Package 'ClonalSim'

May 6, 2026

Type Package

Title Simulation of Tumor Clonal Evolution with Realistic Sequencing Noise

Version 1.1.0

Description ClonalSim generates realistic mutational profiles of tumor samples with hierarchical clonal structure. It simulates founder, shared, and private mutations with biologically realistic noise models including intra-tumor heterogeneity (Beta distribution) and technical sequencing noise (negative binomial depth variation, binomial read sampling, base errors). The package is designed for benchmarking variant callers, testing clonal deconvolution algorithms, and teaching tumor heterogeneity concepts.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.6.0)

Imports methods, stats, utils, ggplot2, tidyr, rlang, GenomicRanges, IRanges, S4Vectors, VariantAnnotation

Suggests testthat (>= 3.0.0), knitr, rmarkdown, BiocStyle

biocViews Software, Sequencing, SomaticMutation, VariantDetection, Coverage, Visualization, DataImport

VignetteBuilder knitr

RoxygenNote 7.3.3

BugReports https://github.com/gbucci/ClonalSim/issues

URL https://github.com/gbucci/ClonalSim

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

git_branch devel

git_last_commit 6048f4b

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-06

Author Gabriele Bucci [aut, cre] (ORCID: <https://orcid.org/0000-0001-9838-7204>)

Maintainer Gabriele Bucci <bucci.g@gmail.com>

Contents

`ClonalSim-package` *ClonalSim: Simulation of Tumor Clonal Evolution with Realistic Sequencing Noise*

Description

ClonalSim generates realistic mutational profiles of tumor samples with hierarchical clonal structure. It simulates founder, shared, and private mutations with biologically realistic noise models including intra-tumor heterogeneity (Beta distribution) and technical sequencing noise (negative binomial depth variation, binomial read sampling, base errors). The package is designed for benchmarking variant callers, testing clonal deconvolution algorithms, and teaching tumor heterogeneity concepts.

ClonalSim generates realistic mutational profiles of tumor samples with hierarchical clonal structure. It simulates founder, shared, and private mutations with biologically realistic noise models.

Details

The package implements a two-stage noise model:

1. Biological Noise (Intra-Tumor Heterogeneity):

- Uses Beta distribution (more realistic than Gaussian for frequencies)
- Models spatial/temporal heterogeneity within tumor
- Configurable concentration parameter controls variability

2. Technical Sequencing Noise:

- **Depth overdispersion**: Negative binomial distribution (models GC bias, mappability, PCR artifacts)
- **Stochastic read sampling**: Binomial distribution (each read independently sampled)
- **Base calling errors**: Illumina-like error rates

Main Functions

- `simulateTumor`: Main simulation function
- `ClonalSimData`: S4 class for results
- `toGRanges`: Export to GenomicRanges
- `toVCF`: Export to VCF format
- `toPyClone`: Export for PyClone analysis
- `toSciClone`: Export for SciClone analysis

Use Cases

- **Benchmarking**: Test variant callers and mutation detection pipelines
- **Algorithm Development**: Test clonal deconvolution algorithms
- **Education**: Teach tumor heterogeneity concepts
- **Method Validation**: Positive controls for analysis pipelines

Author(s)

Maintainer: Gabriele Bucci <bucci.g@gmail.com> (ORCID)

References

McGranahan N, Swanton C. Clonal Heterogeneity and Tumor Evolution: Past, Present, and the Future. Cell. 2017.

Dentro SC, Wedge DC, Van Loo P. Principles of Reconstructing the Subclonal Architecture of Cancers. Cold Spring Harb Perspect Med. 2017.

Roth A, et al. PyClone: statistical inference of clonal population structure in cancer. Nat Methods. 2014.

Miller CA, et al. SciClone: inferring clonal architecture and tracking the spatial and temporal patterns of tumor evolution. PLoS Comput Biol. 2014.

See Also

Useful links:

- https://github.com/gbucci/ClonalSim
- Report bugs at https://github.com/gbucci/ClonalSim/issues

Examples

```
# Basic simulation
sim <- simulateTumor()
sim

# Access results
mutations <- getMutations(sim)
head(mutations)

# Visualize
plot(sim, type = "vaf_density")

# Export to Bioconductor
gr <- toGRanges(sim)

# Custom simulation: low purity tumor
sim_low_purity <- simulateTumor(
  subclone_freqs = c(0.1, 0.15, 0.15),  # 40% purity
  n_mut_per_clone = c(20, 25, 30)
)

# High coverage sequencing
sim_deep <- simulateTumor(
  subclone_freqs = c(0.3, 0.4, 0.3),
  n_mut_per_clone = c(30, 40, 30),
  sequencing_noise = list(mean_depth = 500, depth_dispersion = 100)
)
```

---

applyBiologicalNoise *Apply biological noise using Beta distribution*

Description

Simulates intra-tumor heterogeneity using a Beta distribution. The Beta distribution is more realistic than Gaussian for frequency data as it is naturally bounded between 0 and 1.

Usage

```
applyBiologicalNoise(true_freq, n_mutations, concentration = 50)
```

Arguments

`true_freq`	numeric, the true allele frequency (0-1)
`n_mutations`	integer, number of mutations to generate
`concentration`	numeric, concentration parameter for Beta distribution (alpha + beta). Higher values = less variability. Default: 50

Details

For a clone with true frequency `f`, VAF values are sampled from Beta(alpha, beta) where:

- alpha = f * concentration
- beta = (1-f) * concentration

Higher concentration values result in VAF closer to the expected frequency, while lower values increase spread, simulating spatial/temporal heterogeneity.

Value

numeric vector of VAF values with biological noise

Examples

```
# Generate 100 mutations with true frequency 0.3 and moderate heterogeneity
vaf <- applyBiologicalNoise(0.3, 100, concentration = 50)
hist(vaf, main = "VAF with biological noise", xlab = "VAF")

# High heterogeneity (low concentration)
vaf_high_het <- applyBiologicalNoise(0.3, 100, concentration = 20)

# Low heterogeneity (high concentration)
vaf_low_het <- applyBiologicalNoise(0.3, 100, concentration = 100)
```

`ClonalSimData` *Constructor for ClonalSimData*

Description

Constructor for ClonalSimData

Usage

```
ClonalSimData(
  mutations = data.frame(),
  params = list(),
  clonal_structure = data.frame(),
  metadata = list()
)
```

Arguments

`mutations`	data.frame with mutation data
`params`	list with simulation parameters
`clonal_structure`	data.frame with clone hierarchy
`metadata`	list with additional metadata

Value

ClonalSimData object

Examples

```
# Typically created by simulateTumor(), but can be constructed manually
mutations <- data.frame(
  Mutation = "mut1",
  Chromosome = "chr1",
  Position = 1000000,
  Ref = "A",
  Alt = "T",
  True_VAF = 0.5,
  VAF = 0.48,
  Depth = 100,
  Alt_reads = 48,
  Clone = "Clone1",
  Type = "private"
)
params <- list(subclone_freqs = c(0.5))
sim_data <- ClonalSimData(mutations = mutations, params = params)
```

ClonalSimData-class *ClonalSimData Class*

Description

S4 class to store results from tumor clonal evolution simulation

Arguments

`object`	ClonalSimData object

Value

TRUE if valid, otherwise error message

Slots

`mutations` data.frame containing mutation information with columns: Mutation, Chromosome, Position, Ref, Alt, True_VAF, VAF, Depth, Alt_reads, Clone, Type, Clone_IDs

`params` list containing simulation parameters: subclone_freqs, n_mut_per_clone, n_mut_founder, n_mut_shared, biological_noise, sequencing_noise

`clonal_structure` data.frame defining hierarchical clone relationships

`metadata` list containing additional metadata (date, version, seed)

Examples

```
# Create a simple simulation
sim <- simulateTumor(
  subclone_freqs = c(0.3, 0.4, 0.3),
  n_mut_per_clone = c(30, 40, 30)
)

# Access mutations
head(getMutations(sim))

# Access parameters
getSimParams(sim)
```

---

getClonalStructure *Get clonal structure*

Description

Get clonal structure

Usage

```
getClonalStructure(object)
```

Arguments

object	ClonalSimData object

Value

data.frame defining clone hierarchy

Examples

```
sim <- simulateTumor()
structure <- getClonalStructure(sim)
print(structure)
```

---

getMetadata *Get metadata*

Description

Get metadata

Usage

```
getMetadata(object)
```

Arguments

object	ClonalSimData object

Value

list with metadata

Examples

```
sim <- simulateTumor()
metadata <- getMetadata(sim)
metadata$date
```

---

getMutations *Get mutations data frame*

Description

Get mutations data frame

Usage

```
getMutations(object)
```

Arguments

object	ClonalSimData object

Value

data.frame with mutation information

Examples

```
sim <- simulateTumor()
mutations <- getMutations(sim)
head(mutations)
```

getObservedVAF *Get observed VAF values (with sequencing noise)*

Description

Get observed VAF values (with sequencing noise)

Usage

```
getObservedVAF(object)
```

Arguments

object	ClonalSimData object

Value

numeric vector of observed VAF values

Examples

```
sim <- simulateTumor()
obs_vaf <- getObservedVAF(sim)
summary(obs_vaf)
```

getSimParams *Get simulation parameters*

Description

Get simulation parameters

Usage

```
getSimParams(object)
```

Arguments

object	ClonalSimData object

Value

list with simulation parameters

Examples

```
sim <- simulateTumor()
params <- getSimParams(sim)
params$subclone_freqs
```

getTrueVAF *Get true VAF values (biological, without sequencing noise)*

Description

Get true VAF values (biological, without sequencing noise)

Usage

```
getTrueVAF(object)
```

Arguments

object	ClonalSimData object

Value

numeric vector of true VAF values

Examples

```
sim <- simulateTumor()
true_vaf <- getTrueVAF(sim)
summary(true_vaf)
```

plot,ClonalSimData,missing-method *Plot method for ClonalSimData*

Description

Plot method for ClonalSimData

Usage

```
## S4 method for signature 'ClonalSimData,missing'
plot(x, y, type = "vaf_density", ...)
```

Arguments

x	ClonalSimData object
y	unused (for S4 compatibility)
type	character indicating plot type: "vaf_density", "vaf_scatter", "depth_histogram", "clone_matrix". Default is "vaf_density"
...	additional arguments passed to plotting functions

Details

Note: You need to load ggplot2 explicitly before using plot(): `library(ggplot2)`

Value

ggplot2 object

Examples

```
library(ggplot2)
sim <- simulateTumor()
plot(sim, type = "vaf_density")
plot(sim, type = "vaf_scatter")
```

---

print.summary.ClonalSimData *Print summary method*

Description

Print summary method

Usage

```
## S3 method for class 'summary.ClonalSimData'
print(x, ...)
```

Arguments

x	summary.ClonalSimData object
...	additional arguments (unused)

Value

NULL (prints to console)

---

show,ClonalSimData-method *Show method for ClonalSimData*

Description

Show method for ClonalSimData

Usage

```
## S4 method for signature 'ClonalSimData'
show(object)
```

Arguments

object	ClonalSimData object

Value

NULL (prints to console)

Examples

```
sim <- simulateTumor()
show(sim)
```

`simulateDepth` *Simulate realistic sequencing depth*

Description

Simulates sequencing coverage with realistic overdispersion using negative binomial distribution, which better captures real NGS variability compared to Poisson.

Usage

```
simulateDepth(
  n_mutations,
  mean_depth = 100,
  distribution = "negative_binomial",
  dispersion = 20
)
```

Arguments

`n_mutations`	integer, number of mutations
`mean_depth`	numeric, mean sequencing coverage. Default: 100
`distribution`	character, distribution type: "negative_binomial" (realistic, default), "poisson" (simpler), or "uniform" (for testing)
`dispersion`	numeric, size parameter for negative binomial. Lower values = more variable coverage. Default: 20

Details

The negative binomial distribution allows overdispersion (variance > mean), which occurs in real sequencing due to:

- GC content bias
- Mappability differences
- PCR amplification artifacts

The dispersion parameter controls variability: lower values produce more variable coverage across positions.

Value

integer vector of depth values

Examples

```
# Realistic depth with overdispersion
depth_realistic <- simulateDepth(1000, mean_depth = 100, dispersion = 20)
hist(depth_realistic, main = "Realistic depth", xlab = "Depth")

# More variable coverage (low dispersion)
depth_variable <- simulateDepth(1000, mean_depth = 100, dispersion = 10)

# Uniform coverage (for testing)
depth_uniform <- simulateDepth(1000, mean_depth = 100, distribution = "uniform")
```

---

simulateSequencingReads

Simulate sequencing reads with binomial sampling

Description

Simulates stochastic read counts using binomial distribution, which models the independent sampling of each sequencing read.

Usage

```
simulateSequencingReads(true_vaf, depth, error_rate = 0.001)
```

Arguments

true_vaf	numeric vector, true variant allele frequencies
depth	integer vector, sequencing depth at each position
error_rate	numeric, base miscall rate (0-1). Default: 0.001 (0.1 typical for Illumina)

Details

Each sequencing read is independently sampled from the DNA pool. Given a true VAF and depth D, the number of alternative reads follows a binomial distribution: alt_reads ~ Binomial(D, VAF).

This introduces natural sampling variation, with higher uncertainty at:

- Low sequencing depth
- Low variant frequency

The error_rate parameter simulates base miscalls from sequencer optical/ chemistry errors.

Value

list with three components:

- vaf: observed VAF values
- alt_reads: alternative allele read counts
- ref_reads: reference allele read counts

Examples

```
# Simulate reads for mutations at VAF 0.3 with depth 100
true_vaf <- rep(0.3, 100)
depth <- rep(100, 100)
reads <- simulateSequencingReads(true_vaf, depth)

# Observed VAF varies due to sampling
hist(reads$vaf, main = "Observed VAF", xlab = "VAF")
abline(v = 0.3, col = "red", lty = 2)

# Check alt_reads distribution
hist(reads$alt_reads, main = "Alt read counts", xlab = "Alt reads")
```

`simulateTumor` *Simulate tumor clonal evolution with realistic noise*

Description

Main function to simulate a heterogeneous tumor sample with hierarchical clonal structure, including realistic biological and technical sequencing noise.

Usage

```
simulateTumor(
  subclone_freqs = c(0.15, 0.25, 0.3, 0.3),
  n_mut_per_clone = c(20, 25, 30, 15),
  n_mut_founder = 10,
  n_mut_shared = list(`2 3 4` = 15, `3 4` = 12, `1 2` = 8),
  biological_noise = list(enabled = TRUE, concentration = 50),
  sequencing_noise = list(enabled = TRUE, mean_depth = 100, depth_variation =
   "negative_binomial", depth_dispersion = 20, error_rate = 0.001, binomial_sampling =
    TRUE),
  germline_variants = list(enabled = FALSE, n_variants = 50, vaf_expected = 0.5)
)
```

Arguments

`subclone_freqs`	numeric vector, frequencies of each subclone (must sum to <= 1). Default: c(0.15, 0.25, 0.30, 0.30)
`n_mut_per_clone`	integer vector, number of private mutations per clone. Default: c(20, 25, 30, 15)
`n_mut_founder`	integer, number of founder mutations (present in all clones). Default: 10
`n_mut_shared`	named list, shared mutations between clone groups. Names indicate which clones (e.g., "2 3 4"), values are mutation counts. Default: list("2 3 4" = 15, "3 4" = 12, "1 2" = 8)
`biological_noise`	list with biological noise parameters: • enabled: logical, enable biological noise (default: TRUE) • concentration: numeric, Beta distribution concentration (default: 50)

`sequencing_noise`
: list with sequencing noise parameters:
  - enabled: logical, enable sequencing noise (default: TRUE)
  - mean_depth: numeric, average coverage (default: 100)
  - depth_variation: character, distribution type (default: "negative_binomial")
  - depth_dispersion: numeric, dispersion parameter (default: 20)
  - error_rate: numeric, base miscall rate (default: 0.001)
  - binomial_sampling: logical, use binomial sampling (default: TRUE)

`germline_variants`
: list with germline variant parameters:
  - enabled: logical, include germline variants (default: FALSE)
  - n_variants: integer, number of germline variants to simulate (default: 50)
  - vaf_expected: numeric, expected VAF for heterozygous germline (default: 0.5)

  Germline variants will have VAF adjusted by tumor purity: observed_VAF = germline_vaf * tumor_purity + 0.5 * (1 - tumor_purity)

Value

ClonalSimData object containing mutations, parameters, and metadata

Examples

```
# Basic simulation with default parameters
sim <- simulateTumor()
show(sim)
plot(sim)

# Custom clonal structure
sim <- simulateTumor(
  subclone_freqs = c(0.2, 0.3, 0.5),
  n_mut_per_clone = c(30, 40, 30)
)

# Low purity tumor (40% purity)
sim <- simulateTumor(
  subclone_freqs = c(0.1, 0.15, 0.15),
  n_mut_per_clone = c(20, 25, 30)
)

# High coverage sequencing
sim <- simulateTumor(
  sequencing_noise = list(
    enabled = TRUE,
    mean_depth = 500,
    depth_dispersion = 100
  )
)

# No noise (ideal data for testing)
sim <- simulateTumor(
  biological_noise = list(enabled = FALSE),
  sequencing_noise = list(enabled = FALSE)
)
```

```
# Include germline variants
sim <- simulateTumor(
  subclone_freqs = c(0.3, 0.4),  # 70% purity
  n_mut_per_clone = c(40, 50),
  germline_variants = list(enabled = TRUE, n_variants = 100)
)
```

---

summary,ClonalSimData-method *Summary method for ClonalSimData*

Description

Summary method for ClonalSimData

Usage

```
## S4 method for signature 'ClonalSimData'
summary(object)
```

Arguments

object	ClonalSimData object

Value

list with summary statistics

Examples

```
sim <- simulateTumor()
summary(sim)
```

---

toDataFrame *Export mutation data to data frame*

Description

Simple export to data.frame (useful for compatibility with other tools like PyClone, SciClone)

Usage

```
toDataFrame(object, file = NULL, include_true_vaf = TRUE)
```

Arguments

object	ClonalSimData object
file	character, optional file path to write CSV. Default: NULL
include_true_vaf	logical, include True_VAF column. Default: TRUE

Value

data.frame with mutation data

Examples

```
sim <- simulateTumor()
df <- toDataFrame(sim)
head(df)

# Export to CSV (use tempdir to avoid writing to user's filesystem)
csv_file <- tempfile(fileext = ".csv")
toDataFrame(sim, file = csv_file)
```

---

toGRanges *Convert ClonalSimData to GRanges object*

Description

Converts mutation data to a GenomicRanges::GRanges object, the standard Bioconductor data structure for genomic intervals.

Usage

```
toGRanges(object, include_metadata = TRUE)
```

Arguments

object	ClonalSimData object
include_metadata	logical, include all metadata columns. Default: TRUE

Value

GRanges object with mutation information

Examples

```
sim <- simulateTumor()
gr <- toGRanges(sim)
print(gr)

# Access metadata columns
GenomicRanges::mcols(gr)

# Subset by chromosome
gr_chr1 <- gr[GenomicRanges::seqnames(gr) == "chr1"]
```

toPyClone *Convert to PyClone input format*

Description

Exports data in format suitable for PyClone clonal deconvolution analysis.

Usage

```
toPyClone(object, file, sample_id = "sample1")
```

Arguments

object	ClonalSimData object
file	character, file path to write TSV
sample_id	character, sample identifier. Default: "sample1"

Value

data.frame formatted for PyClone (invisibly)

Examples

```
sim <- simulateTumor()
pyclone_file <- tempfile(fileext = ".tsv")
toPyClone(sim, file = pyclone_file)
```

toSciClone *Convert to SciClone input format*

Description

Exports data in format suitable for SciClone clonal analysis.

Usage

```
toSciClone(object, file)
```

Arguments

object	ClonalSimData object
file	character, file path to write TSV

Value

data.frame formatted for SciClone (invisibly)

Examples

```
sim <- simulateTumor()
sciclone_file <- tempfile(fileext = ".tsv")
toSciClone(sim, file = sciclone_file)
```

toVCF *Convert ClonalSimData to VCF format*

Description

Exports mutation data to Variant Call Format (VCF), the standard format for variant data. Returns a VariantAnnotation::VRanges object which can be written to VCF file.

Usage

```
toVCF(object, sample_name = "TumorSample", output_file = NULL)
```

Arguments

object	ClonalSimData object
sample_name	character, sample name for VCF. Default: "TumorSample"
output_file	character, optional file path to write VCF. If NULL, returns VRanges object without writing. Default: NULL

Value

VRanges object (invisibly if output_file is specified)

Examples

```
sim <- simulateTumor()

# Get VRanges object
vr <- suppressWarnings(toVCF(sim))
print(vr)

# Write to VCF file (use tempdir to avoid writing to user's filesystem)
vcf_file <- tempfile(fileext = ".vcf")
suppressWarnings(toVCF(sim, output_file = vcf_file))
```

Index