# Package 'simphony'

May 9, 2026

**Type** Package

**Title** Simulating Large-Scale, Rhythmic Data

**Version** 1.0.3

**Description** A tool for simulating rhythmic data: transcriptome data using Gaussian or negative binomial distributions, and behavioral activity data using Bernoulli or Poisson distributions. See Singer et al. (2019) <doi:10.7717/peerj.6985>.

**URL** https://simphony.hugheylab.org, https://github.com/hugheylab/simphony

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.1

**Depends** R (>= 3.4)

**Imports** data.table (>= 1.11.4), foreach (>= 1.4.4)

**Suggests** ggplot2 (>= 3.0.0), kableExtra (>= 0.9.0), knitr (>= 1.20), limma (>= 3.34.9), precrec (>= 0.9.1), rmarkdown (>= 1.9), testthat (>= 2.0.0)

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Jake Hughey [aut, cre], Jordan Singer [aut], Darwin Fu [ctb]

**Maintainer** Jake Hughey <jakejhughey@gmail.com>

**Repository** CRAN

**Date/Publication** 2022-08-09 15:30:05 UTC

# Contents



---

`defaultDispFunc` *Default function for mapping expected counts to dispersion.*

---

## Description

The function was estimated from circadian RNA-seq data from mouse liver (PRJNA297287), using local regression in `DESeq2`. In a negative binomial distribution, $variance = mean + mean^2 * dispersion$.

## Usage

```
defaultDispFunc(x)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector of mean counts. |

## Format

An object of class `function` of length 1.

## Value

Numeric vector of dispersions.

## See Also

`simphony()`

## Examples

```
means = 2^(6:10)
dispersions = defaultDispFunc(means)
```

getExpectedAbund *Calculate expected abundance*

## Description

Calculate expected abundance for multiple features at multiple timepoints in multiple conditions.

## Usage

```
getExpectedAbund(
  featureMetadata,
  times = NULL,
  sampleMetadata = NULL,
  byCondGroup = is.null(times)
)
```

## Arguments

| | |
|---|---|
| featureMetadata | data.table with columns feature, base, rhyFunc, amp, period, and phase, where every row corresponds to a gene. If byCondGroup is TRUE, then must also have columns cond and group. |
| times | Numeric vector of the times at which to calculate expected abundance for each row in featureMetadata. |
| sampleMetadata | data.table with columns sample, cond, and time. Either times or sampleMetadata must be provided, and the former takes precedence. |
| byCondGroup | Logical for whether to speed up the calculation by grouping by the columns cond and group. Primarily for internal use. |

## Value

data.table derived from featureMetadata (but with more rows), with additional columns time and mu and possibly others. If sampling will use the negative binomial family, mu corresponds to log2 counts.

## See Also

simphony(), getSampledAbund()

## Examples

```
library('data.table')
featureMetadata = data.table(feature = c('feature_1', 'feature_2'),
                             base = function(x) 0,
                             amp = c(function(x) 0, function(x) 1),
                             period = 24,
                             phase = 0, rhyFunc = sin)
```

```
abundDt = getExpectedAbund(featureMetadata, times = 6:17)
```

---

`getSampledAbund` *Sample abundance values*

---

## Description

Sample feature abundance values from the given distributions. This function is used internally by `simphony()`, and should not usually need to be called directly.

## Usage

```
getSampledAbund(
  abundDt,
  logOdds = FALSE,
  family = c("gaussian", "negbinom", "bernoulli", "poisson"),
  inplace = FALSE
)
```

## Arguments

| | |
|---|---|
| `abundDt` | `data.table` of expected abundance. If `family` is 'gaussian', required columns are `feature`, `sample`, `mu`, and `sd`. If `family` is 'negbinom', required columns are `feature`, `sample`, `mu`, `dispFunc`, `cond`, and `group`. If `family` is 'bernoulli' or 'poisson', required columns are `feature`, `sample`, and `mu`. |
| `logOdds` | Logical for whether `mu` corresponds to log-odds. Only used if `family` is 'bernoulli'. |
| `family` | Character string for the family of distributions from which to sample the abundance values. `simphony` will give a warning if it tries to sample from a distribution outside the region in which the distribution is defined: $\mu < 0$ for negative binomial and Poisson, and $\mu < 0$ or $\mu > 1$ for Bernoulli. |
| `inplace` | Logical for whether to modify `abundDt` in-place, adding a column `abund` containing the abundance values. |

## Value

Matrix of abundance values, where rows correspond to features and columns correspond to samples.

## See Also

`simphony()`, `getExpectedAbund()`

## Examples

```
library('data.table')
set.seed(6022)
abundDt = data.table(feature = 'feature_1', sample = c('sample_1', 'sample_2'),
                     mu = c(0, 5), sd = 1)
abundMat = getSampledAbund(abundDt)
```

---

`mergeSimData` *Merge abundance data, feature metadata, and sample metadata*

---

## Description

Merge a simulation's abundance data, feature metadata, and sample metadata into one `data.table`. This function is useful for making plots using ggplot2.

## Usage

```
mergeSimData(simData, features = NULL)
```

## Arguments

| | |
|---|---|
| `simData` | List with the following elements, such as returned by `simphony()`:<br>**abundData** Matrix of abundance values, with rownames for features and colnames for samples.<br>**sampleMetadata** `data.table` with columns `sample` and `cond`.<br>**featureMetadata** `data.table` with columns `feature` and `cond`. |
| `features` | Character vector of features for which to get abundance data. If `NULL`, then all features. |

## Value

`data.table`.

## See Also

`simphony()`

## Examples

```
library('data.table')
featureGroups = data.table(amp = c(0, 1))
simData = simphony(featureGroups)
mergedSimData = mergeSimData(simData, simData$featureMetadata$feature[1:2])
```

simphony *Simulate feature abundance data*

## Description

Simulate experiments in which abundances of rhythmic and non-rhythmic features are measured at multiple timepoints in one or more conditions.

## Usage

```
simphony(
  featureGroupsList,
  fracFeatures = NULL,
  nFeatures = 10,
  timepointsType = c("auto", "specified", "random"),
  timeRange = c(0, 48),
  interval = 2,
  nReps = 1,
  timepoints = NULL,
  nSamplesPerCond = NULL,
  rhyFunc = sin,
  dispFunc = NULL,
  logOdds = FALSE,
  family = c("gaussian", "negbinom", "bernoulli", "poisson")
)
```

## Arguments

featureGroupsList

data.frame or data.table (for a single condition) or list of data.frames or data.tables (for multiple conditions), where each row corresponds to a group of features to simulate. The following columns are all optional:

**fracFeatures** Fraction of simulated features to allocate to each group. Defaults to 1/(number of groups).

**rhyFunc** Function to generate rhythmic abundance. Must have a period of $2\pi$. Defaults to sin.

**amp** Amplitude of rhythm. Defaults to 0. Corresponds to multiplicative term in front of rhyFunc. Can be numeric (constant over time) or a function (time-dependent). See vignette for examples.

**period** Period of rhythm. Defaults to 24.

**phase** Phase of rhythm, in the same units as period. Defaults to 0. Corresponds to an additive term in rhyFunc.

**base** Baseline abundance, i.e., abundance when rhyFunc term is 0. Depending on family, defaults to 0 ('gaussian'), 8 ('negbinom', mean log2 counts), 0 ('bernoulli' with logOdds as TRUE), 0.5 ('bernoulli' if logOdds as FALSE), or 1 ('poisson'). Can be numeric (constant over time) or a function (time-dependent). See vignette for examples.

**sd** Standard deviation of sampled abundance values. Defaults to 1. Only used if `family` is 'gaussian'.

**dispFunc** Function to calculate dispersion of sampled abundance values, given expected abundance in counts. Only used if `family` is 'negbinom'.

`fracFeatures` Fraction of simulated features to allocate to each group. Defaults to 1/(number of groups). Only used if the first `featureGroupsList` `data.frame` lacks a `fracFeatures` column.

`nFeatures` Integer for the total number of features to simulate.

`timepointsType` Character string for how to set the timepoints for the simulation. Must be 'auto' (default), 'specified', or 'random'.

`timeRange` Numeric vector for the range of timepoints to use for the simulation. Defaults to c(0, 48). Only used if `timepointsType` is 'auto' or 'random'.

`interval` Number for the amount of time between consecutive timepoints, in the same units as `period`. The first timepoint is 0. Only used if `timepointsType` is 'auto'.

`nReps` Integer for the number of replicates per timepoint. Only used if `timepointsType` is 'auto'.

`timepoints` Numeric vector of exact timepoints to simulate, including any replicates. Only used if `timepointsType` is 'specified'.

`nSamplesPerCond` Integer for the number of samples per condition, which will be randomly uniformly spaced between 0 and `period` and different for each condition. Only used if timepointsType is 'random'.

`rhyFunc` Function to generate rhythmic abundance. Must have a period of $2\pi$. Defaults to `sin`. Only used if a `data.frame` in `featureGroupsList` lacks a `rhyFunc` column.

`dispFunc` Function to calculate dispersion of sampled abundance values, given expected abundance in counts. Defaults to `defaultDispFunc`. Only used if `family` is 'negbinom' and a `data.frame` in `featureGroupsList` lacks a `dispFunc` column.

`logOdds` Logical for whether the rhythmic function corresponds to log-odds. Only used if `family` is 'bernoulli'.

`family` Character string for the family of distributions from which to sample the abundance values. `simphony` will give a warning if it tries to sample from a distribution outside the region in which the distribution is defined: $\mu < 0$ for negative binomial and Poisson, and $\mu < 0$ or $\mu > 1$ for Bernoulli.

## Value

List with the following elements:

**abundData** Matrix of abundance values (counts, if `family` is 'negbinom'), with features as rownames and samples as colnames.

**sampleMetadata** `data.table` with one row per sample.

**featureMetadata** `data.table` with one row per feature per condition. Columns `amp` and `base` are functions of time. Columns `amp0` and `base0` are numeric and correspond to the amplitude and baseline abundance at time 0, respectively.

**experMetadata** List of arguments that were passed to `simphony`.

## See Also

`defaultDispFunc()`, `getExpectedAbund()`, `getSampledAbund()`, `mergeSimData()`

## Examples

```
library('data.table')

# Simulate data for features having one of three sets of rhythmic parameters.
featureGroups = data.table(amp = c(0, 1, 1), phase = c(0, 0, 6),
                           rhyFunc = c(cos, cos, sin))
simData = simphony(featureGroups)

# Simulate data for an experiment with specified timepoints and replicates.
featureGroups = data.table(amp = c(0, 1))
simData = simphony(featureGroups, timepointsType = 'specified',
                   timepoints = c(0, 2, 2, 4, 12, 16, 21))

# Simulate data for an experiment with random timepoints between 0 and 24.
featureGroups = data.table(amp = c(0, 2))
simData = simphony(featureGroups, timepointsType = 'random',
                   timeRange = c(0, 24), nSamplesPerCond = 20)

# Simulate data with time-dependent rhythm amplitude or baseline abundance
featureGroups = data.table(amp = c(function(x) 1, function(x) 2^(-x / 24)),
                           base = c(function(x) x / 12, function(x) 0))
simData = simphony(featureGroups)

# Simulate data for features whose rhythmicity varies between two conditions.
featureGroupsList = list(
  data.table(amp = c(1, 2, 2), phase = c(0, -3, 0), period = c(24, 24, 22)),
  data.table(amp = c(3, 2, 2), phase = c(0, 3, 0), period = c(24, 24, 26)))
simData = simphony(featureGroupsList)

# Simulate data from a negative binomial distribution with a higher variance.
featureGroups = data.table(amp = 1, base = 6:8)
dispFunc = function(x) 3 * defaultDispFunc(x)
simData = simphony(featureGroups, family = 'negbinom', dispFunc = dispFunc)

# Simulate data at high temporal resolution from a Poisson distribution that
# alternates between two states.
featureGroups = data.table(amp = 1, base = 0,
                           rhyFunc = function(x) ifelse(x %% (2 * pi) < pi, 0.5, 4))

simData = simphony(featureGroups, timeRange = c(0, 24 * 4), interval = 0.1,
                   nReps = 1, family = 'poisson')
```

```
# Simulate data for 100 features, half non-rhythmic and half rhythmic, with
# amplitudes for rhythmic features sampled from a log-normal distribution.
nFeatures = 100
rhyFrac = 0.5
nRhyFeatures = round(rhyFrac * nFeatures)
rhyAmps = exp(rnorm(nRhyFeatures, mean = 0, sd = 0.25))
fracFeatures = c(1 - rhyFrac, rep(rhyFrac / nRhyFeatures, nRhyFeatures))
featureGroups = data.table(amp = c(0, rhyAmps), fracFeatures = fracFeatures)
simData = simphony(featureGroups, nFeatures = nFeatures)

# Simulate data for 100 rhythmic features, with baseline log2 expected counts
# and residual log dispersion sampled from distributions whose parameters
# were estimated, using DESeq2 and fitdistrplus, from circadian RNA-seq data
# from mouse liver (PRJNA297287).
nFeatures = 100
baseLog2Counts = rnorm(nFeatures, mean = 8.63, sd = 2.73)
dispFactors = exp(rnorm(nFeatures, sd = 0.819))
dispFuncs = sapply(dispFactors, function(z) {function(x) defaultDispFunc(x) * z})
featureGroups = data.table(base = baseLog2Counts, dispFunc = dispFuncs, amp = 1)
simData = simphony(featureGroups, nFeatures = nFeatures, family = 'negbinom')
```

---

`splitDiffFeatureGroups` *Split differential featureGroups*

---

## Description

Split a diffFeatureGroups data.frame into a list of two featureGroups data.frames, which can then be passed to `simphony()`.

## Usage

```
splitDiffFeatureGroups(diffFeatureGroups, checkValid = TRUE)
```

## Arguments

| | |
|---|---|
| `diffFeatureGroups` | `data.frame` with optional columns `meanBase`, `dBase`, `meanSd`, `dSd`, `meanAmp`, `dAmp`, `meanPhase`, and `dPhase` describing the changes in abundance between two conditions. Each row corresponds to a group of features. |
| `checkValid` | Logical for whether to only return rows for which both amplitudes are greater than or equal to zero and both standard deviations are greater than zero. |

## Value

List of two `data.tables` with possible columns `base`, `sd`, `amp`, and `phase`, depending on the columns in `diffFeatureGroups`.

## See Also

simphony()

## Examples

```
dGroups = data.frame(meanAmp = c(1, 1, 1, 1), dAmp = c(1, 1, 2, 2),
                     meanPhase = c(0, 0, 0, 0), dPhase = c(0, 3, 0, 3))
featureGroups = splitDiffFeatureGroups(dGroups)
```

# Index