

# Package ‘fpc’

November 16, 2009

**Title** Fixed point clusters, clusterwise regression and discriminant plots

**Version** 1.2-7

**Date** 2009-11-16

**Author** Christian Hennig <chrish@stats.ucl.ac.uk>

**Depends** R (>= 1.9), MASS, cluster

**Suggests** mclust, trimcluster, prabclus, class

**Description** Fuzzy and crisp fixed point cluster analysis based on Mahalanobis distance and linear regression fixed point clusters. Semi-explorative, semi-model-based clustering methods, operating on  $n \times p$  data, do not need prespecification of number of clusters, produce overlapping clusters. Symmetric and asymmetric discriminant projections separate groups optimally, used to visualize the separation of groupings, visual cluster validation. Clusterwise linear regression by normal mixture modeling. Cluster validation statistics for distance based clustering including corrected Rand index. Clusterwise cluster stability assessment. DBSCAN clustering. Interface functions for many clustering methods implemented in R. Note that the use of the package mclust (called by function prabclus) is protected by a special license, see <http://www.stat.washington.edu/mclust/license.txt>, particularly point 6.

**Maintainer** Christian Hennig <chrish@stats.ucl.ac.uk>

**License** GPL

**URL** <http://www.homepages.ucl.ac.uk/~ucakche/>

**Repository** CRAN

**Date/Publication** 2009-11-16 19:25:06

## R topics documented:

adcoord	2
ancoord	4
awcoord	5
batcoord	7

can . . . . .	9
clujaccard . . . . .	10
clusexpect . . . . .	11
cluster.stats . . . . .	12
clusterboot . . . . .	14
cmahal . . . . .	20
con.comp . . . . .	21
cov.wml . . . . .	22
cweight . . . . .	24
dbscan . . . . .	24
discrcoord . . . . .	27
discrproj . . . . .	28
fixmahal . . . . .	30
fixreg . . . . .	37
fpclusters . . . . .	42
itnumber . . . . .	43
kmeansCBI . . . . .	44
kmeansruns . . . . .	48
mahalanodisc . . . . .	50
mahalanofix . . . . .	51
mahalconf . . . . .	52
minsize . . . . .	53
mvdcoord . . . . .	54
ncoord . . . . .	56
pamk . . . . .	57
plotcluster . . . . .	59
randcmatrix . . . . .	61
randconf . . . . .	62
regmix . . . . .	62
rFace . . . . .	65
simmatrix . . . . .	66
solvecov . . . . .	67
sseg . . . . .	68
tdecomp . . . . .	69
tonedata . . . . .	70
wfu . . . . .	70

**Index****72**

## Description

Asymmetric discriminant coordinates as defined in Hennig (2003). Asymmetric discriminant projection means that there are two classes, one of which is treated as the homogeneous class (i.e., it should appear homogeneous and separated in the resulting projection) while the other may be heterogeneous. The principle is to maximize the ratio between the projection of a between classes separation matrix and the projection of the covariance matrix within the homogeneous class.

## Usage

```
adcoord(xd, clvecd, clnum=1)
```

## Arguments

<code>xd</code>	the data matrix; a numerical object which can be coerced to a matrix.
<code>clvecd</code>	integer vector of class numbers; length must equal <code>nrow(xd)</code> .
<code>clnum</code>	integer. Number of the homogeneous class.

## Details

The square root of the homogeneous classes covariance matrix is inverted by use of `tdecomp`, which can be expected to give reasonable results for singular within-class covariance matrices.

## Value

List with the following components

<code>ev</code>	eigenvalues in descending order.
<code>units</code>	columns are coordinates of projection basis vectors. New points <code>x</code> can be projected onto the projection basis vectors by <code>x %*% units</code>
<code>proj</code>	projections of <code>xd</code> onto <code>units</code> .

## Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

## References

Hennig, C. (2004) Asymmetric linear dimension reduction for classification. *Journal of Computational and Graphical Statistics* 13, 930-945 .

Hennig, C. (2005) A method for visual cluster validation. In: Weihs, C. and Gaul, W. (eds.): *Classification - The Ubiquitous Challenge*. Springer, Heidelberg 2005, 153-160.

## See Also

[plotcluster](#) for straight forward discriminant plots. [discrproj](#) for alternatives. [rFace](#) for generation of the example data used below.

## Examples

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
adcf <- adcoord(face, grface==2)
adcf2 <- adcoord(face, grface==4)
plot(adcf$proj, col=1+(grface==2))
plot(adcf2$proj, col=1+(grface==4))
# ...done in one step by function plotcluster.
```

---

ancoord

*Asymmetric neighborhood based discriminant coordinates*

---

## Description

Asymmetric neighborhood based discriminant coordinates as defined in Hennig (2003). Asymmetric discriminant projection means that there are two classes, one of which is treated as the homogeneous class (i.e., it should appear homogeneous and separated in the resulting projection) while the other may be heterogeneous. The principle is to maximize the ratio between the projection of a between classes covariance matrix, which is defined by averaging the between classes covariance matrices in the neighborhoods of the points of the homogeneous class and the projection of the covariance matrix within the homogeneous class.

## Usage

```
ancoord(xd, clvecd, clnum=1, nn=50, method="mcd", countmode=1000, ...)
```

## Arguments

<code>xd</code>	the data matrix; a numerical object which can be coerced to a matrix.
<code>clvecd</code>	integer vector of class numbers; length must equal <code>nrow(xd)</code> .
<code>clnum</code>	integer. Number of the homogeneous class.
<code>nn</code>	integer. Number of points which belong to the neighborhood of each point (including the point itself).
<code>method</code>	one of "mve", "mcd" or "classical". Covariance matrix used within the homogeneous class. "mcd" and "mve" are robust covariance matrices as implemented in <code>cov.rob</code> . "classical" refers to the classical covariance matrix.
<code>countmode</code>	optional positive integer. Every <code>countmode</code> algorithm runs <code>ancoord</code> shows a message.
<code>...</code>	no effect

## Details

The square root of the homogeneous classes covariance matrix is inverted by use of `tdecomp`, which can be expected to give reasonable results for singular within-class covariance matrices.

**Value**

List with the following components

ev	eigenvalues in descending order.
units	columns are coordinates of projection basis vectors. New points $x$ can be projected onto the projection basis vectors by $x \%*\% \text{units}$
proj	projections of $xd$ onto $\text{units}$ .

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Hennig, C. (2004) Asymmetric linear dimension reduction for classification. *Journal of Computational and Graphical Statistics* 13, 930-945 .

Hennig, C. (2005) A method for visual cluster validation. In: Weihs, C. and Gaul, W. (eds.): *Classification - The Ubiquitous Challenge*. Springer, Heidelberg 2005, 153-160.

**See Also**

[plotcluster](#) for straight forward discriminant plots. [discrproj](#) for alternatives. [rFace](#) for generation of the example data used below.

**Examples**

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
ancf2 <- ancoord(face, grface==4)
plot(ancf2$proj, col=1+(grface==4))
# ...done in one step by function plotcluster.
```

---

awcoord

*Asymmetric weighted discriminant coordinates*


---

**Description**

Asymmetric weighted discriminant coordinates as defined in Hennig (2003). Asymmetric discriminant projection means that there are two classes, one of which is treated as the homogeneous class (i.e., it should appear homogeneous and separated in the resulting projection) while the other may be heterogeneous. The principle is to maximize the ratio between the projection of a between classes separation matrix and the projection of the covariance matrix within the homogeneous class. Points are weighted according to their (robust) Mahalanobis distance to the homogeneous class.

**Usage**

```
awcoord(xd, clvecd, clnum=1, mahal="square", method="classical",
        clweight=switch(method, classical=FALSE, TRUE),
        alpha=0.99, subsample=0, countmode=1000, ...)
```

**Arguments**

<code>xd</code>	the data matrix; a numerical object which can be coerced to a matrix.
<code>clvecd</code>	integer vector of class numbers; length must equal <code>nrow(xd)</code> .
<code>clnum</code>	integer. Number of the homogeneous class.
<code>mahal</code>	"md" or "square". If "md", the points are weighted by the square root of the alpha-quantile of the corresponding chi squared distribution over the roots of their Mahalanobis distance to the homogeneous class, unless this is smaller than 1. If "square" (which is recommended), the (originally squared) Mahalanobis distance and the unrooted quantile is used.
<code>method</code>	one of "mve", "mcd" or "classical". Covariance matrix used within the homogeneous class and for the computation of the Mahalanobis distances. "mcd" and "mve" are robust covariance matrices as implemented in <code>cov.rob</code> . "classical" refers to the classical covariance matrix.
<code>clweight</code>	logical. If <code>FALSE</code> , only the points of the heterogeneous class are weighted. This, together with <code>method="classical"</code> , computes AWC as defined in Hennig (2003). If <code>TRUE</code> , all points are weighted. This, together with <code>method="mcd"</code> , computes ARC as defined in Hennig (2003).
<code>alpha</code>	numeric between 0 and 1. The corresponding quantile of the chi squared distribution is used for the downweighting of points. Points with a smaller Mahalanobis distance to the homogeneous class get full weight.
<code>subsample</code>	integer. If 0, all points are used. Else, only a subsample of <code>subsample</code> of the points is used.
<code>countmode</code>	optional positive integer. Every <code>countmode</code> algorithm runs <code>awcoord</code> shows a message.
<code>...</code>	no effect

**Details**

The square root of the homogeneous classes covariance matrix is inverted by use of `tdecomp`, which can be expected to give reasonable results for singular within-class covariance matrices.

**Value**

List with the following components

<code>ev</code>	eigenvalues in descending order.
<code>units</code>	columns are coordinates of projection basis vectors. New points <code>x</code> can be projected onto the projection basis vectors by <code>x %*% units</code>
<code>proj</code>	projections of <code>xd</code> onto <code>units</code> .

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Hennig, C. (2004) Asymmetric linear dimension reduction for classification. *Journal of Computational and Graphical Statistics* 13, 930-945 .

Hennig, C. (2005) A method for visual cluster validation. In: Weihs, C. and Gaul, W. (eds.): *Classification - The Ubiquitous Challenge*. Springer, Heidelberg 2005, 153-160.

**See Also**

[plotcluster](#) for straight forward discriminant plots. [discrproj](#) for alternatives. [rFace](#) for generation of the example data used below.

**Examples**

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
awcf <- awcoord(face, grface==1)
# awcf2 <- ancoord(face, grface==1, method="mcd")
plot(awcf$proj, col=1+(grface==1))
# plot(awcf2$proj, col=1+(grface==1))
# ...done in one step by function plotcluster.
```

---

batcoord

*Bhattacharyya discriminant projection*


---

**Description**

Computes Bhattacharyya discriminant projection coordinates as described in Fukunaga (1990), p. 455 ff.

**Usage**

```
batcoord(xd, clvecd, clnum=1, dom="mean")
batvarcoord(xd, clvecd, clnum=1)
```

**Arguments**

xd	the data matrix; a numerical object which can be coerced to a matrix.
clvecd	integer or logical vector of class numbers; length must equal <code>nrow(xd)</code> .
clnum	integer, one of the values of <code>clvecd</code> , if this is an integer vector. Bhattacharyya projections can only be computed if there are only two classes in the dataset. <code>clnum</code> is the number of one of the two classes. All the points indicated by other values of <code>clvecd</code> are interpreted as the second class.

dom string. dom="mean" means that the discriminant coordinate for the group means is computed as the first projection direction by `discrcoord` (option `pool="equal"`; both classes have the same weight for computing the within-class covariance matrix). Then the data is projected into a subspace orthogonal (w.r.t. the within-class covariance) to the discriminant coordinate, and the projection coordinates to maximize the differences in variance are computed. dom="variance" means that the projection coordinates maximizing the difference in variances are computed. Then they are ordered with respect to the Bhattacharyya distance, which takes also the mean differences into account. Both procedures are implemented as described in Fukunaga (1990).

### Details

`batvarcoord` computes the optimal projection coordinates with respect to the difference in variances. `batcoord` combines the differences in mean and variance as explained for the argument `dom`.

### Value

`batcoord` returns a list with the components `ev`, `rev`, `units`, `proj`. `batvarcoord` returns a list with the components `ev`, `rev`, `units`, `proj`, `W`, `S1`, `S2`.

`ev` vector of eigenvalues. If `dom="mean"`, then first eigenvalue from `discrcoord`. Further eigenvalues are of  $S_1^{-1}S_2$ , where  $S_i$  is the covariance matrix of class  $i$ . For `batvarcoord` or if `dom="variance"`, all eigenvalues come from  $S_1^{-1}S_2$  and are ordered by `rev`.

`rev` for `batcoord`: vector of projected Bhattacharyya distances (Fukunaga (1990), p. 99). Determine quality of the projection coordinates. For `batvarcoord`: vector of amount of projected difference in variances.

`units` columns are coordinates of projection basis vectors. New points `x` can be projected onto the projection basis vectors by `x %*% units`.

`proj` projections of `xd` onto `units`.

`W` matrix  $S_1^{-1}S_2$ .

`S1` covariance matrix of the first class.

`S2` covariance matrix of the second class.

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition* (2nd ed.). Boston: Academic Press.

**See Also**

[plotcluster](#) for straight forward discriminant plots.

[discrcoord](#) for discriminant coordinates.

[rFace](#) for generation of the example data used below.

**Examples**

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
bcf2 <- batcoord(face, grface==2)
plot(bcf2$proj, col=1+(grface==2))
bcfv2 <- batcoord(face, grface==2, dom="variance")
plot(bcfv2$proj, col=1+(grface==2))
bcfvv2 <- batvarcoord(face, grface==2)
plot(bcfvv2$proj, col=1+(grface==2))
```

---

 can

*Generation of the tuning constant for regression fixed point clusters*


---

**Description**

Generates tuning constants `ca` for [fixreg](#) dependent on the number of points and variables of the dataset.

Only thought for use in [fixreg](#).

**Usage**

```
can(n, p)
```

**Arguments**

<code>n</code>	positive integer. Number of points.
<code>p</code>	positive integer. Number of independent variables.

**Details**

The formula is  $3 + 33/(n * 2^{-(p-1)/2})^{1/3} + 2900000/(n * 2^{-(p-1)/2})^3$ . For justification cf. Hennig (2002).

**Value**

A number.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

## References

Hennig, C. (2002) Fixed point clusters for linear regression: computation and comparison, *Journal of Classification* 19, 249-276.

## See Also

[fixreg](#)

## Examples

```
can(429, 3)
```

---

clujaccard

*Jaccard similarity between logical vectors*

---

## Description

Jaccard similarity between logical or 0-1 vectors:  $\text{sum}(c1 \ \& \ c2) / \text{sum}(c1 \ | \ c2)$ .

## Usage

```
clujaccard(c1, c2, zerobyzero=NA)
```

## Arguments

`c1`                logical or 0-1-vector.  
`c2`                logical or 0-1-vector (same length).  
`zerobyzero`       result if  $\text{sum}(c1 \ | \ c2) = 0$ .

## Value

Numeric between 0 and 1.

## Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

## Examples

```
c1 <- rep(TRUE, 10)
c2 <- c(FALSE, rep(TRUE, 9))
clujaccard(c1, c2)
```

---

`clusexpect`*Expected value of the number of times a fixed point cluster is found*

---

**Description**

A rough approximation of the expectation of the number of times a well separated fixed point cluster (FPC) of size `n` is found in `ir` fixed point iterations of `fixreg`.

**Usage**

```
clusexpect(n, p, cn, ir)
```

**Arguments**

<code>n</code>	positive integer. Total number of points.
<code>p</code>	positive integer. Number of independent variables.
<code>cn</code>	positive integer smaller or equal to <code>n</code> . Size of the FPC.
<code>ir</code>	positive integer. Number of fixed point iterations.

**Details**

The approximation is based on the assumption that a well separated FPC is found iff all  $p+2$  points of the initial configuration come from the FPC. The value is `ir` times the probability for this. For a discussion of this assumption cf. Hennig (2002).

**Value**

A number.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Hennig, C. (2002) Fixed point clusters for linear regression: computation and comparison, *Journal of Classification* 19, 249-276.

**See Also**

[fixreg](#)

**Examples**

```
clusexpect(500, 4, 150, 2000)
```

---

cluster.stats      *Cluster validation statistics*

---

### Description

Computes a number of distance based statistics which can be used for cluster validation, comparison between clusterings and decision about the number of clusters: cluster sizes, cluster diameters, average distances within and between clusters, cluster separation, average silhouette widths, the best distance based statistics to decide about the number of clusters in a study of Milligan and Cooper (1985), Hubert's gamma coefficient, the Dunn index and two indexes to assess the similarity of two clusterings, namely the corrected Rand index and Meila's VI.

### Usage

```
cluster.stats(d,clustering,alt.clustering=NULL,
              silhouette=TRUE,G2=FALSE,G3=FALSE,
              compareonly=FALSE)
```

### Arguments

d	a distance object (as generated by <code>dist</code> ) or a distance matrix between cases.
clustering	an integer vector of length of the number of cases, which indicates a clustering. The clusters have to be numbered from 1 to the number of clusters.
alt.clustering	an integer vector such as for <code>clustering</code> , indicating an alternative clustering. If provided, the corrected Rand index and Meila's VI for <code>clustering</code> vs. <code>alt.clustering</code> are computed.
silhouette	logical. If TRUE, the silhouette statistics are computed, which requires package <code>cluster</code> .
G2	logical. If TRUE, Goodman and Kruskal's index G2 (cf. Gordon (1999), p. 62) is computed. This executes lots of sorting algorithms and can be very slow (it has been improved by R. Francois - thanks!)
G3	logical. If TRUE, the index G3 (cf. Gordon (1999), p. 62) is computed. This executes <code>sort</code> on all distances and can be extremely slow.
compareonly	logical. If TRUE, only the corrected Rand index and Meila's VI are computed and given out (this requires <code>alt.clustering</code> to be specified).

### Value

`cluster.stats` returns a list containing the components `n`, `cluster.number`, `cluster.size`, `diameter`, `average.distance`, `median.distance`, `separation`, `average.together`, `separation.matrix`, `average.between`, `average.within`, `n.between`, `n.within`, `within.cluster.ss`, `clus.avg.silwidths`, `avg.silwidth`, `g2`, `g3`, `hubertgamma`, `dunn`, `entropy`, `wb.ratio`, `corrected.rand`, `vi` except if `compareonly=TRUE`, in which case only the last two components are computed.

n	number of cases.
cluster.number	number of clusters.
cluster.size	vector of cluster sizes (number of points).
diameter	vector of cluster diameters (maximum within cluster distances).
average.distance	vector of clusterwise within cluster average distances.
median.distance	vector of clusterwise within cluster distance medians.
separation	vector of clusterwise minimum distances of a point in the cluster to a point of another cluster.
average.toother	vector of clusterwise average distances of a point in the cluster to the points of other clusters.
separation.matrix	matrix of separation values between all pairs of clusters.
average.between	average distance between clusters.
average.within	average distance within clusters.
n.between	number of distances between clusters.
n.within	number of distances within clusters.
within.cluster.ss	a generalisation of the within clusters sum of squares (k-means objective function), which is obtained if $d$ is a Euclidean distance matrix. For general distance measures, this is half the sum of the within cluster squared dissimilarities divided by the cluster size.
clus.avg.silwidths	vector of cluster average silhouette widths. See <a href="#">silhouette</a> .
avg.silwidth	average silhouette width. See <a href="#">silhouette</a> .
g2	Goodman and Kruskal's Gamma coefficient. See Milligan and Cooper (1985), Gordon (1999, p. 62).
g3	G3 coefficient. See Gordon (1999, p. 62).
hubertgamma	correlation between distances and a 0-1-vector where 0 means same cluster, 1 means different clusters. See Haldiki et al. (2002).
dunn	minimum separation / maximum diameter. Dunn index, see Haldiki et al. (2002).
entropy	entropy of the distribution of cluster memberships, see Meila(2007).
wb.ratio	average.within/average.between.
corrected.rand	corrected Rand index (if <code>alt.clustering</code> has been specified), see Gordon (1999, p. 198).
vi	variation of information (VI) index (if <code>alt.clustering</code> has been specified), see Meila (2007).

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

- Gordon, A. D. (1999) *Classification*, 2nd ed. Chapman and Hall.
- Haldiki, M., Batistakis, Y., Vazirgiannis, M. (2002) Cluster validity methods, *SIGMOD*, Record 31, 40-45.
- Meila, M. (2007) Comparing clusterings? an information based distance, *Journal of Multivariate Analysis*, 98, 873-895.
- Milligan, G. W. and Cooper, M. C. (1985) An examination of procedures for determining the number of clusters. *Psychometrika*, 50, 159-179.

**See Also**

[silhouette](#), [dist](#) `clusterboot` computes clusterwise stability statistics by resampling.

**Examples**

```
set.seed(20000)
face <- rFace(200, dMoNo=2, dNoEy=0, p=2)
dface <- dist(face)
complete3 <- cutree(hclust(dface), 3)
cluster.stats(dface, complete3,
              alt.clustering=as.integer(attr(face, "grouping")))
```

---

clusterboot

*Clusterwise cluster stability assessment by resampling*

---

**Description**

Assessment of the clusterwise stability of a clustering of data, which can be cases\*variables or dissimilarity data. The data is resampled using several schemes (bootstrap, subsetting, jittering, replacement of points by noise) and the Jaccard similarities of the original clusters to the most similar clusters in the resampled data are computed. The mean over these similarities is used as an index of the stability of a cluster (other statistics can be computed as well). The methods are described in Hennig (2007).

`clusterboot` is an integrated function that computes the clustering as well, using interface functions for various clustering methods implemented in R (several interface functions are provided, but you can implement further ones for your favourite clustering method). See the documentation of the input parameter `clustermethod` below.

Quite general clustering methods are possible, i.e. methods estimating or fixing the number of clusters, methods producing overlapping clusters or not assigning all cases to clusters (but declaring them as "noise"). Fuzzy clusterings cannot be processed and have to be transformed to crisp clusterings by the interface function.

### Usage

```
clusterboot(data, B=100, distances=(class(data)=="dist"),
            bootmethod=if(distances) "boot"
            else c("boot", "noise"),
            bscompare=FALSE,
            multipleboot=TRUE,
            jittertuning=0.05, noisetuning=c(0.05, 4),
            subtuning=floor(nrow(data)/2),
            clustermethod, noisemethod=FALSE, count=TRUE,
            showplots=FALSE, dissolution=0.5,
            recover=0.75, seed=NULL, ...)

## S3 method for class 'clboot':
print(x, statistics=c("mean", "dissolution", "recovery"), ...)

## S3 method for class 'clboot':
plot(x, xlim=c(0, 1), breaks=seq(0, 1, by=0.05), ...)
```

### Arguments

data	something that can be coerced into a matrix. The data matrix - either an $n \times p$ -data matrix (or data frame) or an $n \times n$ -dissimilarity matrix (or <code>dist</code> -object).
B	integer. Number of resampling runs for each scheme, see <code>bootmethod</code> .
distances	logical. If TRUE, the data is interpreted as dissimilarity matrix. If data is a <code>dist</code> -object, <code>distances=TRUE</code> automatically, otherwise <code>distances=FALSE</code> by default. This means that you have to set it to TRUE manually if data is a dissimilarity matrix.
bootmethod	vector of strings, defining the methods used for resampling. Possible methods: "boot": nonparametric bootstrap (precise behaviour is controlled by parameters <code>bscompare</code> and <code>multipleboot</code> ). "subset": selecting random subsets from the dataset. Size determined by <code>subtuning</code> . "noise": replacing a certain percentage of the points by random noise, see <code>noisetuning</code> . "jitter" add random noise to all points, see <code>jittertuning</code> . (This didn't perform well in Hennig (2007), but you may want to get your own experience.) "bojit" nonparametric bootstrap first, and then adding noise to the points, see <code>jittertuning</code> . <b>Important:</b> only the methods "boot" and "subset" work with dissimilarity data!

The results in Hennig (2007) indicate that "boot" is generally informative and often quite similar to "subset" and "bojit", while "noise" sometimes provides different information. Therefore the default (for `distances=FALSE`) is to use "boot" and "noise". However, some clustering methods may have problems with multiple points, which can be solved by using "bojit" or "subset" instead of "boot" or by `multipleboot=FALSE` below.

<code>bscompare</code>	logical. If <code>TRUE</code> , multiple points in the bootstrap sample are taken into account to compute the Jaccard similarity to the original clusters (which are represented by their "bootstrap versions", i.e., the points of the original cluster which also occur in the bootstrap sample). If a point was drawn more than once, it is in the "bootstrap version" of the original cluster more than once, too, if <code>bscompare=TRUE</code> . Otherwise (default) multiple points are ignored for the computation of the Jaccard similarities. If <code>multipleboot=FALSE</code> , it doesn't make a difference.
<code>multipleboot</code>	logical. If <code>FALSE</code> , all points drawn more than once in the bootstrap draw are only used once in the bootstrap samples.
<code>jittertuning</code>	positive numeric. Tuning for the "jitter"-method. The noise distribution for jittering is a normal distribution with zero mean. The covariance matrix has the same Eigenvectors as that of the original data set, but the standard deviation along the principal directions is determined by the <code>jittertuning</code> -quantile of the distances between neighboring points projected along these directions.
<code>noisetuning</code>	A vector of two positive numerics. Tuning for the "noise"-method. The first component determines the probability that a point is replaced by noise. Noise is generated by a uniform distribution on a hyperrectangle along the principal directions of the original data set, ranging from <code>-noisetuning[2]</code> to <code>noisetuning[2]</code> times the standard deviation of the data set along the respective direction. Note that only points not replaced by noise are considered for the computation of Jaccard similarities.
<code>subtuning</code>	integer. Size of subsets for "subset".
<code>clustermethod</code>	an interface function (the function name, not a string containing the name, has to be provided!). This defines the clustering method. See the "Details"-section for a list of available interface functions and guidelines how to write your own ones.
<code>noisemethod</code>	logical. If <code>TRUE</code> , the last cluster is regarded as "noise component", which means that for computing the Jaccard similarity, it is not treated as a cluster. The noise component of the original clustering is only compared with the noise component of the clustering of the resampled data. This means that in the <code>clusterboot</code> -output (and plot), if points were assigned to the noise component, the last cluster number refers to it, and its Jaccard similarity values refer to comparisons with estimated noise components in resampled datasets only. (Some cluster methods such as trimmed k-means and <code>mclustBIC</code> produce such noise components.)
<code>count</code>	logical. If <code>TRUE</code> , the resampling runs are counted on the screen.
<code>showplots</code>	logical. If <code>TRUE</code> , a plot of the first two dimensions of the resampled data set (or the classical MDS solution for dissimilarity data) is shown for every resampling run. The last plot shows the original data set.

dissolution	numeric between 0 and 1. If the Jaccard similarity between the resampling version of the original cluster and the most similar cluster on the resampled data is smaller or equal to this value, the cluster is considered as "dissolved". Numbers of dissolved clusters are recorded.
recover	numeric between 0 and 1. If the Jaccard similarity between the resampling version of the original cluster and the most similar cluster on the resampled data is larger than this value, the cluster is considered as "successfully recovered". Numbers of recovered clusters are recorded.
seed	integer. Seed for random generator (fed into <code>set.seed</code> ) to make results reproducible. If <code>NULL</code> , results depend on chance.
...	additional parameters for the cluster methods called by <code>clusterboot</code> . No effect in <code>print.clboot</code> and <code>plot.clboot</code> .
x	object of class <code>clboot</code> .
statistics	specifies in <code>print.clboot</code> , which of the three clusterwise Jaccard similarity statistics "mean", "dissolution" (number of times the cluster has been dissolved) and "recovery" (number of times a cluster has been successfully recovered) is printed.
xlim	transferred to <code>hist</code> .
breaks	transferred to <code>hist</code> .

## Details

Here are some guidelines for interpretation. There is some theoretical justification to consider a Jaccard similarity value smaller or equal to 0.5 as an indication of a "dissolved cluster", see Hennig (2008). Generally, a valid, stable cluster should yield a mean Jaccard similarity value of 0.75 or more. Between 0.6 and 0.75, clusters may be considered as indicating patterns in the data, but which points exactly should belong to these clusters is highly doubtful. Below average Jaccard values of 0.6, clusters should not be trusted. "Highly stable" clusters should yield average Jaccard similarities of 0.85 and above. All of this refers to bootstrap; for the other resampling schemes it depends on the tuning constants, though their default values should grant similar interpretations in most cases.

While  $B=100$  is recommended, smaller run numbers could give quite informative results as well, if computation times become too high.

Note that the stability of a cluster is assessed, but stability is not the only important validity criterion - clusters obtained by very inflexible clustering methods may be stable but not valid, as discussed in Hennig (2007). See [plotcluster](#) for graphical cluster validation.

Information about interface functions for clustering methods:

The following interface functions are currently implemented (in the present package; note that almost all of these functions require the specification of some control parameters, so if you use one of them, look up their common help page [kmeansCBI](#)) first:

- `kmeansCBI` interface to the function `kmeans` for k-means clustering. This assumes a `cases*variables` matrix as input.
- `hclustCBI` interface to the function `hclust` for agglomerative hierarchical clustering with optional noise component. This function produces a partition and assumes a `cases*variables` matrix as input.

- `hclusttreeCBian` interface to the function `hclust` for agglomerative hierarchical clustering. This function produces a tree (not only a partition; therefore the number of clusters can be huge!) and assumes a `cases*variables` matrix as input.
- `disthclustCBian` interface to the function `hclust` for agglomerative hierarchical clustering with optional noise component. This function produces a partition and assumes a dissimilarity matrix as input.
- `noisemclustCBian` interface to the function `mclustBIC` for normal mixture model based clustering. This assumes a `cases*variables` matrix as input. Warning: `mclustBIC` sometimes has problems with multiple points. It is recommended to use this only together with `multipleboot=FALSE`.
- `distnoisemclustCBian` interface to the function `mclustBIC` for normal mixture model based clustering. This assumes a dissimilarity matrix as input and generates a data matrix by multidimensional scaling first. Warning: `mclustBIC` sometimes has problems with multiple points. It is recommended to use this only together with `multipleboot=FALSE`.
- `claraCBian` interface to the functions `pam` and `clara` for partitioning around medoids. This can be used with `cases*variables` as well as dissimilarity matrices as input.
- `pamkCBian` interface to the function `pamk` for partitioning around medoids. The number of cluster is estimated by the average silhouette width. This can be used with `cases*variables` as well as dissimilarity matrices as input.
- `trimkmeansCBian` interface to the function `trimkmeans` for trimmed k-means clustering. This assumes a `cases*variables` matrix as input.
- `distrimkmeansCBian` interface to the function `trimkmeans` for trimmed k-means clustering. This assumes a dissimilarity matrix as input and generates a data matrix by multidimensional scaling first.
- `dbscanCBian` interface to the function `dbscan` for density based clustering. This can be used with `cases*variables` as well as dissimilarity matrices as input.
- `mahalCBian` interface to the function `fixmahal` for fixed point clustering. This assumes a `cases*variables` matrix as input.

You can write your own interface function. The first argument of an interface function should always be a data matrix (of class "matrix", but it may be a symmetrical dissimilarity matrix). Further arguments can be tuning constants for the clustering method. The output of an interface function should be a list containing (at least) the following components:

- `resultclustering` result, usually a list with the full output of the clustering method (the precise format doesn't matter); whatever you want to use later.
- `n` number of clusters. If some points don't belong to any cluster but are declared as "noise", `nc` includes the noise component, and there should be another component `nccl`, being the number of clusters not including the noise component (note that it is not mandatory to define a noise component if not all points are assigned to clusters, but if you do it, the stability of the noise component is assessed as well.)
- `clusterlist` this is a list consisting of a logical vectors of length of the number of data points (`n`) for each cluster, indicating whether a point is a member of this cluster (`TRUE`) or not. If a noise component is included, it should always be the last vector in this list.
- `partition` an integer vector of length `n`, partitioning the data. If the method produces a partition, it should be the clustering. This component is only used for plots, so you could do something like `rep(1, n)` for non-partitioning methods.

- clustermethoda string indicating the clustering method.

### Value

clusterboot returns an object of class "clboot", which is a list with components result, partition, nc, clustermethod, B, bootmethod, multipleboot, dissolution, recover, bootresult, bootmean, bootbrd, bootrecover, jitterresult, jittermean, jitterbrd, jitterrecover, subsetresult, subsetmean, subsetbrd, subsetrecover, bojitresult, bojitmean, bojitbrd, bojitrecover, noiseresult, noisemean, noisebrd, noiserecover.

result	clustering result; full output of the selected clustermethod for the original data set.
partition	partition parameter of the selected clustermethod (note that this is only meaningful for partitioning clustering methods).
nc	number of clusters in original data (including noise component if noisemethod=TRUE). clustermethod, B, bootmethod, multipleboot, dissolution, recover input parameters, see above.
bootresult	matrix of Jaccard similarities for bootmethod="boot". Rows correspond to clusters in the original data set. Columns correspond to bootstrap runs.
bootmean	clusterwise means of the bootresult.
bootbrd	clusterwise number of times a cluster has been dissolved.
bootrecover	clusterwise number of times a cluster has been successfully recovered.
subsetresult, subsetmean, etc.	same as bootresult, bootmean, etc., but for the other resampling methods.

### Author(s)

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

- Hennig, C. (2007) Cluster-wise assessment of cluster stability. *Computational Statistics and Data Analysis*, 52, 258-271.
- Hennig, C. (2008) Dissolution point and isolation robustness: robustness criteria for general cluster analysis methods. *Journal of Multivariate Analysis* 99, 1154-1176.

### See Also

dist, interface functions: kmeansCBI, hclustCBI, hclusttreeCBI, disthclustCBI, noisemclustCBI, distnoisemclustCBI, claraCBI, pamkCBI, trimkmeansCBI, disttrimkmeansCBI, dbscanCBI, mahalCBI

**Examples**

```

set.seed(20000)
face <- rFace(50, dMoNo=2, dNoEy=0, p=2)
cf1 <- clusterboot(face, B=3, bootmethod=
  c("boot", "noise", "jitter"), clustermethod=kmeansCBI,
  k=5, seed=15555)

print(cf1)
plot(cf1)

cf2 <- clusterboot(dist(face), B=3, bootmethod=
  "subset", clustermethod=dsthclustCBI,
  k=5, cut="number", method="average", showplots=TRUE, seed=15555)
print(cf2)

```

---

cmahal

*Generation of tuning constant for Mahalanobis fixed point clusters.*


---

**Description**

Generates tuning constants  $c_a$  for `fixmahal` dependent on the number of points and variables of the current fixed point cluster (FPC).

This is experimental and only thought for use in `fixmahal`.

**Usage**

```
cmahal(n, p, nmin, cmin, ncl, cl = cmin, q = 1)
```

**Arguments**

<code>n</code>	positive integer. Number of points.
<code>p</code>	positive integer. Number of variables.
<code>nmin</code>	integer larger than 1. Smallest number of points for which $c_a$ is computed. For smaller FPC sizes, $c_a$ is set to the value for <code>nmin</code> .
<code>cmin</code>	positive number. Minimum value for $c_a$ .
<code>ncl</code>	positive integer. Number of points at which $c_a=c_1$ .
<code>c1</code>	positive numeric. Tuning constant for <code>cmahal</code> . Value for $c_a$ for FPC size equal to <code>ncl</code> .
<code>q</code>	numeric between 0 and 1. 1 for steepest possible descent of $c_a$ as function of the FPC size. Should presumably always be 1.

**Details**

Some experiments suggest that the tuning constant  $c_a$  should decrease with increasing FPC size and increase with increasing  $p$  in `fixmahal`. This is to prevent too small meaningless FPCs while maintaining the significant larger ones. `cmahal` with  $q=1$  computes  $c_a$  in such a way that as long as  $c_a > c_{min}$ , the decrease in  $n$  is as steep as possible in order to maintain the validity of the convergence theorem in Hennig and Christlieb (2002).

**Value**

A numeric vector of length  $n$ , giving the values for  $c_a$  for all FPC sizes smaller or equal to  $n$ .

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Hennig, C. and Christlieb, N. (2002) Validating visual clusters in large datasets: Fixed point clusters of spectral features, *Computational Statistics and Data Analysis* 40, 723-739.

**See Also**

[fixmahal](#)

**Examples**

```
plot(1:100, cmahal(100, 3, nmin=5, cmin=qchisq(0.99, 3), ncl=90),
     xlab="FPC size", ylab="cmahal")
```

---

con.comp

*Connectivity components of an undirected graph*

---

**Description**

Computes the connectivity components of an undirected graph from a matrix giving the edges.

**Usage**

```
con.comp(comat)
```

**Arguments**

`comat` a symmetric logical or 0-1 matrix, where `comat[i, j]=TRUE` means that there is an edge between vertices  $i$  and  $j$ . The diagonal is ignored.

**Details**

The "depth-first search" algorithm of Cormen, Leiserson and Rivest (1990, p. 477) is used.

**Value**

An integer vector, giving the number of the connectivity component for each vertice.

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990), *Introduction to Algorithms*, Cambridge: MIT Press.

**See Also**

[hclust](#), [cutree](#) for cutted single linkage trees (often equivalent).

**Examples**

```
set.seed(1000)
x <- rnorm(20)
m <- matrix(0,nrow=20,ncol=20)
for(i in 1:20)
  for(j in 1:20)
    m[i,j] <- abs(x[i]-x[j])
d <- m<0.2
cc <- con.comp(d)
max(cc) # number of connectivity components
plot(x,cc)
# The same should be produced by
# cutree(hclust(as.dist(m),method="single"),h=0.2).
```

---

 cov.wml

*Weighted Covariance Matrices (Maximum Likelihood)*


---

**Description**

Returns a list containing estimates of the weighted covariance matrix and the mean of the data, and optionally of the (weighted) correlation matrix. The covariance matrix is divided by the sum of the weights, corresponding to  $n$  and the ML-estimator in the case of equal weights, as opposed to  $n-1$  for [cov.wt](#).

**Usage**

```
cov.wml(x, wt = rep(1/nrow(x), nrow(x)), cor = FALSE, center = TRUE)
```

**Arguments**

<code>x</code>	a matrix or data frame. As usual, rows are observations and columns are variables.
<code>wt</code>	a non-negative and non-zero vector of weights for each observation. Its length must equal the number of rows of <code>x</code> .
<code>cor</code>	A logical indicating whether the estimated correlation weighted matrix will be returned as well.
<code>center</code>	Either a logical or a numeric vector specifying the centers to be used when computing covariances. If <code>TRUE</code> , the (weighted) mean of each variable is used, if <code>FALSE</code> , zero is used. If <code>center</code> is numeric, its length must equal the number of columns of <code>x</code> .

**Value**

A list containing the following named components:

<code>cov</code>	the estimated (weighted) covariance matrix.
<code>center</code>	an estimate for the center (mean) of the data.
<code>n.obs</code>	the number of observations (rows) in <code>x</code> .
<code>wt</code>	the weights used in the estimation. Only returned if given as an argument.
<code>cor</code>	the estimated correlation matrix. Only returned if <code>'cor'</code> is <code>'TRUE'</code> .

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[cov.wt](#), [cov](#), [var](#)

**Examples**

```
x <- c(1,2,3,4,5,6,7,8,9,10)
y <- c(1,2,3,8,7,6,5,8,9,10)
cov.wml(cbind(x,y),wt=c(0,0,0,1,1,1,1,1,0,0))
cov.wt(cbind(x,y),wt=c(0,0,0,1,1,1,1,1,0,0))
```

---

`cweight`*Weight function for AWC*

---

**Description**

For use in `awcoord` only.

**Usage**

```
cweight(x, ca)
```

**Arguments**

`x` numerical.

`ca` numerical.

**Value**

`ca/x` if smaller than 1, else 1.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[awcoord](#)

**Examples**

```
cweight(4, 1)
```

---

`dbscan`*Clustering: DBSCAN density reachability and connectivity*

---

**Description**

Generates a density based clustering of arbitrary shape as introduced in Ester et al. (1996).

**Usage**

```

dbscan(data, eps, MinPts = 5, scale = FALSE, method = c("hybrid", "raw",
  "dist"), seeds = TRUE, showplot = FALSE, countmode = NULL)
## S3 method for class 'dbscan':
print(x, ...)
## S3 method for class 'dbscan':
plot(x, data, ...)
## S3 method for class 'dbscan':
predict(object, data, newdata = NULL,
predict.max=1000, ...)

```

**Arguments**

data	data matrix, data.frame, dissimilarity matrix or dist-object
eps	Reachability Distance
MinPts	Reachability minimum no. of points
scale	scale the data
method	"dist" treats data as distance matrix (relatively fast but memory expensive), "raw" treats data as raw data and avoids calculating a distance matrix (saves memory but may be slow), "hybrid" expects also raw data, but calculates partial distance matrices (very fast with moderate memory requirements)
seeds	FALSE to not include the isseed-vector in the dbscan-object
showplot	0 = no plot, 1 = plot per iteration, 2 = plot per subiteration
countmode	NULL or vector of point numbers at which to report progress
x	object of class dbscan.
object	object of class dbscan.
newdata	matrix or data.frame with raw data to predict
predict.max	max. batch size for predictions
...	Further arguments transferred to plot methods.

**Details**

Clusters require a minimum no of points (MinPts) within a maximum distance (eps) around one of its members (the seed). Any point within eps around any point which satisfies the seed condition is a cluster member (recursively). Some points may not belong to any clusters (noise).

We have clustered a 100.000 x 2 dataset in 40 minutes on a Pentium M 1600 MHz.

`print.dbscan` shows a statistic of the number of points belonging to the clusters that are seeds and border points.

`plot.dbscan` distinguishes between seed and border points by plot symbol.

**Value**

`predict.dbscan` gives out a vector of predicted clusters for the points in `newdata`.

`dbscan` gives out an object of class 'dbscan' which is a LIST with components

<code>cluster</code>	integer vector coding cluster membership with noise observations (singletons) coded as 0
<code>isseed</code>	logical vector indicating whether a point is a seed (not border, not noise)
<code>eps</code>	parameter <code>eps</code>
<code>MinPts</code>	parameter <code>MinPts</code>

**Note**

this is a simplified version of the original algorithm (no K-D-trees used), thus we have  $o(n^2)$  instead of  $o(n * \log(n))$

**Author(s)**

Jens Oehlschlaegel, based on a draft by Christian Hennig.

**References**

Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Institute for Computer Science, University of Munich. Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96).

**Examples**

```
set.seed(665544)
n <- 600
x <- cbind(runif(10, 0, 10)+rnorm(n, sd=0.2), runif(10, 0, 10)+rnorm(n,
  sd=0.2))
par(bg="grey40")
ds <- dbscan(x, 0.2, showplot=TRUE)
ds
plot(ds, x)

x2 <- matrix(0,nrow=4,ncol=2)
x2[1,] <- c(5,2)
x2[2,] <- c(8,3)
x2[3,] <- c(4,4)
x2[4,] <- c(9,9)
predict(ds, x, x2)

n <- 600
x <- cbind((1:3)+rnorm(n, sd=0.2), (1:3)+rnorm(n, sd=0.2))

system.time(ds <- dbscan(x, 0.3, countmode=NULL, method="raw"))[3]
system.time(dsb <- dbscan(x, 0.3, countmode=NULL, method="hybrid"))[3]
```

```
system.time(dsc <- dbSCAN(dist(x), 0.3, countmode=NULL,
  method="dist")) [3]
```

discrcoord

*Discriminant coordinates/canonical variates***Description**

Computes discriminant coordinates, sometimes referred to as "canonical variates" as described in Seber (1984).

**Usage**

```
discrcoord(xd, clvecd, pool = "n", ...)
```

**Arguments**

<code>xd</code>	the data matrix; a numerical object which can be coerced to a matrix.
<code>clvecd</code>	integer vector of class numbers; length must equal <code>nrow(xd)</code> .
<code>pool</code>	string. Determines how the within classes covariance is pooled. "n" means that the class covariances are weighted corresponding to the number of points in each class (default). "equal" means that all classes get equal weight.
<code>...</code>	no effect

**Details**

The matrix  $T$  (see Seber (1984), p. 270) is inverted by use of `tdecomp`, which can be expected to give reasonable results for singular within-class covariance matrices.

**Value**

List with the following components

<code>ev</code>	eigenvalues in descending order.
<code>units</code>	columns are coordinates of projection basis vectors. New points <code>x</code> can be projected onto the projection basis vectors by <code>x %*% units</code>
<code>proj</code>	projections of <code>xd</code> onto <code>units</code> .

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Seber, G. A. F. (1984). *Multivariate Observations*. New York: Wiley.

**See Also**

[plotcluster](#) for straight forward discriminant plots.

[batcoord](#) for discriminating projections for two classes, so that also the differences in variance are shown ([discrcoord](#) is based only on differences in mean).

[rFace](#) for generation of the example data used below.

**Examples**

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
dcf <- discrcoord(face, grface)
plot(dcf$proj, col=grface)
# ...done in one step by function plotcluster.
```

---

discrproj

*Linear dimension reduction for classification*


---

**Description**

An interface for ten methods of linear dimension reduction in order to separate the groups optimally in the projected data. Includes classical discriminant coordinates, methods to project differences in mean and covariance structure, asymmetric methods (separation of a homogeneous class from a heterogeneous one), local neighborhood-based methods and methods based on robust covariance matrices.

**Usage**

```
discrproj(x, clvecd, method="dc", clnum=NULL, ignorepoints=FALSE,
          ignorenum=0, ...)
```

**Arguments**

x	the data matrix; a numerical object which can be coerced to a matrix.
clvecd	vector of class numbers which can be coerced into integers; length must equal <code>nrow(xd)</code> .
method	one of <b>"dc"</b> usual discriminant coordinates, see <a href="#">discrcoord</a> , <b>"bc"</b> Bhattacharyya coordinates, first coordinate showing mean differences, second showing covariance matrix differences, see <a href="#">batcoord</a> , <b>"vbc"</b> variance dominated Bhattacharyya coordinates, see <a href="#">batcoord</a> , <b>"mvdc"</b> added means and variance differences optimizing coordinates, see <a href="#">mvdcoord</a> , <b>"adc"</b> asymmetric discriminant coordinates, see <a href="#">adcoord</a> , <b>"awc"</b> asymmetric discriminant coordinates with weighted observations, see <a href="#">awcoord</a> ,

	"arc" asymmetric discriminant coordinates with weighted observations and robust MCD-covariance matrix, see <a href="#">awcoord</a> ,
	"nc" neighborhood based coordinates, see <a href="#">ncoord</a> ,
	"wnc" neighborhood based coordinates with weighted neighborhoods, see <a href="#">ncoord</a> ,
	"anc" asymmetric neighborhood based coordinates, see <a href="#">ancoord</a> .
	Note that "bc", "vbc", "adc", "awc", "arc" and "anc" assume that there are only two classes.
clnum	integer. Number of the class which is attempted to plot homogeneously by "asymmetric methods", which are the methods assuming that there are only two classes, as indicated above.
ignorepoints	logical. If TRUE, points with label ignorenum in clvecd are ignored in the computation for method and are only projected afterwards onto the resulting units. If pch=NULL, the plot symbol for these points is "N".
ignorenum	one of the potential values of the components of clvecd. Only has effect if ignorepoints=TRUE, see above.
...	additional parameters passed to the projection methods.

### Value

discrproj returns the output of the chosen projection method, which is a list with at least the components `ev`, `units`, `proj`. For detailed informations see the help pages of the projection methods.

ev	eigenvalues in descending order, usually indicating portion of information in the corresponding direction.
units	columns are coordinates of projection basis vectors. New points <code>x</code> can be projected onto the projection basis vectors by <code>x %*% units</code>
proj	projections of <code>xd</code> onto <code>units</code> .

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

- Hennig, C. (2004) Asymmetric linear dimension reduction for classification. *Journal of Computational and Graphical Statistics* 13, 930-945 .
- Hennig, C. (2005) A method for visual cluster validation. In: Weihs, C. and Gaul, W. (eds.): *Classification - The Ubiquitous Challenge*. Springer, Heidelberg 2005, 153-160.
- Seber, G. A. F. (1984). *Multivariate Observations*. New York: Wiley.
- Fukunaga (1990). *Introduction to Statistical Pattern Recognition* (2nd ed.). Boston: Academic Press.

**See Also**

[discrcoord](#), [batcoord](#), [mvdcoord](#), [adcoord](#), [awcoord](#), [ncoord](#), [ancoord](#).  
[rFace](#) for generation of the example data used below.

**Examples**

```
set.seed(4634)
face <- rFace(300, dMoNo=2, dNoEy=0, p=3)
grface <- as.integer(attr(face, "grouping"))
discrproj(face, grface, method="nc")$units
discrproj(face, grface, method="wnc")$units
discrproj(face, grface, clnum=1, method="arc")$units
```

---

fixmahal

---

*Mahalanobis Fixed Point Clusters*


---

**Description**

Computes Mahalanobis fixed point clusters (FPCs), i.e., subsets of the data, which consist exactly of the non-outliers w.r.t. themselves, and may be interpreted as generated from a homogeneous normal population. FPCs may overlap, are not necessarily exhausting and do not need a specification of the number of clusters.

Note that while `fixmahal` has lots of parameters, only one (or few) of them have usually to be specified, cf. the examples. The philosophy is to allow much flexibility, but to always provide sensible defaults.

**Usage**

```
fixmahal(dat, n = nrow(as.matrix(dat)), p = ncol(as.matrix(dat)),
         method = "fuzzy", cgen = "fixed",
         ca = NA, ca2 = NA,
         calpha = ifelse(method=="fuzzy", 0.95, 0.99),
         calpha2 = 0.995,
         pointit = TRUE, subset = n,
         ncl = 100+20*p,
         startn = 18+p, mnc = floor(startn/2),
         mer = ifelse(pointit, 0.1, 0),
         distcut = 0.85, maxit = 5*n, iter = n*1e-5,
         init.group = list(),
         ind.storage = TRUE, countmode = 100,
         plot = "none")
```

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

```
## S3 method for class 'summary.mfpc':
```

```

print(x, maxnc=30, ...)

## S3 method for class 'mfpc':
plot(x, dat, no, bw=FALSE, main=c("Representative FPC No. ",no),
      xlab=NULL, ylab=NULL,
      pch=NULL, col=NULL, ...)

## S3 method for class 'mfpc':
fpclusters(object, dat=NA, ca=object$ca, p=object$p, ...)

fpmi(dat, n = nrow(as.matrix(dat)), p = ncol(as.matrix(dat)),
      gv, ca, ca2, method = "ml", plot,
      maxit = 5*n, iter = n*1e-6)

```

## Arguments

dat	something that can be coerced to a numerical matrix or vector. Data matrix, rows are points, columns are variables. <code>fpclusters.rfpc</code> does not need specification of <code>dat</code> if <code>fixmahal</code> has been run with <code>ind.storage=TRUE</code> .
n	optional positive integer. Number of cases.
p	optional positive integer. Number of independent variables.
method	a string. <code>method="classical"</code> means 0-1 weighting of observations by Mahalanobis distances and use of the classical normal covariance estimator. <code>method="ml"</code> uses the ML-covariance estimator (division by $n$ instead of $n-1$ ) This is used in Hennig and Christlieb (2002). <code>method</code> can also be <code>"mcd"</code> or <code>"mve"</code> , to enforce the use of robust centers and covariance matrices, see <code>cov.rob</code> . This is experimental, not recommended at the moment, may be very slowly and requires library <code>lqs</code> . The default is <code>method="fuzzy"</code> , where weighted means and covariance matrices are used (Hennig, 2005). The weights are computed by <code>wfu</code> , i.e., a function that is constant 1 for arguments smaller than <code>ca</code> , 0 for arguments larger than <code>ca2</code> and continuously linear in between. Convergence is only proven for <code>method="ml"</code> up to now.
cgen	optional string. <code>"fixed"</code> means that the same tuning constant <code>ca</code> is used for all iterations. <code>"auto"</code> means that <code>ca</code> is generated dependently on the size of the current data subset in each iteration by <code>cmahal</code> . This is experimental.
ca	optional positive number. Tuning constant, specifying required cluster separation. By default determined as <code>calpha</code> -quantile of the chisquared distribution with $p$ degrees of freedom.
ca2	optional positive number. Second tuning constant needed if <code>method="fuzzy"</code> . By default determined as <code>calpha2</code> -quantile of the chisquared distribution with $p$ degrees of freedom.
calpha	number between 0 and 1. See <code>ca</code> .
calpha2	number between 0 and 1, larger than <code>calpha</code> . See <code>ca2</code> .
pointit	optional logical. If <code>TRUE</code> , subset fixed point algorithms are started from initial configurations, which are built around single points of the dataset, cf. <code>mahalconf</code> . Otherwise, initial configurations are only specified by <code>init.group</code> .

subset	optional positive integer smaller or equal than $n$ . Initial configurations for the fixed point algorithm (cf. <code>mahalconf</code> ) are built from a subset of <code>subset</code> points from the data. No effect if <code>pointit=FALSE</code> . Default: all points.
ncl	optional positive integer. Tuning constant needed by <code>cmahal</code> to generate <code>ca</code> automatically. Only needed for <code>cgen="auto"</code> .
startn	optional positive integer. Size of the initial configurations. The default value is chosen to prevent that small meaningless FPCs are found, but it should be decreased if clusters of size smaller than the default value are of interest.
mnc	optional positive integer. Minimum size of clusters to be reported.
mer	optional nonnegative number. FPCs (groups of them, respectively, see details) are only reported as stable if the ratio of the number of their findings to their number of points exceeds <code>mer</code> . This holds under <code>pointit=TRUE</code> and <code>subset=n</code> . For <code>subset&lt;n</code> , the ratio is adjusted, but for small <code>subset</code> , the results may extremely vary and have to be taken with care.
distcut	optional value between 0 and 1. A similarity measure between FPCs, given in Hennig (2002), and the corresponding Single Linkage groups of FPCs with similarity larger than <code>distcut</code> are computed. A single representative FPC is selected for each group.
maxit	optional integer. Maximum number of iterations per algorithm run (usually an FPC is found much earlier).
iter	positive number. Algorithm stops when difference between subsequent weight vectors is smaller than <code>iter</code> . Only needed for <code>method="fuzzy"</code> .
init.group	optional list of logical vectors of length $n$ . Every vector indicates a starting configuration for the fixed point algorithm. This can be used for datasets with high dimension, where the vectors of <code>init.group</code> indicate cluster candidates found by graphical inspection or background knowledge, as in Hennig and Christlieb (2002).
ind.storage	optional logical. If <code>TRUE</code> , then all indicator vectors of found FPCs are given in the value of <code>fixmahal</code> . May need lots of memory, but is a bit faster.
countmode	optional positive integer. Every <code>countmode</code> algorithm runs <code>fixmahal</code> shows a message.
plot	optional string. If <code>"start"</code> , you get a scatterplot of the first two variables to highlight the initial configuration, <code>"iteration"</code> generates such a plot at each iteration, <code>"both"</code> does both (this may be very time consuming). The default is <code>"none"</code> .
object	object of class <code>mfp</code> , output of <code>fixmahal</code> .
x	object of class <code>mfp</code> , output of <code>fixmahal</code> .
maxnc	positive integer. Maximum number of FPCs to be reported.
no	positive integer. Number of the representative FPC to be plotted.
bw	optional logical. If <code>TRUE</code> , plot is black/white, FPC is indicated by different symbol. Else FPC is indicated red.
main	plot title.
xlab	label for x-axis. If <code>NULL</code> , a default text is used.

<code>ylab</code>	label for y-axis. If <code>NULL</code> , a default text is used.
<code>pch</code>	plotting symbol, see <code>par</code> . If <code>NULL</code> , the default is used.
<code>col</code>	plotting color, see <code>par</code> . If <code>NULL</code> , the default is used.
<code>gv</code>	logical vector (or, with <code>method="fuzzy"</code> , vector of weights between 0 and 1) of length <code>n</code> . Indicates the initial configuration for the fixed point algorithm.
<code>...</code>	additional parameters to be passed to <code>plot</code> (no effects elsewhere).

## Details

A (crisp) Mahalanobis FPC is a data subset that reproduces itself under the following operation: Compute mean and covariance matrix estimator for the data subset, and compute all points of the dataset for which the squared Mahalanobis distance is smaller than `ca`.

Fixed points of this operation can be considered as clusters, because they contain only non-outliers (as defined by the above mentioned procedure) and all other points are outliers w.r.t. the subset.

The current default is to compute fuzzy Mahalanobis FPCs, where the points in the subset have a membership weight between 0 and 1 and give rise to weighted means and covariance matrices. The new weights are then obtained by computing the weight function `wfu` of the squared Mahalanobis distances, i.e., full weight for squared distances smaller than `ca`, zero weight for squared distances larger than `ca2` and decreasing weights (linear function of squared distances) in between.

A fixed point algorithm is started from the whole dataset, algorithms are started from the subsets specified in `init.group`, and further algorithms are started from further initial configurations as explained under `subset` and in the function `mahalconf`.

Usually some of the FPCs are unstable, and more than one FPC may correspond to the same significant pattern in the data. Therefore the number of FPCs is reduced: A similarity matrix is computed between FPCs. Similarity between sets is defined as the ratio between 2 times size of intersection and the sum of sizes of both sets. The Single Linkage clusters (groups) of level `distcut` are computed, i.e. the connectivity components of the graph where edges are drawn between FPCs with similarity larger than `distcut`. Groups of FPCs whose members are found often enough (cf. parameter `mer`) are considered as stable enough. A representative FPC is chosen for every Single Linkage cluster of FPCs according to the maximum expectation ratio `ser`. `ser` is the ratio between the number of findings of an FPC and the number of points of an FPC, adjusted suitably if `subset<n`. Usually only the representative FPCs of stable groups are of interest.

Default tuning constants are taken from Hennig (2005).

Generally, the default settings are recommended for `fixmahal`. For large datasets, the use of `init.group` together with `pointit=FALSE` is useful. Occasionally, `mnc` and `startn` may be chosen smaller than the default, if smaller clusters are of interest, but this may lead to too many clusters. Decrease of `ca` will often lead to too many clusters, even for homogeneous data. Increase of `ca` will produce only very strongly separated clusters. Both may be of interest occasionally.

Singular covariance matrices during the iterations are handled by `solvecov`.

`summary.mfpc` gives a summary about the representative FPCs of stable groups.

`plot.mfpc` is a plot method for the representative FPC of stable group `no`. `no`. It produces a scatterplot, where the points belonging to the FPC are highlighted, the mean is and for `p<3` also the region of the FPC is shown. For `p>=3`, the optimal separating projection computed by `batcoord` is shown.

`fpclusters.mfpc` produces a list of indicator vectors for the representative FPCs of stable groups.

`fpmi` is called by `fixmahal` for a single fixed point algorithm and will usually not be executed alone.

### Value

`fixmahal` returns an object of class `mfpc`. This is a list containing the components `nc`, `g`, `means`, `covs`, `nfound`, `er`, `tsc`, `ncoll`, `skc`, `grto`, `imatrix`, `smatrix`, `stn`, `stfound`, `ser`, `sfpc`, `ssig`, `sto`, `struc`, `n`, `p`, `method`, `cgen`, `ca`, `ca2`, `cvec`, `calpha`, `pointit`, `subset`, `mnc`, `startn`, `mer`, `distcut`.

`summary.mfpc` returns an object of class `summary.mfpc`. This is a list containing the components `means`, `covs`, `stn`, `stfound`, `sn`, `ser`, `tskip`, `skc`, `tsc`, `sim`, `ca`, `ca2`, `calpha`, `mer`, `method`, `cgen`, `pointit`.

`fpclusters.mfpc` returns a list of indicator vectors for the representative FPCs of stable groups.

`fpmi` returns a list with the components `mg`, `covg`, `md`, `gv`, `coll`, `method`, `ca`.

<code>nc</code>	integer. Number of FPCs.
<code>g</code>	list of logical vectors. Indicator vectors of FPCs. FALSE if <code>ind.storage=FALSE</code> .
<code>means</code>	list of numerical vectors. Means of FPCs. In <code>summary.mfpc</code> , only for representative FPCs of stable groups and sorted according to <code>ser</code> .
<code>covs</code>	list of numerical matrices. Covariance matrices of FPCs. In <code>summary.mfpc</code> , only for representative FPCs of stable groups and sorted according to <code>ser</code> .
<code>nfound</code>	vector of integers. Number of findings for the FPCs.
<code>er</code>	numerical vector. Ratio of number of findings of FPCs to their size. Under <code>pointit=TRUE</code> , this can be taken as a measure of stability of FPCs.
<code>tsc</code>	integer. Number of algorithm runs leading to too small or too seldom found FPCs.
<code>ncoll</code>	integer. Number of algorithm runs where collinear covariance matrices occurred.
<code>skc</code>	integer. Number of skipped clusters.
<code>grto</code>	vector of integers. Numbers of FPCs to which algorithm runs led, which were started by <code>init.group</code> .
<code>imatrix</code>	vector of integers. Size of intersection between FPCs. See <a href="#">sseg</a> .
<code>smatrix</code>	numerical vector. Similarities between FPCs. See <a href="#">sseg</a> .
<code>stn</code>	integer. Number of representative FPCs of stable groups. In <code>summary.mfpc</code> , sorted according to <code>ser</code> .
<code>stfound</code>	vector of integers. Number of findings of members of all groups of FPCs. In <code>summary.mfpc</code> , sorted according to <code>ser</code> .
<code>ser</code>	numerical vector. Ratio of number of findings of groups of FPCs to their size. Under <code>pointit=TRUE</code> , this can be taken as a measure of stability of FPCs. In <code>summary.mfpc</code> , sorted from largest to smallest.
<code>sfpc</code>	vector of integers. Numbers of representative FPCs of all groups.
<code>ssig</code>	vector of integers of length <code>stn</code> . Numbers of representative FPCs of the stable groups.

sto	vector of integers. Numbers of groups ordered according to largest ser.
struc	vector of integers. Number of group an FPC belongs to.
n	see arguments.
p	see arguments.
method	see arguments.
cgen	see arguments.
ca	see arguments, if cgen has been "fixed". Else numerical vector of length nc (see below), giving the final values of ca for all FPC. In fpmi, tuning constant for the iterated FPC.
ca2	see arguments.
cvec	numerical vector of length n for cgen="auto". The values for the tuning constant ca corresponding to the cluster sizes from 1 to n.
calpha	see arguments.
pointit	see arguments.
subset	see arguments.
mnc	see arguments.
startn	see arguments.
mer	see arguments.
distcut	see arguments.
sn	vector of integers. Number of points of representative FPCs.
tskip	integer. Number of algorithm runs leading to skipped FPCs.
sim	vector of integers. Size of intersections between representative FPCs of stable groups. See <a href="#">sseg</a> .
mg	mean vector.
covg	covariance matrix.
md	Mahalanobis distances.
gv	logical (numerical, respectively, if method="fuzzy") indicator vector of iterated FPC.
coll	logical. TRUE means that singular covariance matrices occurred during the iterations.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

## References

Hennig, C. (2002) Fixed point clusters for linear regression: computation and comparison, *Journal of Classification* 19, 249-276.

Hennig, C. (2005) Fuzzy and Crisp Mahalanobis Fixed Point Clusters, in Baier, D., Decker, R., and Schmidt-Thieme, L. (eds.): *Data Analysis and Decision Support*. Springer, Heidelberg, 47-56, <http://www.homepages.ucl.ac.uk/~ucakche/papers/fuzzyfix.ps>

Hennig, C. and Christlieb, N. (2002) Validating visual clusters in large datasets: Fixed point clusters of spectral features, *Computational Statistics and Data Analysis* 40, 723-739.

## See Also

[fixreg](#) for linear regression fixed point clusters.

[mahalconf](#), [wfu](#), [cmahal](#) for computation of initial configurations, weights, tuning constants.

[sseg](#) for indexing the similarity/intersection vectors computed by [fixmahal](#).

[batcoord](#), [cov.rob](#), [solvecov](#), [cov.wml](#), [plotcluster](#) for computation of projections, (inverted) covariance matrices, plotting.

[rFace](#) for generation of example data, see below.

## Examples

```
set.seed(20000)
face <- rFace(400,dMoNo=2,dNoEy=0, p=3)
# The first example uses grouping information via init.group.
initg <- list()
grface <- as.integer(attr(face,"grouping"))
for (i in 1:5) initg[[i]] <- (grface==i)
ff0 <- fixmahal(face, pointit=FALSE, init.group=initg)
summary(ff0)
cff0 <- fpclusters(ff0)
plot(face, col=1+cff0[[1]])
plot(face, col=1+cff0[[4]]) # Why does this come out as a cluster?
plot(ff0, face, 4) # A bit clearer...
# Without grouping information, examples need more time:
# ff1 <- fixmahal(face)
# summary(ff1)
# cff1 <- fpclusters(ff1)
# plot(face, col=1+cff1[[1]])
# plot(face, col=1+cff1[[6]]) # Why does this come out as a cluster?
# plot(ff1, face, 6) # A bit clearer...
# ff2 <- fixmahal(face,method="ml")
# summary(ff2)
# ff3 <- fixmahal(face,method="ml",alpha=0.95,subset=50)
# summary(ff3)
## ...fast, but lots of clusters. mer=0.3 may be useful here.
# set.seed(3000)
# face2 <- rFace(400,dMoNo=2,dNoEy=0)
# ff5 <- fixmahal(face2)
# summary(ff5)
## misses right eye of face data; with p=6,
```

```
## initial configurations are too large for 40 point clusters
# ff6 <- fixmahal(face2, startn=30)
# summary(ff6)
# cff6 <- fpclusters(ff6)
# plot(face2, col=1+cff6[[3]])
# plot(ff6, face2, 3)
# x <- c(1,2,3,6,6,7,8,120)
# ff8 <- fixmahal(x)
# summary(ff8)
# ...dataset a bit too small for the defaults...
# ff9 <- fixmahal(x, mnc=3, startn=3)
# summary(ff9)
```

---

fixreg

---

*Linear Regression Fixed Point Clusters*


---

## Description

Computes linear regression fixed point clusters (FPCs), i.e., subsets of the data, which consist exactly of the non-outliers w.r.t. themselves, and may be interpreted as generated from a homogeneous linear regression relation between independent and dependent variable. FPCs may overlap, are not necessarily exhausting and do not need a specification of the number of clusters.

Note that while `fixreg` has lots of parameters, only one (or few) of them have usually to be specified, cf. the examples. The philosophy is to allow much flexibility, but to always provide sensible defaults.

## Usage

```
fixreg(indep=rep(1,n), dep, n=length(dep),
      p=ncol(as.matrix(indep)),
      ca=NA, mnc=NA, mtf=3, ir=NA, irnc=NA,
      irprob=0.95, mncprob=0.5, maxir=20000, maxit=5*n,
      distcut=0.85, init.group=list(),
      ind.storage=FALSE, countmode=100,
      plot=FALSE)

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

```
## S3 method for class 'summary.rfpc':
print(x, maxnc=30, ...)
```

```
## S3 method for class 'rfpc':
plot(x, indep=rep(1,n), dep, no, bw=TRUE,
     main=c("Representative FPC No. ",no),
     xlab="Linear combination of independents",
     ylab=deparse(substitute(indep)),
     xlim=NULL, ylim=range(dep),
```

```

        pch=NULL, col=NULL, ...)

## S3 method for class 'rfpc':
fpclusters(object, indep=NA, dep=NA, ca=object$ca, ...)

rfpi(indep, dep, p, gv, ca, maxit, plot)

```

### Arguments

<code>indep</code>	numerical matrix or vector. Independent variables. Leave out for clustering one-dimensional data. <code>fpclusters.rfpc</code> does not need specification of <code>indep</code> if <code>fixreg</code> was run with <code>ind.storage=TRUE</code> .
<code>dep</code>	numerical vector. Dependent variable. <code>fpclusters.rfpc</code> does not need specification of <code>dep</code> if <code>fixreg</code> was run with <code>ind.storage=TRUE</code> .
<code>n</code>	optional positive integer. Number of cases.
<code>p</code>	optional positive integer. Number of independent variables.
<code>ca</code>	optional positive number. Tuning constant, specifying required cluster separation. By default determined automatically as a function of <code>n</code> and <code>p</code> , see function <a href="#">can</a> , Hennig (2002a).
<code>mnc</code>	optional positive integer. Minimum size of clusters to be reported. By default determined automatically as a function of <code>mncprob</code> . See Hennig (2002a).
<code>mtf</code>	optional positive integer. FPCs must be found at least <code>mtf</code> times to be reported by <code>summary.rfpc</code> .
<code>ir</code>	optional positive integer. Number of algorithm runs. By default determined automatically as a function of <code>n</code> , <code>p</code> , <code>irnc</code> , <code>irprob</code> , <code>mtf</code> , <code>maxir</code> . See function <a href="#">itnumber</a> and Hennig (2002a).
<code>irnc</code>	optional positive integer. Size of the smallest cluster to be found with approximated probability <code>irprob</code> .
<code>irprob</code>	optional value between 0 and 1. Approximated probability for a cluster of size <code>irnc</code> to be found.
<code>mncprob</code>	optional value between 0 and 1. Approximated probability for a cluster of size <code>mnc</code> to be found.
<code>maxir</code>	optional integer. Maximum number of algorithm runs.
<code>maxit</code>	optional integer. Maximum number of iterations per algorithm run (usually an FPC is found much earlier).
<code>distcut</code>	optional value between 0 and 1. A similarity measure between FPCs, given in Hennig (2002a), and the corresponding Single Linkage groups of FPCs with similarity larger than <code>distcut</code> are computed. A single representative FPC is selected for each group.
<code>init.group</code>	optional list of logical vectors of length <code>n</code> . Every vector indicates a starting configuration for the fixed point algorithm. This can be used for datasets with high dimension, where the vectors of <code>init.group</code> indicate cluster candidates found by graphical inspection or background knowledge.
<code>ind.storage</code>	optional logical. If <code>TRUE</code> , then all indicator vectors of found FPCs are given in the value of <code>fixreg</code> . May need lots of memory, but is a bit faster.

<code>countmode</code>	optional positive integer. Every <code>countmode</code> algorithm runs <code>fixreg</code> shows a message.
<code>plot</code>	optional logical. If <code>TRUE</code> , you get a scatterplot of first independent vs. dependent variable at each iteration.
<code>object</code>	object of class <code>rfpc</code> , output of <code>fixreg</code> .
<code>x</code>	object of class <code>rfpc</code> , output of <code>fixreg</code> .
<code>maxnc</code>	positive integer. Maximum number of FPCs to be reported.
<code>no</code>	positive integer. Number of the representative FPC to be plotted.
<code>bw</code>	optional logical. If <code>TRUE</code> , plot is black/white, FPC is indicated by different symbol. Else FPC is indicated red.
<code>main</code>	plot title.
<code>xlab</code>	label for x-axis.
<code>ylab</code>	label for y-axis.
<code>xlim</code>	plotted range of x-axis. If <code>NULL</code> , the range of the plotted linear combination of independent variables is used.
<code>ylim</code>	plotted range of y-axis.
<code>pch</code>	plotting symbol, see <code>par</code> . If <code>NULL</code> , the default is used.
<code>col</code>	plotting color, see <code>par</code> . If <code>NULL</code> , the default is used.
<code>gv</code>	logical vector of length <code>n</code> . Indicates the initial configuration for the fixed point algorithm.
<code>...</code>	additional parameters to be passed to <code>plot</code> (no effects elsewhere).

## Details

A linear regression FPC is a data subset that reproduces itself under the following operation: Compute linear regression and error variance estimator for the data subset, and compute all points of the dataset for which the squared residual is smaller than `ca` times the error variance.

Fixed points of this operation can be considered as clusters, because they contain only non-outliers (as defined by the above mentioned procedure) and all other points are outliers w.r.t. the subset.

`fixreg` performs `ir` fixed point algorithms started from random subsets of size `p+2` to look for FPCs. Additionally an algorithm is started from the whole dataset, and algorithms are started from the subsets specified in `init.group`.

Usually some of the FPCs are unstable, and more than one FPC may correspond to the same significant pattern in the data. Therefore the number of FPCs is reduced: FPCs with less than `mnc` points are ignored. Then a similarity matrix is computed between the remaining FPCs. Similarity between sets is defined as the ratio between 2 times size of intersection and the sum of sizes of both sets. The Single Linkage clusters (groups) of level `distcut` are computed, i.e. the connectivity components of the graph where edges are drawn between FPCs with similarity larger than `distcut`. Groups of FPCs whose members are found `mtf` times or more are considered as stable enough. A representative FPC is chosen for every Single Linkage cluster of FPCs according to the maximum expectation ratio `ser`. `ser` is the ratio between the number of findings of an FPC and the estimated expectation of the number of findings of an FPC of this size, called *expectation ratio* and computed by `clusexpect`.

Usually only the representative FPCs of stable groups are of interest.

The choice of the involved tuning constants such as `ca` and `ir` is discussed in detail in Hennig (2002a). Statistical theory is presented in Hennig (2003).

Generally, the default settings are recommended for `fixreg`. In cases where they lead to a too large number of algorithm runs (e.g., always for  $p > 4$ ), the use of `init.group` together with `mtf=1` and `ir=0` is useful. Occasionally, `irnc` may be chosen smaller than the default, if smaller clusters are of interest, but this may lead to too many clusters and too many algorithm runs. Decrease of `ca` will often lead to too many clusters, even for homogeneous data. Increase of `ca` will produce only very strongly separated clusters. Both may be of interest occasionally.

`rfpi` is called by `fixreg` for a single fixed point algorithm and will usually not be executed alone.

`summary.rfpc` gives a summary about the representative FPCs of stable groups.

`plot.rfpc` is a plot method for the representative FPC of stable group `no.` It produces a scatterplot of the linear combination of independent variables determined by the regression coefficients of the FPC vs. the dependent variable. The regression line and the region of non-outliers determined by `ca` are plotted as well.

`fpclusters.rfpc` produces a list of indicator vectors for the representative FPCs of stable groups.

## Value

`fixreg` returns an object of class `rfpc`. This is a list containing the components `nc`, `g`, `coefs`, `vars`, `nfound`, `er`, `tsc`, `ncoll`, `grto`, `imatrix`, `smatrix`, `stn`, `stfound`, `sfpc`, `ssig`, `sto`, `struc`, `n`, `p`, `ca`, `ir`, `mnc`, `mtf`, `distcut`.

`summary.rfpc` returns an object of class `summary.rfpc`. This is a list containing the components `coefs`, `vars`, `stfound`, `stn`, `sn`, `ser`, `tsc`, `sim`, `ca`, `ir`, `mnc`, `mtf`.

`fpclusters.rfpc` returns a list of indicator vectors for the representative FPCs of stable groups.

`rfpi` returns a list with the components `coef`, `var`, `g`, `coll`, `ca`.

<code>nc</code>	integer. Number of FPCs.
<code>g</code>	list of logical vectors. Indicator vectors of FPCs. FALSE if <code>ind.storage=FALSE</code> .
<code>coefs</code>	list of numerical vectors. Regression coefficients of FPCs. In <code>summary.rfpc</code> , only for representative FPCs of stable groups and sorted according to <code>stfound</code> .
<code>vars</code>	list of numbers. Error variances of FPCs. In <code>summary.rfpc</code> , only for representative FPCs of stable groups and sorted according to <code>stfound</code> .
<code>nfound</code>	vector of integers. Number of findings for the FPCs.
<code>er</code>	numerical vector. Expectation ratios of FPCs. Can be taken as a stability measure.
<code>tsc</code>	integer. Number of algorithm runs leading to too small or too seldom found FPCs.
<code>ncoll</code>	integer. Number of algorithm runs where collinear regressor matrices occurred.
<code>grto</code>	vector of integers. Numbers of FPCs to which algorithm runs led, which were started by <code>init.group</code> .
<code>imatrix</code>	vector of integers. Size of intersection between FPCs. See <a href="#">sseg</a> .
<code>smatrix</code>	numerical vector. Similarities between FPCs. See <a href="#">sseg</a> .

stn	integer. Number of representative FPCs of stable groups. In <code>summary.rfpc</code> sorted according to <code>stfound</code> .
stfound	vector of integers. Number of findings of members of all groups of FPCs. In <code>summary.rfpc</code> sorted according to <code>stfound</code> .
sfpc	vector of integers. Numbers of representative FPCs.
ssig	vector of integers. As <code>sfpc</code> , but only for stable groups.
sto	vector of integers. Number of representative FPC of most, 2nd most, ..., often found group of FPCs.
struc	vector of integers. Number of group an FPC belongs to.
n	see arguments.
p	see arguments.
ca	see arguments.
ir	see arguments.
mnc	see arguments.
mtf	see arguments.
distcut	see arguments.
sn	vector of integers. Number of points of representative FPCs.
ser	numerical vector. Expectation ratio for stable groups.
sim	vector of integers. Size of intersections between representative FPCs of stable groups. See <a href="#">sseg</a> .
coef	vector of regression coefficients.
var	error variance.
g	logical indicator vector of iterated FPC.
coll	logical. TRUE means that singular covariance matrices occurred during the iterations.

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

- Hennig, C. (2002) Fixed point clusters for linear regression: computation and comparison, *Journal of Classification* 19, 249-276.
- Hennig, C. (2003) Clusters, outliers and regression: fixed point clusters, *Journal of Multivariate Analysis* 86, 183-212.

**See Also**

[fixmahal](#) for fixed point clusters in the usual setup (non-regression).  
[regmix](#) for clusterwise linear regression by mixture modeling ML.  
[can](#), [itnumber](#) for computation of the default settings.  
[clusexpect](#) for estimation of the expected number of findings of an FPC of given size.  
[itnumber](#) for the generation of the number of fixed point algorithms.  
[minsize](#) for the smallest FPC size to be found with a given probability..  
[sseg](#) for indexing the similarity/intersection vectors computed by `fixreg`.

**Examples**

```
set.seed(190000)
data(tonedata)
# Note: If you do not use the installed package, replace this by
# tonedata <- read.table("(path/)tonedata.txt", header=TRUE)
attach(tonedata)
tonefix <- fixreg(stretchratio,tuned,mtf=1,ir=20)
summary(tonefix)
# This is designed to have a fast example; default setting would be better.
# If you want to see more (and you have a bit more time),
# try out the following:
# set.seed(1000)
# tonefix <- fixreg(stretchratio,tuned)
## Default - good for these data
# summary(tonefix)
# plot(tonefix,stretchratio,tuned,1)
# plot(tonefix,stretchratio,tuned,2)
# plot(tonefix,stretchratio,tuned,3,bw=FALSE,pch=5)
# toneclus <- fpclusters(tonefix,stretchratio,tuned)
# plot(stretchratio,tuned,col=1+toneclus[[2]])
# tonefix2 <- fixreg(stretchratio,tuned,distcut=1,mtf=1,countmode=50)
## Every found fixed point cluster is reported,
## no matter how instable it may be.
# summary(tonefix2)
# tonefix3 <- fixreg(stretchratio,tuned,ca=7)
## ca defaults to 10.07 for these data.
# summary(tonefix3)
# subset <- c(rep(FALSE,5),rep(TRUE,24),rep(FALSE,121))
# tonefix4 <- fixreg(stretchratio,tuned,
#                   mtf=1,ir=0,init.group=list(subset))
# summary(tonefix4)
```

**Description**

`fpclusters` is a generic function which extracts the representative fixed point clusters (FPCs) from FPC objects generated by `fixmahal` and `fixreg`. For documentation and examples see `fixmahal` and `fixreg`.

**Usage**

```
fpclusters(object, ...)
```

**Arguments**

`object`            object of class `rfpc` or `mfp`.  
`...`                further arguments depending on the method.

**Value**

a list of logical or numerical vectors indicating or giving the weights of the cluster memberships.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

`fixmahal`, `fixreg`

---

`itnumber`

*Number of regression fixed point cluster iterations*

---

**Description**

Computes the number of fixed point iterations needed by `fixreg` to find `mtf` times a fixed point cluster (FPC) of size `cn` with an approximated probability of `prob`.

Thought for use within `fixreg`.

**Usage**

```
itnumber(n, p, cn, mtf, prob = 0.95, maxir = 20000)
```

**Arguments**

`n`                    positive integer. Total number of points.  
`p`                    positive integer. Number of independent variables.  
`cn`                   positive integer smaller or equal to `n`. Size of the FPC.  
`mtf`                  positive integer.  
`prob`                 number between 0 and 1.  
`maxir`                positive integer. `itnumber` is set to this value if it would otherwise be larger.

### Details

The computation is based on the binomial distribution with probability given by `clusexpect` with `ir=1`.

### Value

An integer.

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

Hennig, C. (2002) Fixed point clusters for linear regression: computation and comparison, *Journal of Classification* 19, 249-276.

### See Also

`fixreg`, `clusexpect`

### Examples

```
itnumber(500, 4, 150, 2)
```

---

kmeansCBI

*Interface functions for clustering methods*

---

### Description

These functions provide an interface to several clustering methods implemented in R, for use together with the cluster stability assessment in `clusterboot` (as parameter `clustermethod`; "CBI" stands for "clusterboot interface"). In some situations it could make sense to use them to compute a clustering even if you don't want to run `clusterboot`, because some of the functions contain some additional features (e.g., normal mixture model based clustering of dissimilarity matrices projected into the Euclidean space by MDS or partitioning around medoids with estimated number of clusters, noise/outlier identification in hierarchical clustering).

### Usage

```
kmeansCBI(data, k, scaling=TRUE, runs=1, ...)
```

```
hclustCBI(data, k, cut="level", method, scaling=TRUE, noise=0, ...)
```

```
hclusttreeCBI(data, minlevel=2, method, scaling=TRUE, ...)
```

```
disthclustCBI(dmatrix, k, cut="level", method, noise=0, ...)
```

```

noisemclustCBI (data, G, emModelNames, nnk, hcmodeL=NULL, Vinv=NULL,
               summary.out=FALSE, ...)

distnoisemclustCBI (dmatrix, G, emModelNames, nnk,
                   hcmodeL=NULL, Vinv=NULL, mdsmethod="classical",
                   mdsdim=4, summary.out=FALSE, points.out=FALSE, ...)

claraCBI (data, k, usepam=TRUE, diss=FALSE, ...)

pamkCBI (data, krange=2:10, scaling=TRUE, diss=FALSE, ...)

trimkmeansCBI (data, k, scaling=TRUE, trim=0.1, ...)

distrtrimkmeansCBI (dmatrix, k, scaling=TRUE, trim=0.1,
                   mdsmethod="classical",
                   mdsdim=4, ...)

dbscanCBI (data, eps, MinPts, diss=FALSE, ...)

mahalCBI (data, clustercut=0.5, ...)

```

### Arguments

<code>data</code>	a numeric matrix. The data matrix - usually a cases*variables-data matrix. <code>claraCBI</code> , <code>pamkCBI</code> and <code>dbscanCBI</code> work with an $n \times n$ -dissimilarity matrix as well, see parameter <code>diss</code> .
<code>dmatrix</code>	a squared numerical dissimilarity matrix or a <code>dist</code> -object.
<code>k</code>	numeric, usually integer. In most cases, this is the number of clusters for methods where this is fixed. For <code>hclustCBI</code> and <code>disthclustCBI</code> see parameter <code>cut</code> below.
<code>scaling</code>	either a logical value or a numeric vector of length equal to the number of variables. If <code>scaling</code> is a numeric vector with length equal to the number of variables, then each variable is divided by the corresponding value from <code>scaling</code> . If <code>scaling</code> is <code>TRUE</code> then scaling is done by dividing the (centered) variables by their root-mean-square, and if <code>scaling</code> is <code>FALSE</code> , no scaling is done before execution.
<code>runs</code>	integer. Number of random initializations from which the k-means algorithm is started.
<code>cut</code>	either "level" or "number". This determines how <code>cutree</code> is used to obtain a partition from a hierarchy tree. <code>cut="level"</code> means that the tree is cut at a particular dissimilarity level, <code>cut="number"</code> means that the tree is cut in order to obtain a fixed number of clusters. The parameter <code>k</code> specifies the number of clusters or the dissimilarity level, depending on <code>cut</code> .
<code>method</code>	method for hierarchical clustering, see the documentation of <code>hclust</code> .
<code>noiseCut</code>	numeric. All clusters of size $\leq$ <code>noiseCut</code> in the <code>disthclustCBI</code> / <code>hclustCBI</code> -partition are joined and declared as noise/outliers.

minlevel	integer. minlevel=1 means that all clusters in the tree are given out by <code>hclusttreeCBI</code> , including one-point clusters (but excluding the cluster with all points). minlevel=2 excludes the one-point clusters. minlevel=3 excludes the two-point cluster which has been merged first, and increasing the value of minlevel by 1 in all further steps means that the remaining earliest formed cluster is excluded.
G	vector of integers. Number of clusters or numbers of clusters used by <code>mclustBIC</code> . If G has more than one entry, the number of clusters is estimated by the BIC.
emModelNames	vector of string. Models for covariance matrices, see documentation of <code>mclustBIC</code> .
nnk	integer. Tuning constant for <code>NNclean</code> , which is used to estimate the initial noise for <code>noisemclustCBI</code> and <code>distnoisemclustCBI</code> . See parameter k in the documentation of <code>NNclean</code> . nnk=0 means that no noise component is fitted.
hcmode1	string or NULL. Determines the initialization of the EM-algorithm for <code>mclustBIC</code> . Documented in <code>hc</code> .
Vinv	numeric. See documentation of <code>mclustBIC</code> .
summary.out	logical. If TRUE, the result of <code>summary.mclustBIC</code> is added as component <code>mclustsummary</code> to the output of <code>noisemclustCBI</code> and <code>distnoisemclustCBI</code> .
mdsmethod	"classical", "kruskal" or "sammon". Determines the multidimensional scaling method to compute Euclidean data from a dissimilarity matrix. See <code>cmdscale</code> , <code>isoMDS</code> and <code>sammon</code> .
mdsdim	integer. Dimensionality of MDS solution.
points.out	logical. If TRUE, the matrix of MDS points is added as component <code>points</code> to the output of <code>noisemclustCBI</code> .
usepam	logical. If TRUE, the function <code>pam</code> is used for clustering, otherwise <code>clara</code> . <code>pam</code> is better, <code>clara</code> is faster.
diss	logical. If TRUE, data will be considered as a dissimilarity matrix. in <code>claraCBI</code> , this requires <code>usepam=TRUE</code> .
krange	vector of integers. Numbers of clusters to be compared.
trim	numeric between 0 and 1. Proportion of data points trimmed, i.e., assigned to noise. See <code>trimkmeans</code> .
eps	numeric. The radius of the neighborhoods to be considered by <code>dbscan</code> .
MinPts	integer. How many points have to be in a neighborhood so that a point is considered to be a cluster seed? See documentation of <code>dbscan</code> .
clustercut	numeric between 0 and 1. If <code>fixmahal</code> is used for fuzzy clustering, a crisp partition is generated and points with cluster membership values above <code>clustercut</code> are considered as members of the corresponding cluster.
...	further parameters to be transferred to the original clustering functions (not required).

## Details

All these functions call clustering methods implemented in R to cluster data and to provide output in the format required by `clusterboot`. Here is a brief overview:

- `kmeansCBI` interface to the function `kmeansruns` calling `kmeans` for k-means clustering. (`kmeansruns` allows the specification of several random initializations of the k-means algorithm.)
- `hclustCBI` interface to the function `hclust` for agglomerative hierarchical clustering with noise component (see parameter `noisecut` above). This function produces a partition and assumes a `cases*variables` matrix as input.
- `hclusttreeCBI` interface to the function `hclust` for agglomerative hierarchical clustering. This function gives out all clusters belonging to the hierarchy (upward from a certain level, see parameter `minlevel` above).
- `disthclustCBI` interface to the function `hclust` for agglomerative hierarchical clustering with noise component (see parameter `noisecut` above). This function produces a partition and assumes a dissimilarity matrix as input.
- `noisemclustCBI` interface to the function `mclustBIC`, for normal mixture model based clustering. Warning: `mclustBIC` often has problems with multiple points. In `clusterboot`, it is recommended to use this together with `multipleboot=FALSE`.
- `distnoisemclustCBI` interface to the function `mclustBIC` for normal mixture model based clustering. This assumes a dissimilarity matrix as input and generates a data matrix by multidimensional scaling first. Warning: `mclustBIC` often has problems with multiple points. In `clusterboot`, it is recommended to use this together with `multipleboot=FALSE`.
- `claraCBI` interface to the functions `pam` and `clara` for partitioning around medoids.
- `pamkCBI` interface to the function `pamk` calling `pam` for partitioning around medoids. The number of clusters is estimated by the average silhouette width.
- `trimkmeansCBI` interface to the function `trimkmeans` for trimmed k-means clustering. This assumes a `cases*variables` matrix as input.
- `disttrimkmeansCBI` interface to the function `trimkmeans` for trimmed k-means clustering. This assumes a dissimilarity matrix as input and generates a data matrix by multidimensional scaling first.
- `dbscanCBI` interface to the function `dbscan` for density based clustering.
- `mahalCBI` interface to the function `fixmahal` for fixed point clustering.

## Value

All interface functions return a list with the following components (there may be some more, see `summary.out` and `points.out` above):

<code>result</code>	clustering result, usually a list with the full output of the clustering method (the precise format doesn't matter); whatever you want to use later.
<code>nc</code>	number of clusters. If some points don't belong to any cluster but are declared as "noise", <code>nc</code> includes the noise component, and there should be another component <code>nccl</code> , being the number of clusters not including the noise component.
<code>clusterlist</code>	this is a list consisting of a logical vectors of length of the number of data points ( <code>n</code> ) for each cluster, indicating whether a point is a member of this cluster (TRUE) or not. If a noise component is included, it should always be the last vector in this list.

`partition` an integer vector of length `n`, partitioning the data. If the method produces a partition, it should be the clustering. This component is only used for plots, so you could do something like `rep(1, n)` for non-partitioning methods.

`clustermethod` a string indicating the clustering method.

The output of some of the functions has further components:

`nccl` see `nc` above.

`nnk` by `noisemclustCBI` and `distnoisemclustCBI`, see above.

`initnoise` logical vector, indicating initially estimated noise by `NNclean`, called by `noisemclustCBI` and `distnoisemclustCBI`.

`noise` logical. TRUE if points were classified as noise/outliers by `dsthclustCBI`.

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### See Also

[clusterboot](#), [dist](#), [kmeans](#), [kmeansruns](#), [hclust](#), [mclustBIC](#), [pam](#), [pamk](#), [clara](#), [trimkmeans](#), [dbscan](#), [fixmahal](#)

### Examples

```
set.seed(20000)
face <- rFace(50, dMoNo=2, dNoEy=0, p=2)
dbs <- dbscanCBI(face, eps=1.5, MinPts=4)
dhc <- dsthclustCBI(dist(face), method="average", k=1.5, noise=2)
table(dbs$partition, dhc$partition)
```

---

kmeansruns

*k-means clustering with several random initializations*

---

### Description

This calls the function `kmeans` to perform a k-means clustering, but initializes the k-means algorithm several times with random points from the data set as means. Furthermore, it is more robust against the occurrence of empty clusters in the algorithm.

### Usage

```
kmeansruns(data, k, iter.max=100, runs=100, scaledata=FALSE, plot=FALSE)
```

**Arguments**

data	A numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
k	integer. The number of clusters.
iter.max	integer. The maximum number of iterations allowed.
runs	integer. Number of starts of the k-means algorithm.
scaledata	logical. If TRUE, the variables are centered and scaled to unit variance before execution.
plot	logical. If TRUE, every clustering resulting from a run of the algorithm is plotted.

**Value**

The output of the optimal run of the `kmeans`-function. A list with components

cluster	A vector of integers indicating the cluster to which each point is allocated.
centers	A matrix of cluster centers.
withinss	The within-cluster sum of squares for each cluster.
size	The number of points in each cluster.

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[kmeans](#)

**Examples**

```
set.seed(20000)
face <- rFace(50, dMoNo=2, dNoEy=0, p=2)
kmr1 <- kmeansruns(face, k=5, runs=1)
kmr5 <- kmeansruns(face, k=5, runs=5)
```

---

mahalanodisc      *Mahalanobis for AWC*

---

**Description**

Vector of Mahalanobis distances or their root. For use in `awcoord` only.

**Usage**

```
mahalanodisc(x2, mg, covg, modus="square")
```

**Arguments**

<code>x2</code>	numerical data matrix.
<code>mg</code>	mean vector.
<code>covg</code>	covariance matrix.
<code>modus</code>	"md" (roots of Mahalanobis distances) or "square" (original squared form of Mahalanobis distances).

**Details**

The covariance matrix is inverted by use of `solvecov`, which can be expected to give reasonable results for singular within-class covariance matrices.

**Value**

vector of (rooted) Mahalanobis distances.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[awcoord](#), [solvecov](#)

**Examples**

```
x <- cbind(rnorm(50), rnorm(50))
mahalanodisc(x, c(0, 0), cov(x))
mahalanodisc(x, c(0, 0), matrix(0, ncol=2, nrow=2))
```

mahalanofix

*Mahalanobis distances from center of indexed points***Description**

Computes the vector of (classical or robust) Mahalanobis distances of all points of `x` to the center of the points indexed (or weighted) by `gv`. The latter also determine the covariance matrix.

Thought for use within [fixmahal](#).

**Usage**

```
mahalanofix(x, n = nrow(as.matrix(x)), p = ncol(as.matrix(x)), gv =
rep(1, times = n), cmax = 1e+10, method = "ml")
```

```
mahalanofuz(x, n = nrow(as.matrix(x)), p = ncol(as.matrix(x)),
gv = rep(1, times=n), cmax = 1e+10)
```

**Arguments**

<code>x</code>	a numerical data matrix, rows are points, columns are variables.
<code>n</code>	positive integer. Number of points.
<code>p</code>	positive integer. Number of variables.
<code>gv</code>	for <code>mahalanofix</code> a logical or 0-1 vector of length <code>n</code> . For <code>mahalanofuz</code> a numerical vector with values between 0 and 1.
<code>cmax</code>	positive number. used in <a href="#">solvecov</a> if covariance matrix is singular.
<code>method</code>	"ml", "classical", "mcd" or "mve". Method to compute the covariance matrix estimator. See <a href="#">cov.rob</a> , <a href="#">fixmahal</a> .

**Details**

[solvecov](#) is used to invert the covariance matrix. The methods "mcd" and "mve" in `mahalanofix` do not work properly with point constellations with singular covariance matrices!

**Value**

A list of the following components:

<code>md</code>	vector of Mahalanobis distances.
<code>mg</code>	mean of the points indexed by <code>gv</code> , weighted mean in <code>mahalanofuz</code> .
<code>covg</code>	covariance matrix of the points indexed by <code>gv</code> , weighted covariance matrix in <code>mahalanofuz</code> .
<code>covinv</code>	<code>covg</code> inverted by <a href="#">solvecov</a> .
<code>coll</code>	logical. If TRUE, <code>covg</code> has been (numerically) singular.

**Note**

Methods "mcd" and "mve" require library lqs.

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[fixmahal](#), [solvecov](#), [cov.rob](#)

**Examples**

```
x <- c(1,2,3,4,5,6,7,8,9,10)
y <- c(1,2,3,8,7,6,5,8,9,10)
mahalanofix(cbind(x,y),gv=c(0,0,0,1,1,1,1,1,0,0))
mahalanofix(cbind(x,y),gv=c(0,0,0,1,1,1,1,0,0,0))
mahalanofix(cbind(x,y),gv=c(0,0,0,1,1,1,1,1,0,0),method="mcd")
mahalanofuz(cbind(x,y),gv=c(0,0,0.5,0.5,1,1,1,0.5,0.5,0))
```

---

mahalconf

*Mahalanobis fixed point clusters initial configuration*

---

**Description**

Generates an initial configuration of `startn` points from dataset `x` for the [fixmahal](#) fixed point iteration.

Thought only for use within [fixmahal](#).

**Usage**

```
mahalconf(x, no, startn, covall, plot)
```

**Arguments**

<code>x</code>	numerical matrix. Rows are points, columns are variables.
<code>no</code>	integer between 1 and <code>nrow(x)</code> . Number of the first point of the configuration.
<code>startn</code>	integer between 1 and <code>nrow(x)</code> .
<code>covall</code>	covariance matrix for the computation of the first Mahalanobis distances.
<code>plot</code>	a string. If equal to "start" or "both", the first two variables and the first <code>ncol(x)+1</code> points are plotted.

**Details**

`mahalconf` first chooses the  $p$  (number of variables) nearest points to point `no` in terms of the Mahalanobis distance w.r.t. `covall`, so that there are  $p + 1$  points. In every further step, the covariance matrix of the current configuration is computed and the nearest point in terms of the new Mahalanobis distance is added. `solvecov` is used to invert singular covariance matrices.

**Value**

A logical vector of length `nrow(x)`.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[fixmahal](#), [solvecov](#)

**Examples**

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0, p=2)
mahalconf(face, no=200, startn=20, covall=cov(face), plot="start")
```

---

minsize

*Minimum size of regression fixed point cluster*

---

**Description**

Computes the minimum size of a fixed point cluster (FPC) which is found at least `mtf` times with approximated probability `prob` by `ir` fixed point iterations of [fixreg](#).

Thought for use within [fixreg](#).

**Usage**

```
minsize(n, p, ir, mtf, prob = 0.5)
```

**Arguments**

<code>n</code>	positive integer. Total number of points.
<code>p</code>	positive integer. Number of independent variables.
<code>ir</code>	positive integer. Number of fixed point iterations.
<code>mtf</code>	positive integer.
<code>prob</code>	numerical between 0 and 1.

**Details**

The computation is based on the binomial distribution with probability given by `clusexpect` with `ir=1`.

**Value**

An integer.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Hennig, C. (2002) Fixed point clusters for linear regression: computation and comparison, *Journal of Classification* 19, 249-276.

**See Also**

`fixreg`, `clusexpect`, `itnumber`

**Examples**

```
minsize(500, 4, 7000, 2)
```

---

mvdcoord

*Mean/variance differences discriminant coordinates*

---

**Description**

Discriminant projections as defined in Young, Marco and Odell (1987). The principle is to maximize the projection of a matrix consisting of the differences between the means of all classes and the first mean and the differences between the covariance matrices of all classes and the first covariance matrix.

**Usage**

```
mvdcoord(xd, clvecd, clnum=1, sphere="mcd", ...)
```

**Arguments**

`xd` the data matrix; a numerical object which can be coerced to a matrix.  
`clvecd` integer vector of class numbers; length must equal `nrow(xd)`.  
`clnum` integer. Number of the class to which all differences are computed.

sphere	a covariance matrix or one of "mve", "mcd", "classical", "none". The matrix used for sphering the data. "mcd" and "mve" are robust covariance matrices as implemented in <code>cov.rob</code> . "classical" refers to the classical covariance matrix. "none" means no sphering and use of the raw data.
...	no effect

### Value

List with the following components

ev	eigenvalues in descending order.
units	columns are coordinates of projection basis vectors. New points $x$ can be projected onto the projection basis vectors by <code>x %*% units</code>
proj	projections of <code>xd</code> onto <code>units</code> .

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

Young, D. M., Marco, V. R. and Odell, P. L. (1987). Quadratic discrimination: some results on optimal low-dimensional representation, *Journal of Statistical Planning and Inference*, 17, 307-319.

### See Also

[plotcluster](#) for straight forward discriminant plots. [discrproj](#) for alternatives. [rFace](#) for generation of the example data used below.

### Examples

```
set.seed(4634)
face <- rFace(300, dMoNo=2, dNoEy=0, p=3)
grface <- as.integer(attr(face, "grouping"))
mcf <- mvdcoord(face, grface)
plot(mcf$proj, col=grface)
# ...done in one step by function plotcluster.
```

---

ncoord *Neighborhood based discriminant coordinates*

---

### Description

Neighborhood based discriminant coordinates as defined in Hastie and Tibshirani (1996) and a robustified version as defined in Hennig (2003). The principle is to maximize the projection of a between classes covariance matrix, which is defined by averaging the between classes covariance matrices in the neighborhoods of all points.

### Usage

```
ncoord(xd, clvecd, nn=50, weighted=FALSE,
       sphere="mcd", orderall=TRUE, countmode=1000, ...)
```

### Arguments

xd	the data matrix; a numerical object which can be coerced to a matrix.
clvecd	integer vector of class numbers; length must equal <code>nrow(xd)</code> .
nn	integer. Number of points which belong to the neighborhood of each point (including the point itself).
weighted	logical. <code>FALSE</code> corresponds to the original method of Hastie and Tibshirani (1996). If <code>TRUE</code> , the between classes covariance matrices <code>B</code> are weighted by <code>w/trace B</code> , where <code>w</code> is some weight depending on the sizes of the classes in the neighborhood. Division by <code>trace B</code> reduces the effect of outliers. <code>TRUE</code> corresponds to WNC as defined in Hennig (2003).
sphere	a covariance matrix or one of "mve", "mcd", "classical", "none". The matrix used for sphering the data. "mcd" and "mve" are robust covariance matrices as implemented in <code>cov.rob</code> . "classical" refers to the classical covariance matrix. "none" means no sphering and use of the raw data.
orderall	logical. By default, the neighborhoods are computed by ordering all points each time. If <code>FALSE</code> , the neighborhoods are computed by selecting <code>nn</code> times the nearest point from the remaining points, which may be faster sometimes.
countmode	optional positive integer. Every <code>countmode</code> algorithm runs <code>ncoord</code> shows a message.
...	no effect

### Value

List with the following components

ev	eigenvalues in descending order.
units	columns are coordinates of projection basis vectors. New points <code>x</code> can be projected onto the projection basis vectors by <code>x %*% units</code>
proj	projections of <code>xd</code> onto <code>units</code> .

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Hastie, T. and Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18, 607-616.

Hennig, C. (2004) Asymmetric linear dimension reduction for classification. *Journal of Computational and Graphical Statistics* 13, 930-945 .

Hennig, C. (2005) A method for visual cluster validation. In: Weihs, C. and Gaul, W. (eds.): *Classification - The Ubiquitous Challenge*. Springer, Heidelberg 2005, 153-160.

**See Also**

[plotcluster](#) for straight forward discriminant plots. [discrproj](#) for alternatives. [rFace](#) for generation of the example data used below.

**Examples**

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
ncf <- ncoord(face, grface)
plot(ncf$proj, col=grface)
ncf2 <- ncoord(face, grface, weighted=TRUE)
plot(ncf2$proj, col=grface)
# ...done in one step by function plotcluster.
```

---

pamk

*Partitioning around medoids with estimation of number of clusters*

---

**Description**

This calls the function [pam](#) to perform a partitioning around medoids clustering with the number of clusters estimated by optimum average silhouette width.

**Usage**

```
pamk(data, krange=2:10, scaling=FALSE, diss=inherits(data, "dist"), ...)
```

**Arguments**

data	a data matrix or data frame, or dissimilarity matrix or object. See <a href="#">pam</a> for more information.
krange	integer vector. Numbers of clusters which are to be compared by the average silhouette width criterion. Note: This can't estimate number of clusters <code>nc=1</code> , and therefore 1 should not be in <code>krange</code> .
scaling	either a logical value or a numeric vector of length equal to the number of variables. If <code>scaling</code> is a numeric vector with length equal to the number of variables, then each variable is divided by the corresponding value from <code>scaling</code> . If <code>scaling</code> is TRUE then scaling is done by dividing the (centered) variables by their root-mean-square, and if <code>scaling</code> is FALSE, no scaling is done.
diss	logical flag: if TRUE (default for <code>dist</code> or <code>dissimilarity</code> -objects), then <code>data</code> will be considered as a dissimilarity matrix. If FALSE, then <code>data</code> will be considered as a matrix of observations by variables.
...	further arguments to be transferred to <a href="#">pam</a> .

**Value**

A list with components

pamobject	The output of the optimal run of the <a href="#">pam</a> -function.
nc	the optimal number of clusters.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**References**

Kaufman, L. and Rousseeuw, P.J. (1990). "Finding Groups in Data: An Introduction to Cluster Analysis". Wiley, New York.

**See Also**

[pam](#)

**Examples**

```
set.seed(20000)
face <- rFace(50, dMoNo=2, dNoEy=0, p=2)
pk <- pamk(face)
```

---

plotcluster                      *Discriminant projection plot.*

---

### Description

Plots to distinguish given classes by ten available projection methods. Includes classical discriminant coordinates, methods to project differences in mean and covariance structure, asymmetric methods (separation of a homogeneous class from a heterogeneous one), local neighborhood-based methods and methods based on robust covariance matrices. One-dimensional data is plotted against the cluster number.

### Usage

```
plotcluster(x, clvecd, clnum=NULL,
            method=ifelse(is.null(clnum), "dc", "awc"),
            bw=FALSE,
            ignorepoints=FALSE, ignorenum=0, pointsbyclvecd=TRUE,
            xlab=NULL, ylab=NULL,
            pch=NULL, col=NULL, ...)
```

### Arguments

x	the data matrix; a numerical object which can be coerced to a matrix.
clvecd	vector of class numbers which can be coerced into integers; length must equal <code>nrow(xd)</code> .
method	one of <b>"dc"</b> usual discriminant coordinates, see <a href="#">discrcoord</a> , <b>"bc"</b> Bhattacharyya coordinates, first coordinate showing mean differences, second showing covariance matrix differences, see <a href="#">batcoord</a> , <b>"vbc"</b> variance dominated Bhattacharyya coordinates, see <a href="#">batcoord</a> , <b>"mvdc"</b> added mean and variance differences optimizing coordinates, see <a href="#">mvdcoord</a> , <b>"adc"</b> asymmetric discriminant coordinates, see <a href="#">adcoord</a> , <b>"awc"</b> asymmetric discriminant coordinates with weighted observations, see <a href="#">awcoord</a> , <b>"arc"</b> asymmetric discriminant coordinates with weighted observations and robust MCD-covariance matrix, see <a href="#">awcoord</a> , <b>"nc"</b> neighborhood based coordinates, see <a href="#">ncoord</a> , <b>"wnc"</b> neighborhood based coordinates with weighted neighborhoods, see <a href="#">ncoord</a> , <b>"anc"</b> asymmetric neighborhood based coordinates, see <a href="#">ancoord</a> . Note that "bc", "vbc", "adc", "awc", "arc" and "anc" assume that there are only two classes.
clnum	integer. Number of the class which is attempted to plot homogeneously by "asymmetric methods", which are the methods assuming that there are only two classes, as indicated above. <code>clnum</code> is ignored for methods "dc" and "nc".

<code>bw</code>	logical. If TRUE, the classes are distinguished by symbols, and the default color is black/white. If FALSE, the classes are distinguished by colors, and the default symbol is <code>pch=1</code> .
<code>ignorepoints</code>	logical. If TRUE, points with label <code>ignorenum</code> in <code>clvecd</code> are ignored in the computation for <code>method</code> and are only projected afterwards onto the resulting units. If <code>pch=NULL</code> , the plot symbol for these points is "N".
<code>ignorenum</code>	one of the potential values of the components of <code>clvecd</code> . Only has effect if <code>ignorepoints=TRUE</code> , see above.
<code>pointsbyclvecd</code>	logical. If TRUE and <code>pch=NULL</code> and/or <code>col=NULL</code> , some hopefully suitable plot symbols (numbers and letters) and colors are chosen to distinguish the values of <code>clvecd</code> , starting with "1"/"black" for the cluster with the smallest <code>clvecd</code> -code (note that colors for clusters with numbers larger than minimum number +3 are drawn at random from all available colors). FALSE produces potentially less reasonable (but nonrandom) standard colors and symbols if <code>method</code> is "dc" or "nc", and will only distinguish whether <code>clvecd=clnum</code> or not for the other methods.
<code>xlab</code>	label for x-axis. If NULL, a default text is used.
<code>ylab</code>	label for y-axis. If NULL, a default text is used.
<code>pch</code>	plotting symbol, see <a href="#">par</a> . If NULL, the default is used.
<code>col</code>	plotting color, see <a href="#">par</a> . If NULL, the default is used.
<code>...</code>	additional parameters passed to <code>plot</code> or the projection methods.

### Note

For some of the asymmetric methods, the area in the plot occupied by the "homogeneous class" (see `clnum` above) may be very small, and it may make sense to run `plotcluster` a second time specifying plot parameters `xlim` and `ylim` in a suitable way. It often makes sense to magnify the plot region containing the homogeneous class in this way so that its separation from the rest can be seen more clearly.

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

- Hennig, C. (2004) Asymmetric linear dimension reduction for classification. *Journal of Computational and Graphical Statistics* 13, 930-945 .
- Hennig, C. (2005) A method for visual cluster validation. In: Weihs, C. and Gaul, W. (eds.): *Classification - The Ubiquitous Challenge*. Springer, Heidelberg 2005, 153-160.
- Seber, G. A. F. (1984). *Multivariate Observations*. New York: Wiley.
- Fukunaga (1990). *Introduction to Statistical Pattern Recognition* (2nd ed.). Boston: Academic Press.

**See Also**

[discrcoord](#), [batcoord](#), [mvdcoord](#), [adcoord](#), [awcoord](#), [ncoord](#), [ancoord](#).

[discrproj](#) is an interface to all these projection methods.

[rFace](#) for generation of the example data used below.

**Examples**

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
plotcluster(face, grface)
plotcluster(face, grface==1)
```

---

randcmatrix

*Random partition matrix*

---

**Description**

For use within `regmix`. Generates a random 0-1-matrix with `n` rows and `cln` columns so that every row contains exactly one one and every columns contains at least `p+3` ones.

**Usage**

```
randcmatrix(n, cln, p)
```

**Arguments**

<code>n</code>	positive integer. Number of rows.
<code>cln</code>	positive integer. Number of columns.
<code>p</code>	positive integer. See above.

**Value**

An `n*cln`-matrix.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[regmix](#)

**Examples**

```
set.seed(111)
randcmatrix(10, 2, 1)
```

`randconf`*Generate a sample indicator vector*

---

**Description**

Generates a logical vector of length `n` with `p` TRUEs.

**Usage**

```
randconf(n, p)
```

**Arguments**

`n` positive integer.  
`p` positive integer.

**Value**

A logical vector.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[sample](#)

**Examples**

```
randconf(10, 3)
```

---

`regmix`*Mixture Model ML for Clusterwise Linear Regression*

---

**Description**

Computes an ML-estimator for clusterwise linear regression under a regression mixture model with Normal errors. Parameters are proportions, regression coefficients and error variances, all independent of the values of the independent variable, and all may differ for different clusters. Computation is by the EM-algorithm. The number of clusters is estimated via the Bayesian Information Criterion (BIC).

**Usage**

```
regmix(indep, dep, ir=1, nclust=1:7, icrit=1.e-5, minsig=1.e-6, warnings=FALSE)
```

```
regem(indep, dep, m, cln, icrit=1.e-5, minsig=1.e-6, warnings=FALSE)
```

**Arguments**

<code>indep</code>	numerical matrix or vector. Independent variables.
<code>dep</code>	numerical vector. Dependent variable.
<code>ir</code>	positive integer. Number of iteration runs for every number of clusters.
<code>nclust</code>	vector of positive integers. Numbers of clusters.
<code>icrit</code>	positive numerical. Stopping criterion for the iterations (difference of loglikelihoods).
<code>minsig</code>	positive numerical. Minimum value for the variance parameters (likelihood is unbounded if variances are allowed to converge to 0).
<code>warnings</code>	logical. If TRUE, warnings are given during the EM iteration in case of collinear regressors, too small mixture components and error variances smaller than minimum. In the former two cases, the algorithm is terminated without a result, but an optimal solution is still computed from other algorithm runs (if there are others). In the latter case, the corresponding variance is set to the minimum.
<code>cln</code>	positive integer. (Single) number of clusters.
<code>m</code>	matrix of positive numerals. Number of columns must be <code>cln</code> . Number of rows must be number of data points. Columns must add up to 1. Initial configuration for the EM iteration in terms of a probability vector for every point which gives its degree of membership to every cluster. As generated by <a href="#">randcmatrix</a> .

**Details**

The result of the EM iteration depends on the initial configuration, which is generated randomly by [randcmatrix](#) for `regmix`. `regmix` calls `regem`. To provide the initial configuration manually, use parameter `m` of `regem` directly. Take a look at the example about how to generate `m` if you want to specify initial parameters.

The original paper DeSarbo and Cron (1988) suggests the AIC for estimating the number of clusters. The use of the BIC is advocated by Wedel and DeSarbo (1995). The BIC is defined here as  $2 * \text{loglik} - \log(n) * ((p+3) * \text{cln} - 1)$ ,  $p$  being the number of independent variables, i.e., the larger the better.

See the entry for the input parameter `warnings` for the treatment of several numerical problems.

**Value**

`regmix` returns a list containing the components `clnopt`, `loglik`, `bic`, `coef`, `var`, `eps`, `z`, `g`.

`regem` returns a list containing the components `loglik`, `coef`, `var`, `z`, `g`, `warn`.

clnopt	optimal number of clusters according to the BIC.
loglik	loglikelihood for the optimal model.
bic	vector of BIC values for all numbers of clusters in <code>nclust</code> .
coef	matrix of regression coefficients. First row: intercept parameter. Second row: parameter of first independent variable and so on. Columns corresponding to clusters.
var	vector of error variance estimators for the clusters.
eps	vector of cluster proportion estimators.
z	matrix of estimated a posteriori probabilities of the points (rows) to be generated by the clusters (columns). Compare input argument <code>m</code> .
g	integer vector of estimated cluster numbers for the points (via <code>argmax</code> over <code>z</code> ).
warn	logical. TRUE if one of the estimated clusters has too few points and/or collinear regressors.

### Author(s)

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

### References

- DeSarbo, W. S. and Cron, W. L. (1988) A maximum likelihood methodology for clusterwise linear regression, *Journal of Classification* 5, 249-282.
- Wedel, M. and DeSarbo, W. S. (1995) A mixture likelihood approach for generalized linear models, *Journal of Classification* 12, 21-56.

### See Also

[fixreg](#) for fixed point clusters for clusterwise linear regression.

[EMclust](#) for Normal mixture model fitting (non-regression).

### Examples

```
set.seed(12234)
data(tonedata)
# Note: If you do not use the installed package, replace this by
# tonedata <- read.table("(path/)tonedata.txt", header=TRUE)
attach(tonedata)
rmt1 <- regmix(stretchratio,tuned,nclust=1:2)
# nclust=1:2 makes the example fast;
# a more serious application would rather use the default.
rmt1$g
rmt1$bic
# start with initial parameter values
cln <- 3
n <- 150
initcoef <- cbind(c(2,0),c(0,1),c(0,2.5))
```

```

initvar <- c(0.001,0.0001,0.5)
initedps <- c(0.4,0.3,0.3)
# computation of m from initial parameters
m <- matrix(nrow=n, ncol=cln)
stm <- numeric(0)
for (i in 1:cln)
  for (j in 1:n){
    m[j,i] <- initedps[i]*dnorm(tuned[j],mean=initcoef[1,i]+
      initcoef[2,i]*stretchratio[j], sd=sqrt(initvar[i]))
  }
  for (j in 1:n){
    stm[j] <- sum(m[j,])
    for (i in 1:cln)
      m[j,i] <- m[j,i]/stm[j]
  }
rmt2 <- regem(stretchratio, tuned, m, cln)
rmt2bic <- 2*rmt2$loglik - log(150)*(4*cln-1)
rmt2bic

```

---

rFace

*"Face-shaped" clustered benchmark datasets*


---

## Description

Generates "face-shaped" clustered benchmark datasets.

## Usage

```
rFace(n, p = 6, nrep.top = 2, smile.coef = 0.6, dMoNo = 1.2, dNoEy = 1)
```

## Arguments

n	integer greater or equal to 10. Number of points.
p	integer greater or equal to 2. Dimension.
nrep.top	integer. Number of repetitions of the hair-top point.
smile.coef	numeric. Coefficient for quadratic term used for generation of mouth-points. Positive values=>smile.
dMoNo	number. Distance from mouth to nose.
dNoEy	number. Minimum vertical distance from mouth to eyes.

## Details

The function generates a nice benchmark example for cluster analysis. There are six "clusters" in this data, of which the first five are clearly homogeneous patterns, but with different distributional shapes and different qualities of separation. The clusters are distinguished only in the first two dimensions. The attribute `grouping` is a factor giving the cluster numbers, see below. The sixth group of points corresponds to some hairs, and is rather a collection of outliers than a cluster in

itself. This group contains `nrep.top+2` points. Of the remaining points, 20% belong to cluster 1, the chin (quadratic function plus noise). 10% belong to cluster 2, the right eye (Gaussian). 30% belong to cluster 3, the mouth (Gaussian/squared Gaussian). 20% belong to cluster 4, the nose (Gaussian/gamma), and 20% belong to cluster 5, the left eye (uniform).

The distributions of the further variables are homogeneous over all points. The third dimension is exponentially distributed, the fourth dimension is Cauchy distributed, all further distributions are Gaussian.

Please consider the source code for exact generation of the clusters.

### Value

An  $n$  times  $p$  numeric matrix with attributes

`grouping` a factor giving the cluster memberships of the points.  
`indexlist` a list of six vectors containing the indices of points belonging to the six groups.

### Author(s)

Martin Maechler <maechler@stat.math.ethz.ch> <http://stat.ethz.ch/~maechler/>

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

### Examples

```
set.seed(4634)
face <- rFace(600, dMoNo=2, dNoEy=0)
grface <- as.integer(attr(face, "grouping"))
plot(face, col = grface)
pairs(face, col = grface, main = "rFace(600, dMoNo=2, dNoEy=0)")
```

---

simmatrix

*Extracting intersections between clusters from fpc-object*

---

### Description

Extracts the information about the size of the intersections between representative Fixed Point Clusters (FPCs) of stable groups from the output of the FPC-functions `fixreg` and `fixmahal`.

### Usage

```
simmatrix(fpcobj)
```

### Arguments

`fpcobj` an object of class `rfpc` or `mfp`.

**Value**

A non-negative real-valued vector giving the number of points in the intersections of the representative FPCs of stable groups.

**Note**

The intersection between representative FPCs no.  $i$  and  $j$  is at position `sseg(i, j)`.

**Author(s)**

Christian Hennig <chrish@stats.ucl.ac.uk> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

`fixmahal`, `fixreg`, `sseg`

**Examples**

```
set.seed(190000)
data(tonedata)
# Note: If you do not use the installed package, replace this by
# tonedata <- read.table("(path/)tonedata.txt", header=TRUE)
attach(tonedata)
tonefix <- fixreg(stretchratio,tuned,mtf=1,ir=20)
simmatrix(tonefix)[sseg(2,3)]
```

---

solvecov

*Inversion of (possibly singular) symmetric matrices*

---

**Description**

Tries to invert a matrix by `solve`. If this fails because of singularity, an eigenvector decomposition is computed, and eigenvalues below  $1/cmax$  are replaced by  $1/cmax$ , i.e., `cmax` will be the corresponding eigenvalue of the inverted matrix.

**Usage**

```
solvecov(m, cmax = 1e+10)
```

**Arguments**

`m` a numeric symmetric matrix.  
`cmax` a positive value, see above.

**Value**

A list with the following components:

<code>inv</code>	the inverted matrix
<code>coll</code>	TRUE if <code>solve</code> failed because of singularity.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[solve](#), [eigen](#)

**Examples**

```
x <- c(1,0,0,1,0,1,0,0,1)
dim(x) <- c(3,3)
solvecov(x)
```

---

sseg

*Position in a similarity vector*

---

**Description**

`sseg(i, j)` gives the position of the similarity of objects `i` and `j` in the similarity vectors produced by `fixreg` and `fixmahal`. `sseg` should only be used as an auxiliary function in `fixreg` and `fixmahal`.

**Usage**

```
sseg(i, j)
```

**Arguments**

<code>i</code>	positive integer.
<code>j</code>	positive integer.

**Value**

A positive integer.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**Examples**

```
sseg(3, 4)
```

---

`tdecomp`*Root of singularity-corrected eigenvalue decomposition*

---

**Description**

Computes transposed eigenvectors of matrix `m` times diagonal of square root of eigenvalues so that eigenvalues smaller than  $1e-6$  are set to  $1e-6$ .

**Usage**

```
tdecomp(m)
```

**Arguments**

`m` a symmetric matrix of minimum format  $2*2$ .

**Details**

Thought for use in `discrcoord` only.

**Value**

a matrix.

**Note**

Thought for use within `discrcoord` only.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**Examples**

```
x <- rnorm(10)
y <- rnorm(10)
z <- cov(cbind(x, y))
tdecomp(z)
```

---

 tonedata

*Tone perception data*


---

### Description

The tone perception data stem from an experiment of Cohen (1980) and have been analyzed in de Veaux (1989). A pure fundamental tone was played to a trained musician. Electronically generated overtones were added, determined by a stretching ratio of `stretchratio`. `stretchratio=2.0` corresponds to the harmonic pattern usually heard in traditional definite pitched instruments. The musician was asked to tune an adjustable tone to the octave above the fundamental tone. `tuned` gives the ratio of the adjusted tone to the fundamental, i.e. `tuned=2.0` would be the correct tuning for all `stretchratio`-values. The data analyzed here belong to 150 trials with the same musician. In the original study, there were four further musicians.

### Usage

```
data(tonedata)
```

### Format

A data frame with 2 variables `stretchratio` and `tuned` and 150 cases.

### Source

Cohen, E. A. (1980) *Inharmonic tone perception*. Unpublished Ph.D. dissertation, Stanford University

### References

de Veaux, R. D. (1989) Mixtures of Linear Regressions, *Computational Statistics and Data Analysis* 8, 227-245.

---

 wfu

*Weight function (for Mahalabobis distances)*


---

### Description

Function of the elements of `md`, which is 1 for arguments smaller than `ca`, 0 for arguments larger than `ca2` and linear (default: continuous) in between.

Thought for use in `fixmahal`.

### Usage

```
wfu(md, ca, ca2, a1 = 1/(ca - ca2), a0 = -a1 * ca2)
```

**Arguments**

md	vector of positive numerals.
ca	positive numerical.
ca2	positive numerical.
a1	numerical. Slope.
a0	numerical. Intercept.

**Value**

A vector of numerals between 0 and 1.

**Author(s)**

Christian Hennig <[chrish@stats.ucl.ac.uk](mailto:chrish@stats.ucl.ac.uk)> <http://www.homepages.ucl.ac.uk/~ucakche/>

**See Also**

[fixmahal](#)

**Examples**

```
md <- seq(0, 10, by=0.1)
wfu(md, ca=5, ca2=8)
```

# Index

## \*Topic **arith**

can, 8  
cweight, 23  
wfu, 69

## \*Topic **array**

con.comp, 21  
solvecov, 66  
tdecomp, 68

## \*Topic **classif**

adcoord, 2  
ancoord, 3  
awcoord, 5  
batcoord, 7  
discrcoord, 26  
discrproj, 27  
mvdcoord, 53  
ncoord, 55  
plotcluster, 58

## \*Topic **cluster**

clujaccard, 9  
clusexpect, 10  
cluster.stats, 11  
clusterboot, 14  
cmahal, 19  
con.comp, 21  
dbscan, 24  
fixmahal, 29  
fixreg, 36  
fpclusters, 41  
itnumber, 42  
kmeansCBI, 43  
kmeansruns, 47  
mahalconf, 51  
minsize, 52  
pamk, 56  
randcmatrix, 60  
regmix, 61

## \*Topic **datasets**

tonedata, 69

## \*Topic **data**

rFace, 64

## \*Topic **distribution**

randconf, 61

## \*Topic **multivariate**

adcoord, 2  
ancoord, 3  
awcoord, 5  
batcoord, 7  
cluster.stats, 11  
clusterboot, 14  
cov.wml, 22  
dbscan, 24  
discrcoord, 26  
discrproj, 27  
fixmahal, 29  
kmeansCBI, 43  
kmeansruns, 47  
mahalanodisc, 49  
mahalanofix, 50  
mahalconf, 51  
mvdcoord, 53  
ncoord, 55  
pamk, 56  
plotcluster, 58

## \*Topic **regression**

fixreg, 36  
regmix, 61

## \*Topic **robust**

fixmahal, 29  
fixreg, 36

## \*Topic **univar**

clusexpect, 10  
itnumber, 42  
minsize, 52

## \*Topic **utilities**

simmatrix, 65  
sseg, 67

adcoord, 2, 28, 29, 58, 60

- ancoord, [3](#), [28](#), [29](#), [58](#), [60](#)  
 awcoord, [5](#), [23](#), [28](#), [29](#), [49](#), [58](#), [60](#)  
  
 batcoord, [7](#), [27–29](#), [33](#), [35](#), [58](#), [60](#)  
 batvarcoord (*batcoord*), [7](#)  
  
 can, [8](#), [37](#), [41](#)  
 clara, [17](#), [45–47](#)  
 claraCBI, [19](#)  
 claraCBI (*kmeansCBI*), [43](#)  
 clujaccard, [9](#)  
 clusexpect, [10](#), [39](#), [41](#), [43](#), [53](#)  
 cluster.stats, [11](#)  
 clusterboot, [13](#), [14](#), [43](#), [45–47](#)  
 cmahal, [19](#), [30](#), [31](#), [35](#)  
 cmdscale, [45](#)  
 con.comp, [21](#)  
 cov, [23](#)  
 cov.rob, [3](#), [5](#), [30](#), [35](#), [50](#), [51](#), [54](#), [55](#)  
 cov.wml, [22](#), [35](#)  
 cov.wt, [22](#), [23](#)  
 cutree, [21](#)  
 cweight, [23](#)  
  
 dbscan, [17](#), [24](#), [45–47](#)  
 dbscanCBI, [19](#)  
 dbscanCBI (*kmeansCBI*), [43](#)  
 discrcoord, [7](#), [8](#), [26](#), [28](#), [29](#), [58](#), [60](#), [68](#)  
 discrproj, [3](#), [4](#), [6](#), [27](#), [54](#), [56](#), [60](#)  
 dist, [13](#), [19](#), [47](#)  
 disthclustCBI, [19](#)  
 disthclustCBI (*kmeansCBI*), [43](#)  
 distnoisemclustCBI, [19](#)  
 distnoisemclustCBI (*kmeansCBI*), [43](#)  
 disttrimkmeansCBI, [19](#)  
 disttrimkmeansCBI (*kmeansCBI*), [43](#)  
  
 eigen, [67](#)  
 EMclust, [63](#)  
  
 fixmahal, [17](#), [19](#), [20](#), [29](#), [41](#), [42](#), [45–47](#),  
     [50–52](#), [65](#), [66](#), [70](#)  
 fixreg, [8–11](#), [35](#), [36](#), [42](#), [43](#), [52](#), [53](#), [63](#), [65](#),  
     [66](#)  
 fpclusters, [41](#)  
 fpclusters.mfpc (*fixmahal*), [29](#)  
 fpclusters.rfpc (*fixreg*), [36](#)  
 fpmi (*fixmahal*), [29](#)  
  
 hc, [45](#)  
  
 hclust, [21](#), [44](#), [46](#), [47](#)  
 hclustCBI, [19](#)  
 hclustCBI (*kmeansCBI*), [43](#)  
 hclusttreeCBI, [19](#)  
 hclusttreeCBI (*kmeansCBI*), [43](#)  
  
 isoMDS, [45](#)  
 itnumber, [37](#), [41](#), [42](#), [53](#)  
  
 kmeans, [17](#), [46–48](#)  
 kmeansCBI, [17](#), [19](#), [43](#)  
 kmeansruns, [46](#), [47](#), [47](#)  
  
 mahalanodisc, [49](#)  
 mahalanofix, [50](#)  
 mahalanofuz (*mahalanofix*), [50](#)  
 mahalCBI, [19](#)  
 mahalCBI (*kmeansCBI*), [43](#)  
 mahalconf, [31](#), [32](#), [35](#), [51](#)  
 mclustBIC, [16](#), [17](#), [45–47](#)  
 minsize, [41](#), [52](#)  
 mvdcoord, [28](#), [29](#), [53](#), [58](#), [60](#)  
  
 ncoord, [28](#), [29](#), [55](#), [58](#), [60](#)  
 NNclean, [45](#), [47](#)  
 noisemclustCBI, [19](#)  
 noisemclustCBI (*kmeansCBI*), [43](#)  
  
 pam, [17](#), [45–47](#), [56](#), [57](#)  
 pamk, [17](#), [46](#), [47](#), [56](#)  
 pamkCBI, [19](#)  
 pamkCBI (*kmeansCBI*), [43](#)  
 par, [32](#), [38](#), [59](#)  
 plot.clboot (*clusterboot*), [14](#)  
 plot.dbscan (*dbscan*), [24](#)  
 plot.mfpc (*fixmahal*), [29](#)  
 plot.rfpc (*fixreg*), [36](#)  
 plotcluster, [3](#), [4](#), [6](#), [8](#), [17](#), [27](#), [35](#), [54](#), [56](#),  
     [58](#)  
 predict.dbscan (*dbscan*), [24](#)  
 print.clboot (*clusterboot*), [14](#)  
 print.dbscan (*dbscan*), [24](#)  
 print.mfpc (*fixmahal*), [29](#)  
 print.rfpc (*fixreg*), [36](#)  
 print.summary.mfpc (*fixmahal*), [29](#)  
 print.summary.rfpc (*fixreg*), [36](#)  
  
 randcmatrix, [60](#), [62](#)  
 randconf, [61](#)  
 regem (*regmix*), [61](#)

regmix, 41, 60, 61  
rFace, 3, 4, 6, 8, 27, 29, 35, 54, 56, 60, 64  
rfpi (*fixreg*), 36

sammon, 45  
sample, 61  
silhouette, 13  
simmatrix, 65  
solve, 67  
solvecov, 33, 35, 49–52, 66  
sseg, 33–35, 40, 41, 66, 67  
summary.mclustBIC, 45  
summary.mfpc (*fixmahal*), 29  
summary.rfpc (*fixreg*), 36

tdecomp, 2, 4, 6, 26, 68  
tonedata, 69  
trimkmeans, 17, 45–47  
trimkmeansCBI, 19  
trimkmeansCBI (*kmeansCBI*), 43

var, 23

wfu, 30, 32, 35, 69