- fixCompartments, [12](#)
- flogit, [12](#)
- getABSignal, [13](#)
- getAssayNames, [14](#)
- getATACABsignal, [14](#)
- getBinMatrix, [16](#)
- getChrs, [17](#)
- getCorMatrix, [18](#)
- getDenoisedCorMatrix  
(getDenoisedMatrix), [19](#)
- getDenoisedMatrix, [19](#)
- getDomainInflections, [20](#)
- getGlobalMeans, [21](#)
- getMatrixBlocks, [21](#)
- getRNAABsignal (getATACABsignal), [14](#)
- getSeqLengths, [22](#)
- getShrinkageTargets, [23](#)
- getSVD, [23](#)
- hdf5TFIDF, [24](#)
- hg19.gr, [25](#)
- hg38.gr, [25](#)
- ifisherZ, [26](#)
- importBigWig, [26](#)
- k562\_scatac\_chr14, [27](#)
- k562\_scrna\_chr14, [28](#)
- k562\_scrna\_se\_chr14, [28](#)
- meanSmoother, [29](#)
- mm10.gr, [29](#)
- mm9.gr, [30](#)
- plotAB, [30](#)
- plotCorMatrix, [32](#)
- precomputeBootstrapMeans, [33](#)
- removeEmptyBoots, [34](#)
- scCompartments, [34](#)
- shrinkBins, [35](#)
- sparseToDenseMatrix, [36](#)
- ss3\_umi\_sce, [37](#)
- summarizeBootstraps, [38](#)
- transformTFIDF, [38](#)