

# Package ‘disclapmix’

July 22, 2025

**Type** Package

**Title** Discrete Laplace Mixture Inference using the EM Algorithm

**Version** 1.7.4

**Author** Mikkel Meyer Andersen [aut, cre],  
Poul Svante Eriksen [aut]

**Maintainer** Mikkel Meyer Andersen <mik1@math.aau.dk>

**Description** Make inference in a mixture of discrete Laplace distributions using the EM algorithm. This can e.g. be used for modelling the distribution of Y chromosomal haplotypes as described in [1, 2] (refer to the URL section).

**License** GPL-2 | file LICENSE

**LinkingTo** Rcpp, RcppProgress

**Imports** Rcpp (>= 0.11), disclap (>= 1.4), cluster (>= 1.14.4), MASS,  
stats, graphics, methods, utils

**Suggests** knitr, ggplot2, gridExtra, ggdendro, scales, seriation,  
fwsim, testthat, rmarkdown

**LazyLoad** yes

**BugReports** <https://github.com/mikldk/disclapmix/issues>

**VignetteBuilder** knitr

**SystemRequirements** C++11

**Encoding** UTF-8

**URL** <http://dx.doi.org/10.1016/j.jtbi.2013.03.009>  
<https://arxiv.org/abs/1304.2129>

**RoxygenNote** 7.2.0

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2022-06-29 14:10:02 UTC

## Contents

disclapmix-package . . . . .	2
clusterdist . . . . .	3
clusterprob . . . . .	3
contributor_pairs . . . . .	4
danes . . . . .	4
disclapmix . . . . .	5
disclapmix_adaptive . . . . .	8
disclapmix_robust . . . . .	9
generate_mixture . . . . .	10
get_rank . . . . .	10
haplotype_diversity . . . . .	11
plot.disclapmixfit . . . . .	11
plot.ranked_contrib_pairs . . . . .	12
predict.disclapmixfit . . . . .	13
print.contrib_pairs . . . . .	13
print.disclapmixfit . . . . .	14
print.ranked_contrib_pairs . . . . .	14
rank_contributor_pairs . . . . .	15
simulate.disclapmixfit . . . . .	16
summary.disclapmixfit . . . . .	17

<b>Index</b>	<b>18</b>
--------------	-----------

---

disclapmix-package     *disclapmix*

---

## Description

Discrete Laplace Mixture Inference using the EM Algorithm. A central function is [disclapmix\\_adaptive](#) (and the underlying [disclapmixfit](#)).

## Author(s)

Mikkel Meyer Andersen <mikl@math.aau.dk> and Poul Svante Eriksen <svante@math.aau.dk>

---

clusterdist	<i>Calculate distance between clusters</i>
-------------	--

---

**Description**

clusterdist calculates the distance between each pair of clusters. The distance measure is based on a symmetric Kullback-Leibler divergence.

**Usage**

```
clusterdist(fit, ...)
```

**Arguments**

fit	A <code>disclapmixfit</code> object.
...	Not used

**Value**

A distance matrix

**See Also**

[disclapmix-package](#) [disclapmix](#) [disclapmixfit](#) [clusterprob](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#) [disclap](#)

---

clusterprob	<i>Cluster origin probabilities for haplotypes</i>
-------------	--

---

**Description**

clusterprob calculates the cluster origin probabilities for haplotypes.

**Usage**

```
clusterprob(fit, newdata, ...)
```

**Arguments**

fit	A <code>disclapmixfit</code> object.
newdata	The haplotypes to predict the cluster origin probabilities for.
...	Not used

**Value**

A matrix where the rows correspond to the rows in newdata and the sum of each row is 1.

**See Also**

[disclapmix-package](#) [disclapmix](#) [disclapmixfit](#) [clusterdist](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#) [disclap](#)

---

`contributor_pairs`      *Contributor pairs from a 2 person mixture*

---

**Description**

Get all possible contributor pairs from a 2 person mixture

**Usage**

```
contributor_pairs(mixture)
```

**Arguments**

`mixture`      A list of integer vectors. The k'th element in the list is an integer vector with the alleles in the mixture at locus k.

**Value**

A `contrib_pairs` object that is a unordered list of pairs. Note, that contributor order is disregarded so that each contributor pair is only present once (and not twice as would be the case if taking order into consideration). See example usage at [rank\\_contributor\\_pairs](#).

**See Also**

[rank\\_contributor\\_pairs](#) [generate\\_mixture](#) [disclapmix-package](#) [disclapmix](#) [disclapmixfit](#) [clusterprob](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#) [disclap](#)

---

`danesh`      *Y-STR haplotypes*

---

**Description**

185 Y-STR 10 loci haplotypes

**Format**

A data frame with 185 observations on the following 10 loci (n is the number of times each haplotype has been observed)

**DYS19**  
**DYS389I**  
**DYS389II**  
**DYS390**  
**DYS391**  
**DYS392**  
**DYS393**  
**DYS437**  
**DYS438**  
**DYS439**  
**n**

**Source**

”Y-chromosome STR haplotypes Danes” by Hallenberg et al (2005), <http://www.sciencedirect.com/science/article/pii/S03790>

---

disclapmix

*Discrete Laplace mixture inference using the EM algorithm*

---

**Description**

disclapmix makes inference in a mixture of Discrete Laplace distributions using the EM algorithm. After the EM algorithm has converged, the centers are moved if the marginal likelihood increases by doing so. And then the EM algorithm is run again. This continues until the centers are not moved.

**Usage**

```
disclapmix(  
  x,  
  clusters,  
  init_y = NULL,  
  iterations = 100L,  
  eps = 0.001,  
  verbose = 0L,  
  glm_method = "internal_coef",  
  glm_control_maxit = 50L,  
  glm_control_eps = 1e-06,  
  init_y_method = "pam",  
  init_v = NULL,  
  ret_x = FALSE,  
  ...  
)
```

**Arguments**

<code>x</code>	Dataset.
<code>clusters</code>	The number of clusters/components to fit the model for.
<code>init_y</code>	Initial central haplotypes, if NULL, these will be estimated as described under the <code>init_y_method</code> argument.
<code>iterations</code>	Maximum number of iterations in the EM-algorithm.
<code>eps</code>	Convergence stop criteria in the EM algorithm which is compared to $\frac{\max\{v_{new} - v_{old}\}}{\max\{v_{old}\}}$ , where $v$ is a matrix of each observation's probability of belonging to a certain center.
<code>verbose</code>	from 0 to 2 (both including): 0 for silent, 2 for extra verbose.
<code>glm_method</code>	<code>internal_coef</code> , <code>internal_dev</code> or <code>glm.fit</code> . Please see details.
<code>glm_control_maxit</code>	Integer giving the maximal number of IWLS iterations.
<code>glm_control_eps</code>	Positive convergence tolerance epsilon; the iterations converge when $ x - x_{old}  / ( x  + 0.1) < \text{epsilon}$ , where $x = \text{beta\_correction}$ for <code>internal_coef</code> and $x = \text{deviance}$ otherwise.
<code>init_y_method</code>	Which cluster method to use for finding initial central haplotypes, $y$ : <code>pam</code> (recommended) or <code>clara</code> . Ignored if <code>init_y</code> is supplied.
<code>init_v</code>	Matrix with <code>nrow(x)</code> rows and <code>clusters</code> columns specifying initial posterior probabilities to get EM started, if none specified, then <code>matrix(1/clusters, nrow = nrow(x), ncol = clusters)</code> is used.
<code>ret_x</code>	Return data <code>'x'</code>
<code>...</code>	Used to detect obsolete usage (when using parameters <code>centers</code> , <code>use.parallel</code> , <code>calculate.logLs</code> or <code>plots.prefix</code> ).

**Details**

`glm_method`: `internal_coef` is the fastest as it uses the relative changes in the coefficients as a stopping criterium, hence it does not need to compute the deviance until the very end. In normal situations, it would not be a problem to use this method. `internal_dev` is the reasonably fast method that uses the deviance as a stopping criterium (like `glm.fit`). `glm.fit` to use the traditional `glm.fit` IWLS implementation and is slow compared to the other two methods.

`init_y_method`: For `init_y_method = 'clara'`, the sampling parameters are: `samples = 100`, `sampsize = min(ceiling(nrow(x)/2), 100 + 2*clusters)` and the random number generator in R is used.

**Value**

A `disclapmixfit` object:

**list("glm\_method")** The supplied GLM method.

**list("init\_y")** The supplied initial central haplotypes, `init_y`.

**list("init\_y\_method")** The supplied method for choosing initial central haplotypes (only used if `init_y` is NULL).

- list("converged")** Whether the estimation converged or not.
- list("x")** Dataset used to fit the model if 'ret\_x' is 'TRUE', else 'NULL'.
- list("y")** The central haplotypes, y.
- list("tau")** The prior probabilities of belonging to a cluster, tau.
- list("v\_matrix")** The matrix v of each observation's probability of belonging to a certain cluster. The rows are in the same order as the observations in x used to generate this fit.
- list("disclap\_parameters")** A matrix with the estimated discrete Laplace parameters.
- list("glm\_coef")** The coefficients from the last GLM fit (used to calculate disclap\_parameters).
- list("model\_observations")** Number of observations.
- list("model\_parameters")** Number of parameters in the model.
- list("iterations")** Number of iterations performed in total (including moving centers and re-estimating using the EM algorithm).
- list("logL\_full")** Full log likelihood of the final model.
- list("logL\_marginal")** Marginal log likelihood of the final model.
- list("BIC\_full")** BIC based on the full log likelihood of the final model.
- list("BIC\_marginal")** BIC based on the marginal log likelihood of the final model.
- list("v\_gain\_iterations")** The gain  $\frac{\max\{v_{new} - v_{old}\}}{\max\{v_{old}\}}$ , where v is vic\_matrix mentioned above, during the iterations.
- list("tau\_iterations")** The prior probability of belonging to the centers during the iterations.
- list("logL\_full\_iterations")** Full log likelihood of the models during the iterations (only calculated when verbose = 2L).
- list("logL\_marginal\_iterations")** Marginal log likelihood of the models during the iterations (only calculated when verbose = 2L).
- list("BIC\_full\_iterations")** BIC based on full log likelihood of the models during the iterations (only calculated when verbose = 2L).
- list("BIC\_marginal\_iterations")** BIC based on marginal log likelihood of the models during the iterations (only calculated when verbose = 2L).

### See Also

[disclapmix-package](#) [disclapmix](#) [disclapmixfit](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#) [clusterdist](#) [clusterprobgm](#).fit [disclap](#) [pam](#) [clara](#)

### Examples

```
# Generate sample database
db <- matrix(disclap::rdisclap(1000, 0.3), nrow = 250, ncol = 4)

# Add location parameters
db <- sapply(1:ncol(db), function(i) as.integer(db[, i]+13+i))

head(db)
```

```

fit1 <- disclapmix(db, clusters = 1L, verbose = 1L, glm_method = "glm.fit")
fit1$disclap_parameters
fit1$y

fit1b <- disclapmix(db, clusters = 1L, verbose = 1L, glm_method = "internal_coef")
fit1b$disclap_parameters
fit1b$y

max(abs(fit1$disclap_parameters - fit1b$disclap_parameters))

# Generate another type of database
db2 <- matrix(disclap::rdiscap(2000, 0.1), nrow = 500, ncol = 4)
db2 <- sapply(1:ncol(db2), function(i) as.integer(db2[, i]+14+i))
fit2 <- disclapmix(rbind(db, db2), clusters = 2L, verbose = 1L)
fit2$disclap_parameters
fit2$y
fit2$tau

```

---

disclapmix\_adaptive    *Adaptive fitting*

---

## Description

A wrapper around ‘disclapmix\_robust()’ that instead of fitting one model for a given number of clusters, fits models until the best model (lowest marginal BIC) is in the interior (with margin ‘M’) of all number of clusters tried.

## Usage

```
disclapmix_adaptive(x, margin = 5L, criteria = "BIC_marginal", ...)
```

## Arguments

x	Dataset.
margin	Fit models until there is at least this margin
criteria	The slot to chose the best model from (small values indicate better model)
...	Passed on to ‘disclapmix_robust()’ (and further to ‘disclapmix()’)

## Details

E.g., the best model has 3 clusters and the margin ‘M = 5’, then this function ensures that models with 1, 2, ..., 3+5 = 8 clusters are fitted. If e.g. then 7 is better than 3, then it continues such that also models with up to 7+5 = 12 clusters are fitted.

Note that models with 1-5 clusters are always fitted.

## Value

A list of all ‘disclapmix’ fits

**Examples**

```

data(danes)
db <- as.matrix(danes[rep(1:nrow(danes), danes$n), 1:(ncol(danes)-1)])
fits <- disclapmix_adaptive(db, margin = 5L)
fits
BICs <- sapply(fits, function(x) x$BIC_marginal)
BICs
ks <- sapply(fits, function(x) nrow(x$y)) # Always same as seq_along(fits)
ks
max_k <- max(ks)
best_k <- which.min(BICs)
max_k
best_k
max_k - best_k # = margin = 5

```

---

disclapmix\_robust      *Robust fitting*

---

**Description**

A wrapper around ‘disclapmix()’ that tries to avoid errors. Can sometimes avoid errors with SVD problems happening with ‘glm\_method = ‘internal\_coef’ and ‘glm\_method = ‘internal\_dev’.

**Usage**

```
disclapmix_robust(x, clusters, ...)
```

**Arguments**

x	Dataset.
clusters	The number of clusters/components to fit the model for.
...	Passed on to ‘disclapmix()’

**Examples**

```

data(danes)
db <- as.matrix(danes[rep(1:nrow(danes), danes$n), 1:(ncol(danes)-1)])
fit <- disclapmix_robust(db, 3L)
fit

```

---

generate_mixture	<i>Generate a mixture</i>
------------------	---------------------------

---

**Description**

This function can generate a mixture given a list of contributors.

**Usage**

```
generate_mixture(profiles)
```

**Arguments**

profiles      A list with profiles to mix.

**Value**

A list, e.g. for use with [contributor\\_pairs](#). See example usage at [rank\\_contributor\\_pairs](#).

**See Also**

[contributor\\_pairs](#) [rank\\_contributor\\_pairs](#) [disclapmix-package](#) [disclapmix](#) [disclapmixfit](#)  
[clusterprob](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#)  
[disclap](#)

---

get_rank	<i>Get rank of pair</i>
----------	-------------------------

---

**Description**

Get rank of pair

**Usage**

```
get_rank(x, haplotype)
```

**Arguments**

x              A ranked\_contrib\_pairs object.  
haplotype      A haplotype.

---

haplotype\_diversity    *Calculate haplotype diversity from a disclapmixfit*

---

### Description

Calculate haplotype diversity from a [disclapmixfit](#) object. The method is based on simulating a huge database that approximates the population.

### Usage

```
haplotype_diversity(object, nsim = 10000L)
```

### Arguments

object            a [disclapmixfit](#) object, usually from a result of a call to [disclapmix](#).  
 nsim             number of haplotypes to generate for calculating the haplotype diversity.

### Value

The calculated haplotype diversity.

### See Also

[disclapmix](#) [disclapmixfit](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#)  
[simulate.disclapmixfit](#)

---

[plot.disclapmixfit](#)    *Plot a disclapmixfit*

---

### Description

Plot a [disclapmixfit](#) object.

### Usage

```
## S3 method for class 'disclapmixfit'  
plot(x, which = 1L, clusdist = clusterdist(x), ...)
```

### Arguments

x                 a [disclapmixfit](#) object, usually from a result of a call to [disclapmix](#).  
 which            What plot to make. 1L = clusters and their distances.  
 clusdist         To use previously computed cluster distances to avoid doing the same computations twice.  
 ...              not used

**Value**

A data frame with discrete Laplace distributions for each cluster and locus. Side effect: A plot.

**See Also**

[disclapmix](#) [disclapmixfit](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [simulate.disclapmixfit](#)  
[summary.disclapmixfit](#)

**Examples**

```
data(danes)
db <- as.matrix(danes[rep(1:nrow(danes), danes$n), 1:(ncol(danes)-1)])
fit <- disclapmix(db, clusters = 4L)
plot(fit)
```

---

```
plot.ranked_contrib_pairs
```

*Plot ranked contributor pairs*

---

**Description**

Plot ranked contributor pairs

**Usage**

```
## S3 method for class 'ranked_contrib_pairs'
plot(x, top = NULL, ..., xlab = "Rank", ylab = "P(H1)P(H2)")
```

**Arguments**

x	A ranked_contrib_pairs object.
top	The top ranked number of pairs to print. NULL for all.
...	Delegated to the generic <a href="#">plot</a> function.
xlab	Graphical parameter.
ylab	Graphical parameter.

---

predict.disclapmixfit *Predict from a disclapmixfit*

---

### Description

Is able to predict haplotype frequencies using a [disclapmixfit](#) object.

### Usage

```
## S3 method for class 'disclapmixfit'  
predict(object, newdata, ...)
```

### Arguments

object	a <a href="#">disclapmixfit</a> object
newdata	the haplotypes in matrix format to estimate haplotype probabilities for
...	not used

### See Also

[disclapmix](#) [disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#)  
[plot.disclapmixfit](#) [clusterprob](#)

---

print.contrib\_pairs *Print contributor pairs*

---

### Description

Print contributor pairs

### Usage

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

### Arguments

x	A <a href="#">contrib_pairs</a> object.
...	Ignored

---

```
print.disclapmixfit Print a disclapmixfit
```

---

### Description

Prints a [disclapmixfit](#) object.

### Usage

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

### Arguments

x                    a [disclapmixfit](#) object, usually from a result of a call to [disclapmix](#).  
 ...                  not used

### See Also

[disclapmix](#) [disclapmixfit](#) [predict.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#)  
[plot.disclapmixfit](#)

---

```
print.ranked_contrib_pairs
Print ranked contributor pairs
```

---

### Description

Print ranked contributor pairs

### Usage

```
## S3 method for class 'ranked_contrib_pairs'
print(x, top = 5L, hide_non_varying_loci = TRUE, ...)
```

### Arguments

x                    A [ranked\\_contrib\\_pairs](#) object.  
 top                  The top ranked number of pairs to print/plot. NULL for all.  
 hide\_non\_varying\_loci  
                      Whether to hide alleles on loci that do not vary.  
 ...                  Ignored

---

rank\_contributor\_pairs

*Separate a 2 person mixture*


---

### Description

Separate a 2 person mixture by ranking the possible contributor pairs.

### Usage

```
rank_contributor_pairs(contrib_pairs, fit, max_rank = NULL)
```

### Arguments

`contrib_pairs` A `contrib_pairs` object obtained from [contributor\\_pairs](#).  
`fit` A `disclapmixfit` object.  
`max_rank` Not used. Reserved for future use.

### Value

A `ranked_contrib_pairs` object that is basically an order vector and the probabilities for each pair (in the same order as given in `contrib_pairs`), found by using `fit`. Note, that contributor order is disregarded so that each contributor pair is only present once (and not twice as would be the case if taking order into consideration).

### See Also

[contributor\\_pairs](#) [generate\\_mixture](#) [disclapmix-package](#) [disclapmix](#) [disclapmixfit](#) [clusterprob](#)  
[predict.disclapmixfit](#) [print.disclapmixfit](#) [summary.disclapmixfit](#) [simulate.disclapmixfit](#)  
[disclap](#)

### Examples

```
data(danes)
db <- as.matrix(danes[rep(1L:nrow(danes), danes$n), 1L:(ncol(danes) - 1L)])

set.seed(1)
true_contribs <- sample(1L:nrow(db), 2L)
h1 <- db[true_contribs[1L], ]
h2 <- db[true_contribs[2L], ]
db_ref <- db[-true_contribs, ]

h1h2 <- c(paste(h1, collapse = ";"), paste(h2, collapse = ";"))
tab_db <- table(apply(db, 1, paste, collapse = ";"))
tab_db_ref <- table(apply(db_ref, 1, paste, collapse = ";"))
tab_db[h1h2]
tab_db_ref[h1h2]
```

```

rm(db) # To avoid use by accident

mixture <- generate_mixture(list(h1, h2))

possible_contributors <- contributor_pairs(mixture)
possible_contributors

fits <- lapply(1L:5L, function(clus) disclapmix(db_ref, clusters = clus))

best_fit_BIC <- fits[[which.min(sapply(fits, function(fit) fit$BIC_marginal))]]
best_fit_BIC

ranked_contributors_BIC <- rank_contributor_pairs(possible_contributors, best_fit_BIC)
ranked_contributors_BIC

plot(ranked_contributors_BIC, top = 10L, type = "b")

get_rank(ranked_contributors_BIC, h1)

```

---

```
simulate.disclapmixfit
```

*Simulate from a disclapmixfit*

---

## Description

Simulate from a [disclapmixfit](#) object.

## Usage

```
## S3 method for class 'disclapmixfit'
simulate(object, nsim = 1L, seed = NULL, ...)
```

## Arguments

object	a <a href="#">disclapmixfit</a> object, usually from a result of a call to <code>disclapmix</code> .
nsim	number of haplotypes to generate.
seed	not used
...	not used

## Value

A matrix where the rows correspond to the simulated haplotypes.

## See Also

[disclapmix](#) [disclapmixfit](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [plot.disclapmixfit](#)  
[summary.disclapmixfit](#)

---

summary.disclapmixfit *Summary of a disclapmixfit*

---

**Description**

Summary of a [disclapmixfit](#) object.

**Usage**

```
## S3 method for class 'disclapmixfit'  
summary(object, ...)
```

**Arguments**

object	a <a href="#">disclapmixfit</a> object, usually from a result of a call to <code>disclapmix</code> .
...	not used

**See Also**

[disclapmix](#) [disclapmixfit](#) [predict.disclapmixfit](#) [print.disclapmixfit](#) [simulate.disclapmixfit](#)  
[clusterdist](#)

# Index

- \* **clusters**
    - clusterdist, 3
    - clusterprob, 3
    - disclapmix, 5
  - \* **datasets**
    - danes, 4
  - \* **deconvolution**
    - contributor\_pairs, 4
    - generate\_mixture, 10
    - rank\_contributor\_pairs, 15
  - \* **disclapmix**
    - disclapmix, 5
  - \* **distance**
    - clusterdist, 3
    - clusterprob, 3
  - \* **eps**
    - disclapmix, 5
  - \* **mixture**
    - contributor\_pairs, 4
    - generate\_mixture, 10
    - rank\_contributor\_pairs, 15
  - \* **plot**
    - plot.disclapmixfit, 11
  - \* **predict**
    - predict.disclapmixfit, 13
  - \* **print**
    - haplotype\_diversity, 11
    - print.disclapmixfit, 14
    - simulate.disclapmixfit, 16
    - summary.disclapmixfit, 17
  - \* **separation**
    - contributor\_pairs, 4
    - generate\_mixture, 10
    - rank\_contributor\_pairs, 15
- clara, 7
- clusterdist, 3, 4, 7, 17
- clusterprob, 3, 3, 4, 7, 10, 13, 15
- contributor\_pairs, 4, 10, 15
- danes, 4
- disclap, 3, 4, 7, 10, 15
- disclapmix, 3, 4, 5, 7, 10–17
- disclapmix-package, 2
- disclapmix\_adaptive, 2, 8
- disclapmix\_robust, 9
- disclapmixfit, 2–4, 6, 7, 10–17
- disclapmixfit (disclapmix), 5
- generate\_mixture, 4, 10, 15
- get\_rank, 10
- glm.fit, 7
- haplotype\_diversity, 11
- pam, 7
- plot, 12
- plot.disclapmixfit, 11, 13, 14, 16
- plot.ranked\_contrib\_pairs, 12
- predict.disclapmixfit, 3, 4, 7, 10–12, 13, 14–17
- print.contrib\_pairs, 13
- print.disclapmixfit, 3, 4, 7, 10–13, 14, 15–17
- print.ranked\_contrib\_pairs, 14
- rank\_contributor\_pairs, 4, 10, 15
- simulate.disclapmixfit, 3, 4, 7, 10–15, 16, 17
- summary.disclapmixfit, 3, 4, 7, 10–16, 17