Package 'RSVSim'

November 12, 2026

Type Package

Title RSVSim: an R/Bioconductor package for the simulation of structural variations

Version 1.52.0

Date 2024-04-23

Author Christoph Bartenhagen

Maintainer Christoph Bartenhagen

<c.bartenhagen@uni-koeln.de>

Imports methods, IRanges, ShortRead

Depends R (>= 3.0.0), Biostrings, GenomicRanges

Suggests BSgenome.Hsapiens.UCSC.hg19, BSgenome.Hsapiens.UCSC.hg19.masked, MASS, rtracklayer, pwalign

Description RSVSim is a package for the simulation of deletions, insertions, inversion, tandem-duplications and translocations of various sizes in any genome available as FASTA-file or BSgenome data package. SV breakpoints can be placed uniformly accross the whole genome, with a bias towards repeat regions and regions of high homology (for hg19) or at user-supplied coordinates.

License LGPL-3

biocViews Sequencing

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

git_branch RELEASE_3_23

git_last_commit 44b6005

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

compareSV 2
estimateSVSizes 5
segDups 6
simulateSV 7
weightsMechanisms 12
weightsRepeats 13

Index **14**

`compareSV` *Compare the simulation with a set of SVs*

Description

A typical use case of structural variant (SV) simulation with `simulateSV` is the evaluation of SV detection algorithms. The function `compareSV` looks for breakpoint overlaps between the output of the simulation (ground truth) and the output of an SV detection program and computes the sensitivity and precision. There is currently no common standard format for SVs. Because the main information about SVs is their position in the genome and, sometimes, the breakpoint sequence (depending on the SV detection algorithm), `compareSV` expects the SV detections in a simple BED- or BEDPE format.

Note that `compareSV` requires the **pwalign** package.

Usage

```
compareSV(querySVs, simSVs, tol=200)
```

Arguments

`querySVs`	The set detected of SVs. Either a filename for a table of SVs on disk or a `data.frame`. The table has to be in BED- or BEDPE-format (see details below). The `querySVs` may only contain SVs of one type and not a mixture of several kinds of SVs.
`simSVs`	The set of simulated SVs as returned by the function `simulateSV`. It can be either a filename for a table of SVs on disk or a `data.frame`.
`tol`	The tolerance in bp, when comparing the (approximate) positions in `querySVs` with the exact, simulated breakpoints `simSVs`. Positions in `querySVs` may only differ by at most `tol` base pairs from the simulated breakpoints.

Details

An overlap is defined as the overlap between the breakpoints / breakpoint regions up to the given tolerance in bp. Overlap does not mean the whole affected region between the start and end of the SV.

The comparison has to be done for each type of SV separately. It is required to use the returned

tables from simulateSV for the argument simSVs. For querySVs, the tables have to be in BED- or BEDPE-format, a simple tab-separated table with genome coordinates and an optional column with the breakpoint sequence (bpSeq):

- Deletions: (1) BED with columns chr, start, end (, bpSeq) for exact breakpoints or (2) BEDPE with columns chr, start1, end1, chr, start2, end2 (, bpSeq) for approximate breakpoint regions
- Insertions: BEDPE with columns chrA, startA, endA, chrB, startB, endB (, bpSeq). Typically, a complete insertion is reported in two rows, one for the breakpoint at the 5' and one at the 3' end
- Inversions: (1) BED with columns chr, start, end (, bpSeq1, bpSeq2) or (2) BEDPE with columns chr, start1, end1, chr, start2, end2 (, bpSeq1, bpSeq2). Inversions, the larger ones, typically have two breakpoint sequences (one for the 5' and on for the 3' end)
- Tandem duplications: (1) BED with columns chr, start, end (, bpSeq) or (2) BEDPE with columns chr, start1, end1, chr, start2, end2 (, bpSeq)
- Translocations: BEDPE with columns chrA, startA, endA, chrB, startB, endB (, bpSeq1, bpSeq2)

The BEDPE-format is required for insertions and translocations, since they involve two regions on the genome. For other SVs, the BEDPE allows to specify approximate regions for each breakpoint. For example a deletion "chr:start-end" can be given as BED-file with columns chr, start, end or (a little redundant) as BEDPE-file with columns chr, start, start, chr, end, end or (with some tolerance) like chr, start-tol, start+tol, chr, end-tol, end+tol. The tolerance can also be regulated by the function argument tol.

Value

The table of simulated SVs, as given in the function argument querySVs, but with additional columns for the overlapping region in querySVs and the percentage overlap between the breakpoint sequences (if they were provided as a column in querySVs).
Furthermore, the function prints the sensitivity and precision in the R console.

Author(s)

Christoph Bartenhagen

References

More information about the BED-format can be found on the BEDTools homepage: http://code.google.com/p/bedtools

See Also

simulateSV

Examples

```
 ## Toy example: Artificial genome with two chromosomes
genome = DNAStringSet(c("AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTTT", "GGGGGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCCCC"
 names(genome) = c("chr1","chr2")
```

```
#######################
## Example 1: Deletions
## Simulation of 5 deletions of 5bp each
sim = simulateSV(output=NA, genome=genome, dels=5, sizeDels=5, bpSeqSize=10, seed=246)
simSVs = metadata(sim)$deletions

## An SV detection in BED format may look like this:
## Four of five deletions were detected; two with exact and two with an approximate breakpoint
## Two additional deletions were detected, which were not part of the simulation
## The column with the breakpoint sequence is optional, the column names not important (BED-files have no header)
querySVs = data.frame(
  chr=c("chr1","chr1","chr1","chr2","chr2","chr2"),
  start=c(4,12,20,10,21,34),
  end=c(8,16,28,14,31,38),
 bpSeq=c("AAAAAAAAAA", "AAAAAAAAAT", "AAAATTTTTT", "GGGGGGGGGG", "GGGGGGGCCC", "CCCCCCCCCC")
)

## Compare the SVs with 0bp tolerance:
## Only the two exact detections have an overlap
simSVs_overlap1 = compareSV(querySVs, simSVs, tol=0)
simSVs_overlap1

## Increasing the breakpoint tolerance to +/- 3bp :
## Now, the overlap also includes the more imprecise detections
## And the sensitivity and precision are better
## Note that for deletion2, the breakpoint sequence matches only by 50%
simSVs_overlap2 = compareSV(querySVs, simSVs, tol=3)
simSVs_overlap2

############################
## Example 2: Translocations
## Simulation of 2 translocations (only one of them is balanced):
sim = simulateSV(output=NA, genome=genome, trans=2, percBalancedTrans=0.5, bpSeqSize=10, seed=246)
simSVs = metadata(sim)$translocations

## Detected translocations have to be given in BEDPE-format (i.e. at least six columns with chr,start,end for break
## In this example, the breakpoints were approximated up to 1 or 2bp
## Optional breakpoint sequences are missing
querySVs = data.frame(
  chr=c("chr2", "chr1", "chr2"),
  start1=c(25,3,9),
  end1=c(29,7,12),
  chr2=c("chr1","chr2","chr1"),
  start2=c(22,10,3),
  end2=c(25,13,4)
)

simSVs_overlap = compareSV(querySVs, simSVs, tol=0)
simSVs_overlap
```

`estimateSVSizes` *Draw random structural variation sizes from a beta distribution*

Description

RSVSim can implement structural variations (SVs) of specific sizes. `estimateSVSizes` draws random values for SV sizes from a beta distribution. It can fit the distribution according to given SV sizes or default values.

Usage

```
estimateSVSizes(n, svSizes, minSize, maxSize, default, hist=TRUE)
```

Arguments

`n`	The number of SVs to simulate
`svSizes`	A numeric vector with SV sizes to calculate the parameters for the beta distribution
`minSize`	The minimum returned SV size
`maxSize`	The maximum returned SV size
`hist`	TRUE or FALSE to show a histogram of the SV sizes (or not)
`default`	Setting this to "deletions", "insertions", "inversions" or "tandemDuplications" loads default shape parameters for the beta distribution for these SV types (see details below)

Details

- `minSize` and `maxSize` are optional and taken from the given set of `svSizes` if omitted
- The default shape parameters for deletions, insertions, inversions and tandem duplications were estimated from sequencing studies in the Database of Genomic Variants release 2012-03-29. In total, 1.129 deletions, 490 insertions, 202 inversions and 145 tandem duplications between 500bp and 10kb were used to estimate the shape of the distribution.

Value

A numeric vector with beta distributed values between `minSize` and `maxSize`.

Note

- It is intended to run this function separately for every SV type and then provide the output to the respective size argument of `simulateSV`
- It is recommended to use a `minSize` and `maxSize` that is consistent with the minimum/maximum values in `svSizes`.
- When using default shape parameters for the beta distribution, it works best to simulate SVs that do not differ too much in size (aroung 500bp-10kb).

Author(s)

Christoph Bartenhagen

References

Database of Genomic Variants: http://dgvbeta.tcag.ca/dgv/app/home?ref=NCBI36/hg18

Examples

```
## estimate sizes for 20 SVs from a given set of values:
svSizes = c(10,20,30,40,60,80,100,150,200,250,300,400,500,750,1000)
estimateSVSizes(n=20, svSizes=svSizes, hist=FALSE)

## when using the default shape parameters for deletions:
estimateSVSizes(n=20, minSize=500, maxSize=10000, hist=FALSE, default="deletions")
```

---

segDups *Segmental duplications*

Description

A list with coordinates of segmental duplications detected by RepeatMasker. It is loaded automatically when the repeat bias feature has been turned on (`repeatBias=TRUE` in function `simulateSV`). (But this dataset is intended for internal use; no need to handle it manually.)

Usage

```
data("segmentalDuplications")
```

Format

Formal class 'GRanges'

Source

Downloaded from the UCSC Browser's RepeatMasker track. Meyer et al., The UCSC Genome Browser database: extensions and updates 2013, 2013, Nucleic Acids Res, 41(Database issue), 64-69. Smit et al., RepeatMasker Open-3.0., 1996-2010, <http://www.repeatmasker.org>.

Examples

```
data("segmentalDuplications")
segDups
```

simulateSV *Structural Variant Simulation*

Description

A tool for simulating deletions, insertions, inversions, tandem duplications and translocations in any genome available as FASTA-file or BSgenome data package. Structural variations (SVs) are placed within the given genome, or only a subset of it, in a random, non-overlapping manner or at given genomic coordinates. SV breakpoints can be positioned uniformly or with a bias towards repeat regions and regions of high homology.

Usage

```
simulateSV(output=".", genome, chrs, dels=0, ins=0, invs=0, dups=0, trans=0, size, sizeDels=10, sizeIns
```

Arguments

output	Output directory for the rearranged genome and SV lists; turn this off by passing NA (default: current directory)
genome	The genome as DNAStringSet or as filename pointing to a FASTA-file containing the genome sequence
chrs	Restrict simulation to certain chromosomes only (default: all chromosomes available)
dels	Number of deletions
ins	Number of insertions
invs	Number of inversions
dups	Number of tandem duplications
trans	Number of translocations
size	Size of SVs in bp (a single numeric value); a quick way to set a size, which is applied to all simulated SVs
sizeDels	Size of deletions: Either a single number for all deletions or a vector with a length for every single deletion
sizeIns	Size of insertions: Either a single number for all insertions or a vector with a length for every single insertion
sizeInvs	Size of inversions: Either a single number for all inversions or a vector with a length for every single inversion
sizeDups	Size of tandem duplications: Either a single number for all tandem duplications or a vector with a length for every single tandem duplication
regionsDels	GRanges object with regions within the genome where to place the deletions
regionsIns	GRanges object with regions within the genome where to place the Insertions
regionsInvs	GRanges object with regions within the genome where to place the inversions

regionsDups	GRanges object with regions within the genome where to place the tandem duplications
regionsTrans	GRanges object with regions within the genome where to place the translocations
maxDups	Maximum number of repeats for tandem duplications
percCopiedIns	Percentage of copy-and-paste-like insertions (default: 0, i.e. no inserted sequences are duplicated)
percBalancedTrans	Percentage of balanced translocations (default: 1, i.e. all translocations are balanced)
bpFlankSize	Size of the each breakpoint's flanking regions, which may contain additional SNPs and/or indels
percSNPs	Percentage of SNPs within a breakpoint's flanking region
indelProb	Probability for an indel within a breakpoint's flanking region
maxIndelSize	Maximum size of an indel
repeatBias	If TRUE, the breakpoint positioning is biased towards repeat regions instead of a uniform distribution; turned off by default (see details below)
weightsMechanisms	Weights for SV formation mechanisms (see details and examples below)
weightsRepeats	Weights for repeat regions (see details and examples below)
repeatMaskerFile	Filename of a RepeatMasker output file
bpSeqSize	Length of the breakpoint sequences in the output
random	If TRUE, the SVs will be placed randomly within the genome or the given regions; otherwise, the given regions will be used as SV coordinates (random can also be a vector of five elements with TRUE/FALSE for every SV in the following order: deletions, insertions, inversions, duplications, translocations)
seed	Fixed seed for generation of random SV positions
verbose	If TRUE, some messages about the progress of the simulation will be printed into the R console

Details

About the supported SV types:

- Deletions: A segment is cut out from the genome.
- Insertions: A segment is cut or copied (see parameter percCopiedIns) from one chromosome A and inserted into another chromosome B.
- Inversions: A segment is cut out from one chromosome and its reverse complement is inserted at the same place without loss or a shift of sequence.
- Tandem duplication: A segment is duplicated one after the other at most maxDups times.
- Translocation: A segment from the 5' or 3' end of one chromosome A is exchanged with the 5' or 3' end of another chromosome B. If it is not balanced (see parameter percBalancedTrans), the segment from chromosome B will be lost, what results in a duplicated sequence from chromosome A. Segments translocated between two different ends (5'<->3' or 3'<->5') are always inverted.

About SV sizes and predefined regions:

- The region arguments (`regionsDels`,`regionsIns`, `regionsInvs`,`regionsDups`,`regionsTrans`) can be used in two ways: 1. as subsets of the genome where to place the SVs randomly or 2. as coordinates of SVs that shall be implemented at these exact positions. The latter can be useful to implement a predefined set of previously detected or known SVs. Set the parameter `random` to FALSE accordingly. In this case, the rownames of the region arguments will be used to name the SVs in the output.
- In case of insertions and translocations, where two genomic regions are involved, add columns "chrB" and "startB" for insertions and "chrB", "startB", "endB" for translocations to the `GRanges` objects in `regionsTrans` and `regionsIns`, respectively.
- It is recommended to set the size of an SV individually for every deletion, insertion, inversion or tandem duplication. See the function `estimateSVSizes` to estimate beta-distributed sizes from a training set of SVs (defaults for deletions, insertions, inversions and tandem duplications are available). Using the argument `size` to set the size for all SVs overrides all other size arguments.
- There is no size argument for translocations. After random generation of the breakpoint, the translocation spans the chromosome until the closest of both chromosome ends.

About biases towards SV formation mechanisms and repeat regions:

- When using the default genome hg19 and setting `repeatBias=TRUE`, RSVSim simulates a bias of breakpoint positioning towards certain kinds of repeat regions and regions of high homology. If `repeatBias=FALSE` (the default), the breakpoints will be placed uniformly across the whole genome.
- This is done in two steps: 1. Weighting SV formation mechanisms (here: NAHR, NHR, VNTR, TEI, Other) for each SV type. 2. Weighting each SV formation mechanism for each kind of repeat (supported: LINE/L1, LINE/L2, SINE/Alu, SINE/MIR, segmental duplications (SD), tandem repeats (TR), Random). The default weights were chosen from studies with SVs >1.000bp by Mills et al., Pang et al., Ou et al. and Hu et al. It is possible to change these weights by using the arguments weightsMechanisms and weightsRepeats, which need to have a certain data.frame structure (see vignette and examples below for details).
- For the mechanism NAHR, both breakpoints will lie within a repeat region, while for NHR, VNTR, TEI and Other, the repeat must make up for at least 75% of the SV region.
- This feature requires the coordinates of repeat regions for hg19. This can be handled in two ways: 1. When using this feature the first time, RSVSim downloads the coordinates once automatically from the UCSC Browser's RepeatMasker track (which may take up to 45 Minutes!) 2. The user may specify the filename of a RepeatMasker output file downloaded from their homepage: http://www.repeatmasker.org/species/homSap.html (e.g. hg19.fa.out.gz). In both cases, RSVSim saves the coordinates as RData object to the RSVSim installation directory for a faster access in the future.
- This feature is turned off automatically, when the user specifies his own genome. Then, breakpoints will be placed uniformly across the genome.

About additional breakpoint mutations:

- RSVSim allows to randomly generate additional SNPs and indels within the flanking regions of each breakpoint.

- By default, this feature is turned off. It is recommended to set the four arguments bpFlankSize, percSNPs, indelProb and maxIndelSize to use this feature.
- Each flanking region may only contain one indel, while insertions and deletions are equally likely. SNPs can affect 0-100% of the region.

Misc:

- By default, the human genome (hg19) will be used, which requires the package **BSgenome.Hsapiens.UCSC.hg19**.
- SVs will not be placed within unannotated regions or assembly gaps denoted by the letter "N". These regions are detected and filtered automatically.

Value

The rerranged genome as a `DNAStringSet`. Its `metadata` slot contains a named list of `data.frames` with information about the simulated SVs:

`deletions`	The coordinates of the implemented deletions and the breakpoint sequence
`insertions`	The coordinates of the origin (chrA) and destination (chrB) of the inserted sequence and the breakpoints sequences at both ends (5' and 3'). If the sequence is cut out from the original chromosome A, the sequence of this breakpoint is given as well.
`inversions`	The coordinates of the implemented inversions and the breakpoint sequences at both ends (5' and 3')
`tandemDuplications`	The coordinates of the duplicated sequence and the breakpoint sequence
`translocations`	The coordinates of the translocated sequences and the two breakpoint sequences (if balanced)

The coordinates in the tables refer to the "normal" reference genome.
All the list items can also be written to the specified output directory (which is the current directory by default). The genome will be saved in FASTA format and the SVs `data.frames` as CSV tables.

Author(s)

Christoph Bartenhagen

References

Chen W. et al., Mapping translocation breakpoints by next-generation sequencing, 2008, Genome Res, 18(7), 1143-1149. Lam H.Y. et al., Nucleotide-resolution analysis of structural variants using BreakSeq and a breakpoint library, 2010, Nat Biotechnol, 28(1), 47-55. Mills R.E. et al., Mapping copy number variation by population-scale genome sequencing, 2011, Nature, 470(7332), 59-65 Ou Z. et al., Observation and prediction of recurrent human translocations mediated by NAHR between nonhomologous chromosomes, 2011, Genome Res, 21(1), 33-46. Pang A.W. et al., Mechanisms of Formation of Structural Variation in a Fully Sequenced Human Genome, 2013, Hum Mutat, 34(2), 345-354. Smit et al., RepeatMasker Open-3.0., 1996-2010, <http://www.repeatmasker.org>

See Also

`estimateSVSizes`

Examples

```
## Toy example: Artificial genome with two chromosomes
genome = DNAStringSet(c("AAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTTT", "GGGGGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCCCCCC")
names(genome) = c("chr1","chr2")

## Three deletions of sizes 10bp each
sim = simulateSV(output=NA, genome=genome, dels=3, sizeDels=10, bpSeqSize=10)
sim
metadata(sim)

## Three insertions of 5bp each; all cut-and-paste-like (default)
sim = simulateSV(output=NA, genome=genome, ins=3, sizeIns=5, bpSeqSize=10)
sim
metadata(sim)
## Three insertions of 5bp each; all copy-and-paste-like (note the parameter \code{percCopiedIns})
sim = simulateSV(output=NA, genome=genome, ins=3, sizeIns=5, percCopiedIns=1, bpSeqSize=10)
sim
metadata(sim)

## Three inversions of sizes 2bp, 4bp and 6bp
sim = simulateSV(output=NA, genome=genome, invs=3, sizeInvs=c(2,4,6), bpSeqSize=10)
sim
metadata(sim)

## A tandem duplication of 4bp with at most ten duplications
## The duplication shall be placed somewhere within chr4:18-40
library(GenomicRanges)
region = GRanges(IRanges(10,30),seqnames="chr1")
sim = simulateSV(output=NA, genome=genome, dups=1, sizeDups=4, regionsDups=region, maxDups=10, bpSeqSize=10)
sim
metadata(sim)

## A balanced translocation (default)
sim = simulateSV(output=NA, genome=genome,trans=1, bpSeqSize=6, seed=246)
sim
metadata(sim)
## Another translocation, but unbalanced (note the parameter \code{percBalancedTrans})
sim = simulateSV(output=NA, genome=genome, trans=1, percBalancedTrans=0, bpSeqSize=6)
sim
metadata(sim)

## Simulate all four SV types at once:
## 2 deletions (5bp), 2 insertions (5bp),2 inversions (3bp), 1 tandem duplication (4bp), 1 translocations
sim = simulateSV(output=NA, genome=genome, dels=2, ins=2, invs=2, dups=1, trans=1, sizeDels=5, sizeIns=5, sizeInvs
sim
metadata(sim)

## Avoid random generation of coordinates and implement a given deletion of 10bp on chr2:16-25
knownDeletion = GRanges(IRanges(16,25), seqnames="chr2")
names(knownDeletion) = "myDeletion"
knownDeletion
sim = simulateSV(output=NA, genome=genome, regionsDels=knownDeletion, bpSeqSize=10, random=FALSE)
```

```
sim
metadata(sim)

## Avoid random generation of coordinates and implement a given insertion from chr1:16:25 at chr2:26
knownInsertion = GRanges(IRanges(16,25), seqnames="chr1", chrB="chr2", startB=26)
names(knownInsertion) = "myInsertion"
knownInsertion
sim = simulateSV(output=NA, genome=genome, regionsIns=knownInsertion, bpSeqSize=10, random=FALSE)
sim
metadata(sim)

## This example simulates a translocation t(9;22) leading to the BCR-ABL fusion gene.
## It uses simple breakpoints within 9q34.1 and 22q11.2 for demonstration
## Take care to add coordinates of both chromosomes to the GRanges object:
trans_BCR_ABL = GRanges(IRanges(133000000,141213431), seqnames="chr9", chrB="chr22", startB=23000000, endB=51304
names(trans_BCR_ABL) = "BCR_ABL"
trans_BCR_ABL
## This example requires the \pkg{BSgenome.Hsapiens.UCSC.hg19} which is used by default (hence, no genome argument)
## Not run: sim = simulateSV(output=NA, chrs=c("chr9", "chr22"), regionsTrans=trans_BCR_ABL, bpSeqSize=30, random=


## Add additional SNPs and indels at the flanking regions of each SV breakpoint:
## One deletion and 10% SNPs, 100% indel probability within 10bp up-/downstream of the breakpoint
sim = simulateSV(output=NA, genome=genome, dels=1, sizeDels=5, bpFlankSize=10, percSNPs=0.25, indelProb=1, maxInd
sim
metadata(sim)

## Setting the weights for SV formation mechanism and repeat biases demands a given data.frame structure
## The following weights are the default settings
## Please make sure your data.frames have the same row and column names, when setting your own weights
data(weightsMechanisms, package="RSVSim")
weightsMechanisms
data(weightsRepeats, package="RSVSim")
weightsRepeats
## The weights take effect, when no genome argument has been specified (i.e. the default genome hg19 will be used) an
## Not run: sim = simulateSV(output=NA, dels=10, invs=10, ins=10, dups=10, trans=10, repeatBias = TRUE, weightsMech
## If weightsMechanisms and weightsRepeats were omitted, RSVSim loads the default weights automatically (see detail
## Not run: sim = simulateSV(output=NA, dels=10, invs=10, ins=10, dups=10, trans=10, repeatBias = TRUE)
```

`weightsMechanisms` *Weights for SV formation mechanisms*

Description

The default weights of all supported SV formation mechanisms for all SVs. It is loaded automatically when the repeat bias feature has been turned on (`repeatBias=TRUE` in function `simulateSV`). (But this dataset is intended for internal use; no need to handle it manually.)

Usage

```
data("weightsMechanisms")
```

Format

Formal class 'data.frame'

Examples

```
data("weightsMechanisms")
weightsMechanisms
```

`weightsRepeats` *Weights for repeat region bias*

Description

The default weights of all supported repeat regions for all supported SV formation mechanisms. It is loaded automatically when the repeat bias feature has been turned on (`repeatBias=TRUE` in function `simulateSV`). (But this dataset is intended for internal use; no need to handle it manually.)

Usage

```
data("weightsRepeats")
```

Format

Formal class 'data.frame'

Examples

```
data("weightsRepeats")
weightsRepeats
```

Index

∗ **datagen**
simulateSV, 7
∗ **datasets**
segDups, 6
weightsMechanisms, 12
weightsRepeats, 13

compareSV, 2
compareSV,character,character-method *(*compareSV*)*, 2
compareSV,character,data.frame-method *(*compareSV*)*, 2
compareSV,data.frame,data.frame-method *(*compareSV*)*, 2

estimateSVSizes, 5, *10*
estimateSVSizes,numeric,missing,ANY,ANY,character-method *(*estimateSVSizes*)*, 5
estimateSVSizes,numeric,missing,ANY,ANY,missing-method *(*estimateSVSizes*)*, 5
estimateSVSizes,numeric,numeric,ANY,ANY,missing-method *(*estimateSVSizes*)*, 5

segDups, 6
simulateSV, *2*, *3*, *5*, 7
simulateSV,ANY *(*simulateSV*)*, 7
simulateSV,ANY-method *(*simulateSV*)*, 7

weightsMechanisms, 12
weightsRepeats, 13