Package 'CloneSeeker'

September 25, 2026

Version 1.0.17

Date 2026-09-26

Title Seeking and Finding Clones in Copy Number and Sequencing Data

Description Defines the classes and functions used to simulate and to analyze data sets describing copy number variants and, optionally, sequencing mutations in order to detect clonal subsets. See Zucker et al. (2019) <doi:10.1093/bioinformatics/btz057>.

Depends R (>= 3.0)

Imports methods, graphics, combinat, mc2d, quantmod

License Apache License (== 2.0)

URL http://silicovore.com/OOMPA/clonefinder.html

NeedsCompilation no

Author Mark Zucker [aut],
Kevin R. Coombes [aut, cre]

Maintainer Kevin R. Coombes <krc@silicovore.com>

Repository CRAN

Date/Publication 2026-09-25 15:30:02 UTC

Contents

Seeking Clones 2
Simplices 5
Simulating Clones 6
Tumor-class 8
WeightVector-class 9

Index **11**

Seeking Clones *Seeking Tumor Clones From Data*

Description

Starting with copy number segmentation data and/or sequencing mutation data for a tumor, seek the number of clones, the fraction of cells belonging to each clone, and the likely set of abnormalities in each clone.

Usage

```
seekClones(cndata, vardata, cnmodels, psiset, pars, imputedCN = NULL)
runAlg(...)
```

Arguments

cndata	A data frame with seven columns; can also be NULL. The names of the required columns are enumerated in the man page for generateTumorData. These are the same as the output typically produced by the DNAcopy algorithm, where each row represents a segment (a contiguous region of a chromosome) where the copy number is believed to be constant.
vardata	A data frame with eight columns; can also be NULL. The names of the required columns are enumerated in the man page for generateTumorData. These are typical of the output of a DNA sequencing experiment that has been processed to identify variants, which may be either germline or somatic.
cnmodels	A matrix. Each row represents a model to be considered; each column represents a clone. The entries are integers specifying the number of DNA copies present in that clone. See details.
psiset	A matrix. Each column represents a clone, and each row represents a different possible model of the fraction of cells per clone. See details.
pars	A list of algorithm parameters; see details.
imputedCN	a logical value; if missing, should the copy number be imputed from the mutation data.
...	additional variables

Details

The algorithm starts with an initial set of 'psi' parameters (representing the fraction of tumor belonging to each clone). It computes the best (maximum a posteriori) clonal copy number and/or number of mutated alleles for each clone for each segment/mutation, conditional on the data and each of the initial psi vectors. It then computes the posterior probability for each psi-vector and its computed copy number and mutation parameters. It uses these posterior probabilities to resample new possible psi-vectors. The process repeats iteratively, and with each iterations obtains a better estimate of psi and the clonal segment copy number and mutation assignments until it terminates.

The set of copy number models that we use is typically generated using the following command: `as.matrix(expand.grid(lapply(1:5, function(i){0:5})))` This setup considers all (7776) possible models with up to five clones, where the copy number for each clone ranges from 0 to 5. (In the future, we are likely to make this the default; right now, you have to generate these models yourself.)

The set of possible psi-vectors (that is, the fraction of cells allocated to each clone) that we use is typically generated using the following command: `psis.20 <- generateSimplex(20,5)` This setup considers all (192) possible divisions of the tumor into up to five clones, where the fraction of cells per clone is any possible multiple of 0.05. Each row is sorted to put the most abundant clones first, which makes it easier to identify specific clones, except in the rare case when two clones contain exactly the same fraction of cells. (In the future, we are likely to make this the default; right now, you have to generate these models yourself.)

The object `pars` is a list of numerical algorithm parameters. The elements are:

sigma0 The standard deviation of measured allelic copy number at the SNP level.

ktheta The probability parameter of the geometric prior distribution on `K`, the number of clones.

theta The probability parameter of the geometric prior distribution on genomic copy number.

mtheta The probability parameter of the geometric prior distribution on the occurence of point mutations.

alpha The (repeated) alpha parameter of a symmetric Dirichlet distributed prior on the fractions of cells belong to each clone; default value is 0.5, giving a Jeffreys Prior.

thresh The threshold determining the smallest possible detectable clone.

cutoff SNP array segments with fewer markers than this are excluded.

Q Determines the number of new `psi` vectors resampled from the estimated posterior probability distribution at each iteration of the algorithm

iters The number of iterations in the algorithm.

The default settings we used are from commonly used unfinformative priors (e.g., alpha=0.5 for the Dirichlet distribution is the Jeffreys Prior) or based on empirical assessments of the variation in data (sigma0, for example, which describes variation in SNP array data).

Note that `runAlg` (an alias for `seekClones`) is DEPRECATED.

Value

The `seekClones` function returns a (rather long) list containing:

`psi`	The most likely posterior psi-vector, given the data. The number of non-zero entries is the number of clones found, and the non-zero entries are the fraction of cells per clone
`A`	The most likely copy numbers for the A allele in each segment in each clone.
`B`	The most likely copy numbers for the B allele in each segment in each clone.
`psibank`	A matrix, where each row is one of the psi-vectors considered during the analysis.
`psiPosts`	A numeric vector, the (marginal) posterior probability of each psi-vector considered during the analysis.

indices	???
data	a list with two data-frame components containing the data used during the analysis.
filtered.data	a list with two data-frame components containing the filtered data used during the analysis. Filtering removes non-informative segments that have normal copy number or contain only germline mutations.
etaA	A vector of the weighted average allelic copy number for the 'A-Allele' at each segment (that is, the sum of the clonal A-allelic copy number values multiplied by the fraction of the tumor made up by each clone)
etaB	A vector of the weighted average allelic copy number for the 'B-Allele' at each segment
etaM	A vector of the weighted average number of copies of the mutated allele at each mutation
mutated	A matrix of the number of mutated alleles at each locus in each clone, where the number of rows is the number of somatic mutations in the data and the number of columns is the number of clones

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Mark Zucker <zucker.64@buckeyemail.osu.edu>

References

Zucker MR, Abruzzo LV, Herling CD, Barron LL, Keating MJ, Abrams ZB, Heerema N, Coombes KR. Inferring Clonal Heterogeneity in Cancer using SNP Arrays and Whole Genome Sequencing. Bioinformatics. To appear. doi: 10.1093/bioinformatics/btz057.

Examples

```
# set up models
psis.20 <- generateSimplex(20,5)
cnmodels <- as.matrix(expand.grid(lapply(1:5, function(i){ 0:5 })))
# set up algortihm parameters
pars <- list(sigma0=5, theta = 0.9, ktheta = 0.3, mtheta = 0.9,
             alpha = 0.5, thresh = 0.04, cutoff = 100, Q = 100, iters = 4)
# create a tumor
psis <- c(0.6, 0.3, 0.1) # three clones
tumor <- Tumor(psis, rounds = 100, nu = 0, pcnv = 1, norm.contam = FALSE)
# simulate a dataset
dataset <- generateTumorData(tumor, 10000, 600000, 70, 25, 0.15, 0.03, 0.1)
result <- seekClones(dataset$cn.data, dataset$seq.data,
             cnmodels, psis.20, pars = pars, imputedCN = NULL)
```

Simplices *Simplices and Clonal Fractions*

Description

Utility functions for working with vectors of clonal fractions.

Usage

```
sampleSimplex(n, d = 5)
generateSimplex(k, d, reps = 1)
```

Arguments

d	an integer, the dimension of the simplex, or the number of clones.
n	an integer, the number of vectors to sample randomly.
k	an integer, the number of equally spaced points to select along each side of the simplex while constructing a lattice.
reps	an integer, the number of times to repeat the lattice.

Details

When studying the clonal subpopulations of a tumor sample, we frequently need access to vectors that contain the fraction of cells belonging to each clone. These vectors are characterized by the fact that each entry is nonzero and they must add up to 1. The set of such vectors/points in d-dimensional space defines the "d-simplex". The functions defined here allow us to work with d-simplices, either by randomly sampling vectors (`sampleSimplex`) or by systematically filling the space with a regular lattice (`generateSimplex`).

Value

Both functions return a matrix with `d` columns. Each row contains nonzeo real numbers that sum to 1. The `generateSimplex` function ensures that (a) each row is unique and (b) the entries in each row appear in decreasing order.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Mark Zucker <zucker.64@buckeyemail.osu.edu>

Examples

```
sampleSimplex(5, 3)
generateSimplex(5, 3)
```

Simulating Clones *Simulating Tumor Clones*

Description

Simulating copy number segmentation data and sequencing mutation data for tumors composed of multiple clones.

Usage

```
generateTumorData(tumor, snps.seq, snps.cgh, mu, sigma.reads,
                  sigma0.lrr, sigma0.baf, density.sigma)
plotTumorData(tumor, data)
tumorGen(...)
dataGen(tumor, ...)
```

Arguments

`tumor`	an object of the `Tumor` class.
`snps.seq`	an integer; the total number of germline variants and somatic mutations to simulate in the tumor genome.
`snps.cgh`	an integer; the number of single nucleotide polymorphisms (SNPs) to simulate as meaurements made to estimate copy number.
`mu`	an integer; the average read depth of a simulated sequencing study giving rise to mutations.
`sigma.reads`	a real number; the standard deviation of the number of simulated sequencing reads per base.
`sigma0.lrr`	a real number; the standard deviation of the simulated per-SNP log R ratio (LRR) for assessing copy number.
`sigma0.baf`	a real number; the standard deviation of the simulated B allele frequency (BAF) for assessing copy number.
`density.sigma`	a real number; the standard deviation of a beta distribution used to simulate the number of SNP markers per copy number segment.
`data`	a list containing two data frames, `cn.data` and `seq.data`, as produced by `generateTumorData`.
`...`	additional variables

Details

Copy number and mutation data are simulated essentially independently. Each simulation starts with a single "normal" genome, and CNVs and/or mutations are randomly generated for each new "branch" or subclone. (The number of subclones depends on the input parameters.) Each successive branch is randomly determined to descend from one of the existing clones, and therefore contains both the aberrations belonging to its parent clone and the novel aberrations assigned to it. Depending on input parameters, the algorithm can also randomly select some clones for extinction in the process of generating the heterogeneous tumor, to yield a more realistic population structure.

Note that `tumorGen` (an alias for `Tumor` that returns a list instead of a Tumor object) and `dataGen` (an alias for `generateTumorData`) are DEPRECATED.

Value

The `generateTumorData` function returns a list with two components, `cn.data` and `seq.data`. Each component is itself a data frame. Note that in some cases, one of these data frames may have zero rows or may be returned as an `NA`.

The `cn.data` component contains seven columns:

`chr` the chromosome number;

`seq` a unique segment identifier;

`LRR` simulated segment-wise log ratios;

`BAF` simulated segment-wise B allele frequencies;

`X` **and** `Y` simulated intensities for two separate alleles/haplotypes per segment; and

`markers` the simulated number of SNPS per segment.

The `seq.data` component contains eight columns:

`chr` the chromosome number;

`seq` a unique "segment" identifier;

`mut.id` a unique mutation identifier;

`refCounts` **and** `varCounts` the simulated numbers of reference and variant counts per mutation;

`VAF` the simulated variant allele frequency;

`totalCounts` the simulated total number of read counts; and

`status` a character (that should probably be a factor) indicating whether a variant should be viewed as somatic or germline.

The `plotTumorData` function invisibly returns its `data` argument.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Mark Zucker <zucker.64@buckeyemail.osu.edu>

Examples

```
psis <- c(0.6, 0.3, 0.1) # three clones
# create tumor with copy number but no mutation data
tumor <- Tumor(psis, rounds = 400, nu = 0, pcnv = 1, norm.contam = FALSE)
# simulate the dataset
dataset <- generateTumorData(tumor, 10000, 600000, 70, 25, 0.15, 0.03, 0.1)
#plot it
plotTumorData(tumor, dataset)
```

Tumor-class *Class "Tumor"*

Description

A class that represents tumors, thought of as a collection of (sub)clones each with an associated measure as a fraction of all tumor cells.

Usage

```
Tumor(psi, rounds, nu = 100, pcnv = 0.5, norm.contam = FALSE, cnmax = 4)
getClone(tumor, i)
```

Arguments

psi	a numeric vector containing non-negative values.
rounds	an integer; the number of generations through which to evolve the potential clones.
nu	an integer; the expected number of mutations in each clonal generation.
pcnv	a real number between 0 and 1; the probability of a CNV occuring per generation.
norm.contam	a logical value; should we treat one of the cell populations as normal cells that are "contaminating" the tumor specimen?
cnmax	an integer, the maximum copy number allowed in the simulated data.
tumor	an object of the Tumor class.
i	a integer; which clone to extract.

Details

The Tumor class is used to represent compex tumors, each of which consists a set of subclones representing different fractional parts of the tumor. Each clone is characterized by a set of copy number variants (modeled by the output produced by something like the DNAcopy package) and, optionally, a set of sequence mutations. Each of these genetic events is mapped to a specific interval or point in the human genome.

In the current implementation, a Tumor consists of a weight vector that specifies the fractions of cells for each clone and a list of clones. At present, each clone is itself a list containing one (if there are no mutations) or two (if there are both copy number variants and mutations) data frames. This structure is likely to change in later versions of the package, since we expect to implement a full-fledged S4 class to represent clones. So, one should not rely on the current implementation.

Value

The constructor returns a valid object of the Tumor class.

Objects from the Class

Although objects can be created using new, the preferred method is to use the constructor function, Tumor.

Slots

psi a WeightVector containing non-negative values whose sum equals one.

clones a list, each of whose elements represents a clone.

Methods

coerce(from, to, strict = TRUE) Convert the Tumor object into a simple numeric vector. Never actually used in this form, since the preferred method is to write as(WV, "list").

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Mark Zucker <zucker.64@buckeyemail.osu.edu>

See Also

WeightVector

Examples

```
showClass("Tumor")

tumor <- Tumor(c(0.5, 0.3, 0.2), 100)
simpleTumor <- as(tumor, "list")
reformed <- as(simpleTumor, "Tumor")
```

WeightVector-class *Class "WeightVector"*

Description

A class, with validity checking, to contain vectors of non-negative real numbers whose sum equals one.

Usage

```
WeightVector(phi)
```

Arguments

phi	a numeric vector containing non-negative values.

Details

When trying to simulate or model biological data from (sub)clonal populations of cells, we need vectors that keep track of the fraction of cells belonging to each clone. These vectors can only contain non-negative entries, and the entries must add up to one. (We thought about calling these things "ClonalFractions", but that seems overly specialized for a notion that is likely to prove useful in other contexts.) Such vectors of length d can also be viewed as points of a d-dimensional simplex.

We have implemented `WeightVectors` as an S4 class, primarily so we can enforce the defining properties. We also expect this design to make it easier to use them as slots in other classes.

Value

The constructor returns a valid object of the `WeightVector` class.

Objects from the Class

Although objects can be created using `new`, the preferred method is to use the constructor function, `WeightVector`.

Slots

`psi` a vector containing non-negative values whose sum equals one.

Methods

coerce(from, to, strict = TRUE) Convert the `WeightVector` object into a simple numeric vector. Never actually used in this form, since the preferred method is to write `as(WV, "numeric")`.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Mark Zucker <zucker.64@buckeyemail.osu.edu>

See Also

Simplices

Examples

```
showClass("WeightVector")

WeightVector(1:4)
as(WeightVector(c(2, 3, 5)), "numeric")
```

Index

∗ **array**
    Seeking Clones, 2
    Simplices, 5
    Simulating Clones, 6
∗ **classes**
    Tumor-class, 8
    WeightVector-class, 9

coerce,Tumor,list-method *(*Tumor-class*)*, 8
coerce,WeightVector,numeric-method *(*WeightVector-class*)*, 9

dataGen *(*Simulating Clones*)*, 6

generateSimplex *(*Simplices*)*, 5
generateTumorData *(*Simulating Clones*)*, 6
getClone *(*Tumor-class*)*, 8

plotTumorData *(*Simulating Clones*)*, 6

runAlg *(*Seeking Clones*)*, 2

sampleSimplex *(*Simplices*)*, 5
seekClones *(*Seeking Clones*)*, 2
Seeking Clones, 2
Simplices, 5, *10*
Simulating Clones, 6
summary,Tumor-method *(*Tumor-class*)*, 8

Tumor *(*Tumor-class*)*, 8
Tumor-class, 8
tumorGen *(*Simulating Clones*)*, 6

WeightVector, *9*
WeightVector *(*WeightVector-class*)*, 9
WeightVector-class, 9