Package 'GLAD'

November 6, 2026

Version 2.77.0

Date 2020-04-15

Title Gain and Loss Analysis of DNA

Depends R (>= 2.10)

SystemRequirements gsl. Note: users should have GSL installed. Windows users: 'consult the README file available in the inst directory of the source distribution for necessary configuration instructions'.

Imports aws

Author Philippe Hupe

Maintainer Philippe Hupe <glad@curie.fr>

Description Analysis of array CGH data : detection of breakpoints in genomic profiles and assignment of a status (gain, normal or loss) to each chromosomal regions identified.

License GPL-2

URL http://bioinfo.curie.fr

biocViews Microarray, CopyNumberVariation

RoxygenNote 6.0.1

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

git_branch devel

git_last_commit d99f1e9

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-11-05

Contents

array 2
arrayCGH 3
arrayPersp 4
arrayPlot 5
as.data.frame.profileCGH 7
as.profileCGH 9

ChrNumeric 10
ColorBar 11
cytoband 12
daglad 12
glad 17
GLAD-internal 20
hclustglad 21
kernelpen 23
myPalette 24
plotProfile 25
profileCGH 27
snijders 28
veltman 29

Index **30**

`array` *Bladder cancer CGH data*

Description

Bladder cancer data from 3 arrays CGH (Comparative Genomic Hybridyzation). Arrays dimension are 4 blocs per column, 4 blocs per row, 21 columns per bloc and 22 rows by blocs.

Usage

```
data(arrayCGH)
```

Format

A data frame composed of the following elements :

Log2Rat Log 2 ratio.

Position BAC position on the genome.

CHROMOSOME Chromosome.

Col Column location on the array.

Row Row location on the array.

Source

Institut Curie, <glad@curie.fr>.

Examples

```
data(arrayCGH)
data <- array1 #array1 to array3
```

arrayCGH *Object of Class arrayCGH*

Description

Description of the object arrayCGH.

Value

The object arrayCGH is a list with at least a data.frame named arrayValues and a vector named arrayDesign. The data.frame arrayValues must contain the following fields :

Col	Vector of columns coordinates.
Row	Vector of rows coordinates.
...	Other elements can be added.

The vector arrayDesign is composed of 4 values : c(arrayCol, arrayRow, SpotCol, SpotRow). The array CGH is represented by arrayRow*arrayCol blocs and each bloc is composed of SpotRow*SpotCol spots.

N.B. : Col takes the values in 1:arrayRow*SpotRow and Row takes the values in 1:arrayCol*SpotCol

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>.

See Also

glad.

Examples

```
data(arrayCGH)

# object of class arrayCGH

array <- list(arrayValues=array2, arrayDesign=c(4,4,21,22))
class(array) <- "arrayCGH"
```

arrayPersp *Perspective image of microarray spots statistic*

Description

The function arrayPersp creates perspective images of shades of gray or colors that correspond to the values of a statistic for each spot on the array. The statistic can be the intensity log-ratio, a spot quality measure (e.g. spot size or shape), or a test statistic. This function can be used to explore whether there are any spatial effects in the data, for example, print-tip or cover-slip effects.

Usage

```
## Default S3 method:
arrayPersp(Statistic, Col, Row,
                   ArrCol, ArrRow, SpotCol, SpotRow,
                   mediancenter=FALSE,
                   col=myPalette("green","red","yellow"),
                   zlim=zlim, bar=TRUE, ...)


## S3 method for class 'arrayCGH'
arrayPersp(arrayCGH, variable,
                 mediancenter=FALSE,
      col=myPalette("green","red","yellow"),
             zlim=zlim, bar=TRUE, ...)
```

Arguments

arrayCGH	Object of class arrayCGH.
variable	Variable to be plotted
Statistic	Statistic to be plotted.
Col	Vector of columns coordinates.
Row	Vector of rows coordinates.
ArrCol	Number of columns for the blocs.
ArrRow	Number of rows for the blocs.
SpotCol	Number of column for each bloc.
SpotRow	Number of rows for each bloc.
mediancenter	If mediancenter=TRUE, values of Statistic are median-centered.
col	List of colors such as that generated by Palettes. In addition to these color palettes functions, a new function myPalette was defined to generate color palettes from user supplied low, middle, and high color values.
zlim	Numerical vector of length 2 giving the extreme values of z to associate with colors low and high of myPalette. By default zlim is the range of z. Any values of z outside the interval zlim will be truncated to the relevant limit.
bar	If bar=TRUE, a calibration color bar is shown to the right of the image.
...	Graphical parameters can be given as arguments to function persp.

N.B. : Col takes the values in 1:arrayRow*SpotRow and Row takes the values in 1:arrayCol*SpotCol

Value

An image is created on the current graphics device.

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>.

See Also

persp, arrayPlot, myPalette.

Examples

```
## Not run:
data(arrayCGH)


# object of class arrayCGH

array <- list(arrayValues=array2, arrayDesign=c(4,4,21,22))
class(array) <- "arrayCGH"

arrayPersp(array,"Log2Rat", main="Perspective image of array CGH",
           box=FALSE, theta=110, phi=40)


## End(Not run)
```

---

arrayPlot *Spatial image of microarray spots statistic*

Description

The function arrayPlot creates spatial images of shades of gray or colors that correspond to the values of a statistic for each spot on the array. The statistic can be the intensity log-ratio, a spot quality measure (e.g. spot size or shape), or a test statistic. This function can be used to explore whether there are any spatial effects in the data, for example, print-tip or cover-slip effects.

Usage

```
## Default S3 method:
arrayPlot(Statistic, Col, Row,
                  ArrCol, ArrRow, SpotCol, SpotRow,
                  mediancenter=FALSE,
                  col=myPalette("green", "red", "yellow"),
                  contour=FALSE, nlevels=5,
```

```
                    zlim=NULL, bar=c("none", "horizontal", "vertical"),
                    layout=TRUE, ...)

## S3 method for class 'arrayCGH'
arrayPlot(arrayCGH, variable,
                     mediancenter=FALSE,
                     col=myPalette("green", "red", "yellow"),
                     contour=FALSE, nlevels=5,
                     zlim=NULL, bar=c("none", "horizontal", "vertical"),
                     layout=TRUE, ...)
```

Arguments

arrayCGH	Object of class arrayCGH.
variable	Variable to be plotted
Statistic	Statistic to be plotted.
Col	Vector of columns coordinates.
Row	Vector of rows coordinates.
ArrCol	Number of columns for the blocs.
ArrRow	Number of rows for the blocs.
SpotCol	Number of column for each bloc.
SpotRow	Number of rows for each bloc.
mediancenter	If mediancenter=TRUE, values of Statistic are median-centered.
col	List of colors such as that generated by Palettes. In addition to these color palettes functions, a new function myPalette was defined to generate color palettes from user supplied low, middle, and high color values.
contour	If contour=TRUE, contour are plotted, otherwise they are not shown.
nlevels	Numbers of levels added by contour if contour=TRUE.
zlim	Numerical vector of length 2 giving the extreme values of z to associate with colors low and high of myPalette. By default zlim is the range of z. Any values of z outside the interval zlim will be truncated to the relevant limit.
bar	If bar=='horizontal' (resp. 'vertical'), an horizontal (resp. vertical) calibration color bar is shown to the right of the image.
layout	If layout==TRUE plot layout is automatically set when a color bar is asked for
...	Graphical parameters can be given as arguments to function image.

N.B. : Col takes the values in 1:arrayRow*SpotRow and Row takes the values in 1:arrayCol*SpotCol

Details

This function is very similar to the maImage written by Sandrine Dudoit (available in marrayPlots package) with added options zlim, mediancenter and layout.

Value

An image is created on the current graphics device.

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>.

See Also

image, contour, arrayPersp, myPalette.

Examples

```
data(arrayCGH)

pdf(file="arrayCGH.pdf",height=21/cm(1),width=29.7/cm(1))
arrayPlot(array2$Log2Rat, array2$Col, array2$Row, 4,4,21,22, main="Spatial Image of array CGH")
dev.off()


# object of class arrayCGH

array <- list(arrayValues=array2, arrayDesign=c(4,4,21,22))
class(array) <- "arrayCGH"

arrayPlot(array,"Log2Rat", main="Spatial Image of array CGH")
```

---

as.data.frame.profileCGH

profileCGH consercion

Description

Convert a profileCGH object into a data.frame.

Usage

```
## S3 method for class 'profileCGH'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	The object to converted into data.frame.
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If 'TRUE', setting row names and converting column names (to syntactic names) is optional.
...	...

Details

The attributes `profileValues` and `profileValuesNA` are binded into a data.frame.

Value

A data.frame object

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>

See Also

`as.profileCGH`

Examples

```
data(snijders)

### Creation of "profileCGH" object
profileCGH <- as.profileCGH(gm13330)



############################################################
###
###  glad function as described in HupÃ© et al. (2004)
###
############################################################


res <- glad(profileCGH, mediancenter=FALSE,
                smoothfunc="lawsglad", bandwidth=10, round=2,
                model="Gaussian", lkern="Exponential", qlambda=0.999,
                base=FALSE,
                lambdabreak=8, lambdacluster=8, lambdaclusterGen=40,
                type="tricubic", param=c(d=6),
                alpha=0.001, msize=5,
                method="centroid", nmax=8,
                verbose=FALSE)


res <- as.data.frame(res)
```

`as.profileCGH` *Create an object of class profileCGH*

Description

Create an object of class profileCGH.

Usage

```
as.profileCGH(object,...)
## S3 method for class 'data.frame'
as.profileCGH(object, infaction=c("value","empty"),
value=20, keepSmoothing=FALSE, ...)
```

Arguments

`object`	A data.frame to be convert into profileCGH.
`infaction`	If "value" then the LogRatio with infinite values (-Inf, Inf) are replace by + or - `value` according to the sign. If "empty" then NAs are put instead.
`value`	replace Inf by `value` if `infaction` is "value".
`keepSmoothing`	if TRUE the smoothing value in object is kept
`...`	...

Details

The data.frame to be convert must at least contain the following fields: LogRatio, PosOrder, and Chromosome. If the field Chromosome is of mode character, it is automatically converted into a numeric vector (see `ChrNumeric`); a field ChromosomeChar contains the character labels. The data.frame to be converted into a profileCGH objet is split into two data.frame: profileValuesNA contains the rows for which there is at least a missing value for either LogRatio, PosOrder or Chromosome; profileValues contains the remaining rows.

Value

A list with the following attributes

`profileValues`	A data.frame
`profileValuesNA`	A data.frame

Note

People interested in tools dealing with array CGH analysis can visit our web-page `http://bioinfo.curie.fr`.

Author(s)

Philippe HupÃ©, `<glad@curie.fr>`

See Also

`as.data.frame.profileCGH`

Examples

```
data(snijders)

### Creation of "profileCGH" object
profileCGH <- as.profileCGH(gm13330)

attributes(profileCGH)
```

ChrNumeric *Convert chromosome into numeric values*

Description

Convert chromosome into numeric values.

Usage

```
ChrNumeric(Chromosome)
```

Arguments

Chromosome	A vector with chromosome labels.

Details

For sexual chromosome, labels must contains X or Y which are coded by 23 and 24 respectively.

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>

Examples

```
Chromosome <- c("1","X","Y","chr X", "ChrX", "chrX", "Chr   Y")
ChrNumeric(Chromosome)
```

ColorBar *Calibration bar for color images*

Description

This function produces a color image (color bar) which can be used for the legend to another color image obtained from the functions image or arrayPlot.

Usage

```
ColorBar(x, horizontal=TRUE, col=heat.colors(50), scale=1:length(x), k=10, ...)
```

Arguments

x	If "numeric", a vector containing the "z" values in the color image, i.e., the values which are represented in the color image. Otherwise, a "character" vector representing colors.
horizontal	If TRUE, the values of x are represented as vertical color strips in the image, else, the values are represented as horizontal color strips.
col	Vector of colors such as that generated by rainbow, heat.colors, topo.colors, terrain.colors, or similar functions. In addition to these color palette functions, a new function myPalette was defined to generate color palettes from user supplied low, middle, and high color values.
scale	A "numeric" vector specifying the "z" values in the color image. This is used when the argument x is a "character" vector representing color information.
k	Object of class "numeric", for the number of labels displayed on the bar.
...	Optional graphical parameters, see par.

Author(s)

Sandrine Dudoit, Yee Hwa (Jean) Yang.

See Also

image, arrayPlot myPalette.

Examples

```
par(mfrow=c(3,1))
Rcol <- myPalette(low="white", high="red", k=10)
Gcol <- myPalette(low="white", high="green", k=50)
RGcol <- myPalette(low="green", high="red", k=100)
ColorBar(Rcol)
ColorBar(Gcol, scale=c(-5,5))
ColorBar(1:50, col=RGcol)

par(mfrow=c(1,3))
x<-seq(-1, 1, by=0.01)
ColorBar(x, col=Gcol, horizontal=FALSE, k=11)
ColorBar(x, col=Gcol, horizontal=FALSE, k=21)
ColorBar(x, col=Gcol, horizontal=FALSE, k=51)
```

cytoband *Cytogenetic banding*

Description

Cytogenetic banding

Usage

```
data(cytoband)
```

Examples

```
data(cytoband)
cytoband
```

daglad *Analysis of array CGH data*

Description

This function allows the detection of breakpoints in genomic profiles obtained by array CGH technology and affects a status (gain, normal or lost) to each clone.

Usage

```
## S3 method for class 'profileCGH'
daglad(profileCGH, mediancenter=FALSE,
 normalrefcenter=FALSE, genomestep=FALSE,
 OnlySmoothing = FALSE, OnlyOptimCall = FALSE,
 smoothfunc="lawsglad", lkern="Exponential",
 model="Gaussian", qlambda=0.999, bandwidth=10,
 sigma=NULL, base=FALSE, round=2,
 lambdabreak=8, lambdaclusterGen=40, param=c(d=6),
 alpha=0.001, msize=2, method="centroid", nmin=1, nmax=8, region.size=2,
 amplicon=1, deletion=-5, deltaN=0.10,  forceGL=c(-0.15,0.15),
 nbsigma=3, MinBkpWeight=0.35, DelBkpInAmp=TRUE, DelBkpInDel=TRUE,
 CheckBkpPos=TRUE, assignGNLOut=TRUE,
 breaksFdrQ = 0.0001, haarStartLevel = 1,
 haarEndLevel = 5, weights.name = NULL,
 verbose=FALSE, ...)
```

Arguments

profileCGH	Object of class profileCGH
mediancenter	If TRUE, LogRatio are center on their median.

genomestep — If TRUE, a smoothing step over the whole genome is performed and a "clustering throughout the genome" allows to identify a cluster corresponding to the Normal DNA level. The threshold used in the daglad function (deltaN, forceGL, amplicon, deletion) and then compared to the median of this cluster.

normalrefcenter — If TRUE, the LogRatio are centered through the median of the cluster identified during the genomestep.

OnlySmoothing — If TRUE, only segmentation is performed without optimization of breakpoints and calling.

OnlyOptimCall — If TRUE, the user can provide data which have been already segmented. In this case, profileCGH\$profileValues must contain a field with the name "Smoothing". The daglad function skip the smoothing step but bith the optimization of breakpoints and calling are performed.

smoothfunc — Type of algorithm used to smooth LogRatio by a piecewise constant function. Choose either lawsglad, haarseg, aws or laws (aws package).

lkern — lkern determines the location kernel to be used (see laws in aws package for details).

model — model determines the distribution type of LogRatio (see laws in aws package for details).

qlambda — qlambda determines the scale parameter qlambda for the stochastic penalty (see laws in aws package for details).

base — If TRUE, the position of clone is the physical position onto the chromosome, otherwise the rank position is used.

sigma — Value to be passed to either argument sigma2 of aws (see aws package) function or shape of laws (see aws package). If NULL, sigma is calculated from the data.

bandwidth — Set the maximal bandwidth hmax in the aws or laws functions in aws package. For example, if bandwidth=10 then the hmax value is set to $10*X_N$ where X_N is the position of the last clone.

round — The smoothing results of either aws or laws functions (in aws package) are rounded or not depending on the round argument. The round value is passed to the argument digits of the round function.

lambdabreak — Penalty term (λ') used during the "Optimization of the number of breakpoints" step.

lambdaclusterGen — Penalty term ($\lambda*$) used during the "clustering throughout the genome" step.

param — Parameter of kernel used in the penalty term.

alpha — Risk alpha used for the "Outlier detection" step.

msize — The outliers MAD are calculated on regions with a cardinality greater or equal to msize.

method — The agglomeration method to be used during the "clustering throughout the genome" steps.

nmin — Minimum number of clusters (N*max) allowed during the "clustering throughout the genome" clustering step.

nmax — Maximum number of clusters (N*max) allowed during the "clustering throughout the genome" clustering step.

region.size — The breakpoints which define regions with a number of probes lower or equal to this value are discared.

amplicon	Level (and outliers) with a smoothing value (log-ratio value) greater than this threshold are consider as amplicon. Note that first, the data are centered on the normal reference value computed during the "clustering throughout the genome" step.
deletion	Level (and outliers) with a smoothing value (log-ratio value) lower than this threshold are consider as deletion. Note that first, the data are centered on the normal reference value computed during the "clustering throughout the genome" step.
deltaN	Region with smoothing values in between the interval [-deltaN,+deltaN] are supposed to be normal.
forceGL	Level with smoothing value greater (lower) than rangeGL[1] (rangeGL[2]) are considered as gain (lost). Note that first, the data are centered on the normal reference value computed during the "clustering throughout the genome" step.
nbsigma	For each breakpoints, a weight is calculated which is a function of absolute value of the Gap between the smoothing values of the two consecutive regions. Weight = 1- kernelpen(abs(Gap),param=c(d=nbsigma*Sigma)) where Sigma is the standard deviation of the LogRatio.
MinBkpWeight	Breakpoints which GNLchange==0 and Weight less than MinBkpWeight are discarded.
DelBkpInAmp	If TRUE, the breakpoints identified inside amplicon regions are deleted. For amplicon, the log-ratio values are highly variable which lead to identification of false positive breakpoints.
DelBkpInDel	If TRUE, the breakpoints identified inside deletion regions are deleted. For deletion, the log-ratio values are highly variable which lead to identification of false positive breakpoints.
CheckBkpPos	If TRUE, the accuracy position of each breakpoints is checked.
assignGNLOut	If FALSE the status (gain/normal/loss) is not assigned for outliers.
breaksFdrQ	breaksFdrQ for HaarSeg algorithm.
haarStartLevel	haarStartLevel for HaarSeg algorithm.
haarEndLevel	haarEndLevel for HaarSeg algorithm.
weights.name	The name of the fields which contains the weights used for the haarseg algorithm. By default, the value is set to NULL meaning that all the observations have the same weights. If provided, the field must contain positive values.
verbose	If TRUE some information are printed.
...	...

Details

The function daglad implements a slightly modified version of the methodology described in the article : Analysis of array CGH data: from signal ratio to gain and loss of DNA regions (HupÃ© et al., Bioinformatics, 2004). For smoothing, it is possible to use either the AWS algorithm (Polzehl and Spokoiny, 2002) or the HaarSeg algorithm (Ben-Yaacov and Eldar, Bioinformatics, 2008). The daglad function allows to choose some threshold to help the algorithm to identify the status of the genomic regions. The threshodls are given in the following parameters:

- deltaN
- forceGL
- deletion
- amplicon

Value

An object of class "profileCGH" with the following attributes:

`profileValues` is a data.frame with the following information:

- **Smoothing**The smoothing values correspond to the median of each Level
- **Breakpoints**The last position of a region with identical amount of DNA is flagged by 1 otherwise it is 0. Note that during the "Optimization of the number of breakpoints" step, removed breakpoints are flagged by -1.
- **Level**Each position with equal smoothing value are labelled the same way with an integer value starting from one. The label is incremented by one when a new level occurs or when moving to the next chromosome.
- **OutliersAws**Each AWS outliers are flagged -1 (if it is in the $\alpha/2$ lower tail of the distribution) or 1 (if it is in the $\alpha/2$ upper tail of the distribution) otherwise it is 0.
- **OutliersMad**Each MAD outliers are flagged -1 (if it is in the $\alpha/2$ lower tail of the distribution) or 1 (if it is in the $\alpha/2$ upper tail of the distribution) otherwise it is 0.
- **OutliersTot**OutliersAws + OutliersMad.
- **NormalRef**Clusters which have been used to set the normal reference during the "clustering throughout the genome" step are code by 0. Note that if `genomestep=FALSE`, all the value are set to 0.
- **ZoneGNL**Status of each clone: Gain is coded by 1, Loss by -1, Amplicon by 2, deletion by -10 and Normal by 0.

`BkpInfo` is a data.frame sum up the information for each breakpoint:

- **Chromosome**Chromosome name.
- **Smoothing**Smoothing value for the breakpoint.
- **Gap**absolute value of the gap between the smoothing values of the two consecutive regions.
- **Sigma**The estimation of the standard-deviation of the chromosome.
- **Weight**1 - `kernelpen`(Gap, type, param=c(d=nbsigma*Sigma))
- **ZoneGNL**Status of the level where is the breakpoint.
- **GNLchange**Takes the value 1 if the ZoneGNL of the two consecutive regions are different.
- **LogRatio**Test over Reference log-ratio.

`NormalRef` If `genomestep=TRUE` and `normalrefcenter=FALSE`, then NormalRef is the median of the cluster which has been used to set the normal reference during the "clustering throughout the genome" step. Otherwise NormalRef is 0.

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>.

References

HupÃ© et al. (Bioinformatics, 2004): Analysis of array CGH data: from signal ratio to gain and loss of DNA regions.

Polzehl and Spokoiny (WIAS-Preprint 787, 2002): Local likelihood modelling by adaptive weights smoothing.

Ben-Yaacov and Eldar (Bioinformatics, 2008): A fast and flexible method for the segmentation of aCGH data.

See Also

glad.

Examples

```
data(snijders)
gm13330$Clone <- gm13330$BAC
profileCGH <- as.profileCGH(gm13330)


############################################################
###
###  daglad function
###
############################################################


res <- daglad(profileCGH, mediancenter=FALSE, normalrefcenter=FALSE, genomestep=FALSE,
              smoothfunc="lawsglad", lkern="Exponential", model="Gaussian",
              qlambda=0.999,  bandwidth=10, base=FALSE, round=1.5,
              lambdabreak=8, lambdaclusterGen=40, param=c(d=6), alpha=0.001, msize=2,
              method="centroid", nmin=1, nmax=8,
              amplicon=1, deletion=-5, deltaN=0.10,  forceGL=c(-0.15,0.15), nbsigma=3,
              MinBkpWeight=0.35, CheckBkpPos=TRUE)


### data for cytoband
data(cytoband)

### Genomic profile on the whole genome
plotProfile(res, unit=3, Bkp=TRUE, labels=FALSE, Smoothing="Smoothing",
main="Breakpoints detection: DAGLAD analysis", cytoband = cytoband)



###Genomic profile for chromosome 1
plotProfile(res, unit=3, Bkp=TRUE, labels=TRUE, Chromosome=1,
Smoothing="Smoothing", main="Chromosome 1: DAGLAD analysis", cytoband = cytoband)


### The standard-deviation of LogRatio are:
res$SigmaC


### The list of breakpoints is:
res$BkpInfo
```

glad *Analysis of array CGH data*

Description

This function allows the detection of breakpoints in genomic profiles obtained by array CGH technology and affects a status (gain, normal or lost) to each clone.

Usage

```
## S3 method for class 'profileCGH'
glad(profileCGH, mediancenter=FALSE,
                 smoothfunc="lawsglad", bandwidth=10, round=1.5,
                 model="Gaussian", lkern="Exponential", qlambda=0.999,
                 base=FALSE, sigma,
                 lambdabreak=8, lambdacluster=8, lambdaclusterGen=40,
                 type="tricubic", param=c(d=6),
                 alpha=0.001, msize=5,
                 method="centroid", nmax=8, assignGNLOut=TRUE,
                 breaksFdrQ = 0.0001, haarStartLevel = 1, haarEndLevel = 5,
                 verbose=FALSE, ...)
```

Arguments

profileCGH	Object of class profileCGH
mediancenter	If TRUE, LogRatio are centered on their median.
smoothfunc	Type of algorithm used to smooth LogRatio by a piecewise constant function. Choose either lawsglad, haarseg, aws or laws in aws package.
bandwidth	Set the maximal bandwidth hmax in the aws or laws functions in aws package. For example, if bandwidth=10 then the hmax value is set to $10*X_N$ where X_N is the position of the last clone.
round	The smoothing results are rounded or not depending on the round argument. The round value is passed to the argument digits of the round function.
model	Determines the distribution type of the LogRatio. Keep always the model as "Gaussian" (see laws in aws package).
lkern	Determines the location kernel to be used (see aws or laws in aws package).
qlambda	Determines the scale parameter for the stochastic penalty (see aws or laws in aws package)
base	If TRUE, the position of clone is the physical position on the chromosome, otherwise the rank position is used.
sigma	Value to be passed to either argument sigma2 ofaws function or shape of laws (see aws package). If NULL, sigma is calculated from the data.
lambdabreak	Penalty term (λ') used during the **Optimization of the number of breakpoints** step.

lambdacluster	Penalty term ($\lambda*$) used during the **MSHR clustering by chromosome** step.
lambdaclusterGen	Penalty term ($\lambda*$) used during the **HCSR clustering throughout the genome** step.
type	Type of kernel function used in the penalty term during the **Optimization of the number of breakpoints** step, the **MSHR clustering by chromosome** step and the **HCSR clustering throughout the genome** step.
param	Parameter of kernel used in the penalty term.
alpha	Risk alpha used for the **Outlier detection** step.
msize	The outliers MAD are calculated on regions with a cardinality greater or equal to msize.
method	The agglomeration method to be used during the **MSHR clustering by chromosome** and the **HCSR clustering throughout the genome** clustering steps.
nmax	Maximum number of clusters (N*max) allowed during the the **MSHR clustering by chromosome** and the **HCSR clustering throughout the genome** clustering steps.
assignGNLOut	If FALSE the status (gain/normal/loss) is not assigned for outliers.
breaksFdrQ	breaksFdrQ for HaarSeg algorithm.
haarStartLevel	haarStartLevel for HaarSeg algorithm.
haarEndLevel	for HaarSeg algorithm.
verbose	If TRUE some information are printed
...	...

Details

The function glad implements the methodology which is described in the article: Analysis of array CGH data: from signal ratio to gain and loss of DNA regions (HupÃ© et al., Bioinformatics, 2004).

The principles of the GLAD algorithm: First, the detection of breakpoints is based on the estimation of a piecewise constant function with the Adaptive Weights Smoothing (AWS) procedure (Polzehl and Spokoiny, 2002). Alternatively, it is possible to use the HaarSeg algorithm (Ben-Yaacov and Eldar, Bioinformatics, 2008). Then, a procedure based on penalyzed maximum likelihood optimizes the number of breakpoints and removes the undesirable breakpoints. Finally, based on the regions previously identified, a two-step unsupervised classification (**MSHR clustering by chromosome** and the **HCSR clustering throughout the genome**) with model selection criteria allows a status to be assigned for each region (gain, loss or normal).

Main parameters to be tuned:

qlambda	if you want the smoothing to fit some very local effect, choose a smaller qlambda.
bandwidth	choose a bandwidth not to small otherwise you will have a lot of little discontinuities.
lambdabreak	The higher the parameter is, the higher the number of undesirable breakpoints is.
lambdacluster	The higher the parameter is, the higher is the number of the regions within a chromosome which be
lambdaclusterGen	More the parameter is high more the regions over the whole genome are supposed to belong to the

Value

An object of class "profileCGH" with the following attributes:

profileValues: a data.frame with the following added information:

- **Smoothing**The smoothing values correspond to the median of each **MSHR (i.e.** `Region`**).**
- **Breakpoints**The last position of a region with identical amount of DNA is flagged by 1 otherwise it is 0. Note that during the "Optimization of the number of breakpoints" step, removed breakpoints are flagged by -1.
- **Region**Each position between two breakpoints are labelled the same way with an integer value starting from one. The label is incremented by one when a new breakpoint is found or when moving to the next chromosome. The variable `region` is what we call MSHR.
- **Level**Each position with equal smoothing value is labelled the same way with an integer value starting from one. The label is incremented by one when a new level is found or when moving to the next chromosome.
- **OutliersAws**Each AWS outliers are flagged -1 or 1 otherwise it is 0.
- **OutliersMad**Each MAD outliers are flagged -1 (if it is in the $\alpha/2$ lower tail of the distribution) or 1 (if it is in the $\alpha/2$ upper tail of the distribution) otherwise it is 0.
- **OutliersTot**OutliersAws + OutliersMad.
- **ZoneChr**Clusters identified after **MSHR (i.e.** `Region`**) clustering by chromosome**.
- **ZoneGen**Clusters identified after **HCSR clustering throughout the genome**.
- **ZoneGNL**Status of each clone : Gain is coded by 1, Loss by -1 and Normal by 0.

`BkpInfo:` the data.frame attribute `BkpInfo` which gives the list of breakpoints:

- **PosOrder**The rank position of each clone on the genome.
- **PosBase**The base position of each clone on the genome.
- **Chromosome**Chromosome name.

`SigmaC:` the data.frame attribute `SigmaC` gives the estimation of the LogRatio standard-deviation for each chromosome:

- **Chromosome**Chromosome name.
- **Value**The estimation is based on the Inter Quartile Range.

Note

People interested in tools dealing with array CGH analysis can visit our web-page `http://bioinfo.curie.fr`.

Author(s)

Philippe HupÃ©, `<glad@curie.fr>`.

References

- HupÃ© et al. (Bioinformatics, 2004) Analysis of array CGH data: from signal ratio to gain and loss of DNA regions.
- Polzehl and Spokoiny (WIAS-Preprint 787, 2002)Local likelihood modelling by adaptive weights smoothing.
- Ben-Yaacov and Eldar (Bioinformatics, 2008)A fast and flexible method for the segmentation of aCGH data.

See Also

profileCGH, as.profileCGH, plotProfile.

Examples

```
data(snijders)

### Creation of "profileCGH" object
gm13330$Clone <- gm13330$BAC
profileCGH <- as.profileCGH(gm13330)



############################################################
###
###  glad function as described in HupÃ© et al. (2004)
###
############################################################


res <- glad(profileCGH, mediancenter=FALSE,
                smoothfunc="lawsglad", bandwidth=10, round=1.5,
                model="Gaussian", lkern="Exponential", qlambda=0.999,
                base=FALSE,
                lambdabreak=8, lambdacluster=8, lambdaclusterGen=40,
                type="tricubic", param=c(d=6),
                alpha=0.001, msize=5,
                method="centroid", nmax=8,
                verbose=FALSE)

### cytoband data to plot chromosomes
data(cytoband)

### Genomic profile on the whole genome
plotProfile(res, unit=3, Bkp=TRUE, labels=FALSE, Smoothing="Smoothing",
main="Breakpoints detection: GLAD analysis", cytoband = cytoband)

###Genomic profile for chromosome 1
plotProfile(res, unit=3, Bkp=TRUE, labels=TRUE, Chromosome=1,
Smoothing="Smoothing", main="Chromosome 1: GLAD analysis", cytoband = cytoband)

### The standard-deviation of LogRatio are:
res$SigmaC

### The list of breakpoints is:
res$BkpInfo
```

GLAD-internal *GLAD-internal*

Description

Internal functions

Usage

" "

Value

" "

Author(s)

Philippe HupÃ©, glad@curie.fr.

See Also

" "

`hclustglad` *Hierarchical Clustering*

Description

Hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it.

Usage

```
hclustglad(d, method = "complete", members=NULL)
```

Arguments

`d`	a dissimilarity structure as produced by `dist`.
`method`	the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of `"ward"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"` or `"centroid"`.
`members`	`NULL` or a vector with length size of `d`.

Details

This function performs a hierarchical cluster analysis using a set of dissimilarities for the n objects being clustered. Initially, each object is assigned to its own cluster and then the algorithm proceeds iteratively, at each stage joining the two most similar clusters, continuing until there is just a single cluster. At each stage distances between clusters are recomputed by the Lance–Williams dissimilarity update formula according to the particular clustering method being used.

A number of different clustering methods are provided. *Ward's* minimum variance method aims at finding compact, spherical clusters. The *complete linkage* method finds similar clusters. The *single linkage* method (which is closely related to the minimal spanning tree) adopts a 'friends of friends' clustering strategy. The other methods can be regarded as aiming for clusters with characteristics somewhere between the single and complete link methods.

If `members!=NULL`, then `d` is taken to be a dissimilarity matrix between clusters instead of dissimilarities between singletons and `members` gives the number of observations per cluster. This way the hierarchical cluster algorithm can be "started in the middle of the dendrogram", e.g., in order to reconstruct the part of the tree above a cut (see examples). Dissimilarities between clusters can be

efficiently computed (i.e., without hclustglad itself) only for a limited number of distance/linkage combinations, the simplest one being squared Euclidean distance and centroid linkage. In this case the dissimilarities between the clusters are the squared Euclidean distances between cluster means.

In hierarchical cluster displays, a decision is needed at each merge to specify which subtree should go on the left and which on the right. Since, for n observations there are $n-1$ merges, there are $2^{(n-1)}$ possible orderings for the leaves in a cluster tree, or dendrogram. The algorithm used in hclustglad is to order the subtree so that the tighter cluster is on the left (the last, i.e. most recent, merge of the left subtree is at a lower value than the last merge of the right subtree). Single observations are the tightest clusters possible, and merges involving two observations place them in order by their observation sequence number.

Value

An object of class **hclust** which describes the tree produced by the clustering process. The object is a list with components:

merge	an $n-1$ by 2 matrix. Row i of merge describes the merging of clusters at step i of the clustering. If an element j in the row is negative, then observation $-j$ was merged at this stage. If j is positive then the merge was with the cluster formed at the (earlier) stage j of the algorithm. Thus negative entries in merge indicate agglomerations of singletons, and positive entries indicate agglomerations of non-singletons.
height	a set of $n-1$ non-decreasing real values. The clustering *height*: that is, the value of the criterion associated with the clustering method for the particular agglomeration.
order	a vector giving the permutation of the original observations suitable for plotting, in the sense that a cluster plot using this ordering and matrix merge will not have crossings of the branches.
labels	labels for each of the objects being clustered.
call	the call which produced the result.
method	the cluster method that has been used.
dist.method	the distance that has been used to create d (only returned if the distance object has a "method" attribute).

Author(s)

The hclustglad function is based an Algorithm contributed to STATLIB by F. Murtagh.

References

Everitt, B. (1974). *Cluster Analysis*. London: Heinemann Educ. Books.

Hartigan, J. A. (1975). *Clustering Algorithms*. New York: Wiley.

Sneath, P. H. A. and R. R. Sokal (1973). *Numerical Taxonomy*. San Francisco: Freeman.

Anderberg, M. R. (1973). *Cluster Analysis for Applications*. Academic Press: New York.

Gordon, A. D. (1999). *Classification*. Second Edition. London: Chapman and Hall / CRC

Murtagh, F. (1985). "Multidimensional Clustering Algorithms", in *COMPSTAT Lectures 4*. Wuerzburg: Physica-Verlag (for algorithmic details of algorithms used).

Examples

```
data(USArrests)
hc <- hclustglad(dist(USArrests), "ave")
plot(hc)
plot(hc, hang = -1)

## Do the same with centroid clustering and squared Euclidean distance,
## cut the tree into ten clusters and reconstruct the upper part of the
## tree from the cluster centers.
hc <- hclustglad(dist(USArrests)^2, "cen")
memb <- cutree(hc, k = 10)
cent <- NULL
for(k in 1:10){
  cent <- rbind(cent, colMeans(USArrests[memb == k, , drop = FALSE]))
}
hc1 <- hclustglad(dist(cent)^2, method = "cen", members = table(memb))
opar <- par(mfrow = c(1, 2))
plot(hc,  labels = FALSE, hang = -1, main = "Original Tree")
plot(hc1, labels = FALSE, hang = -1, main = "Re-start from 10 clusters")
par(opar)
```

`kernelpen` *Kernelpen function*

Description

Kernel function used in the penalty term.

Usage

```
kernelpen(x, type="tricubic", param)
```

Arguments

`x`	Real Value.
`type`	Type of kernelpen to be used
`param`	a named vector.

Details

The only kernel available is the "tricubic" kernel which takes the values $(1-(x/d)^3)^3$. The value of d is given by `param=c(d=6)` for example.

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>

myPalette *Microarray color palette*

Description

This function returns a vector of color names corresponding to a range of colors specified in the arguments.

Usage

```
myPalette(low = "white", high = c("green", "red"), mid=NULL, k =50)
```

Arguments

low	Color for the lower end of the color palette, specified using any of the three kinds of R colors, i.e., either a color name (an element of colors), a hexadecimal string of the form "#rrggbb", or an integer i meaning palette()[i].
high	Color for the upper end of the color palette, specified using any of the three kinds of R colors, i.e., either a color name (an element of colors), a hexadecimal string of the form "#rrggbb", or an integer i meaning palette()[i].
mid	Color for the middle portion of the color palette, specified using any of the three kinds of R colors, i.e., either a color name (an element of colors), a hexadecimal string of the form "#rrggbb", or an integer i meaning palette()[i].
k	Number of colors in the palette.

Value

A "character" vector of color names. This can be used to create a user-defined color palette for subsequent graphics by palette, in a col= specification in graphics functions, or in par.

Author(s)

Sandrine Dudoit, Yee Hwa (Jean) Yang.

See Also

palette, rgb, colors, col2rgb, image, ColorBar, arrayPlot.

Examples

```
par(mfrow=c(1,4))
pal <- myPalette(low="red", high="green")
ColorBar(seq(-2,2, 0.2), col=pal, horizontal=FALSE, k=21)
pal <- myPalette(low="red", high="green", mid="yellow")
ColorBar(seq(-2,2, 0.2), col=pal, horizontal=FALSE, k=21)
pal <- myPalette()
ColorBar(seq(-2,2, 0.2), col=pal, horizontal=FALSE, k=21)
pal <- myPalette(low="purple", high="purple",mid="white")
ColorBar(seq(-2,2, 0.2), col=pal, horizontal=FALSE, k=21)
```

plotProfile *Plot genomic profile and cytogenetic banding*

Description

Plot genomic profile with breakpoints, outliers, smoothing line and cytogenetic banding.

Usage

```
## S3 method for class 'profileCGH'
plotProfile(profileCGH, variable="LogRatio", Chromosome=NULL,
                        Smoothing=NULL, GNL="ZoneGNL", Bkp=FALSE,
                        labels=TRUE, plotband=TRUE, unit=0,
                        colDAGLAD=c("black","blue","red","green","yellow"),
                        pchSymbol=c(20,13),
                        colCytoBand=c("white","darkblue"),
                        colCentro="red", text=NULL,
                        cytoband = NULL, main="", ylim=NULL, ...)
```

Arguments

profileCGH	Object of class profileCGH
variable	The variable to be plot.
Chromosome	A numeric vector with chromosome number to be plotted. Use 23 and 24 for chromosome X and Y respectively. If NULL, all the genome is plotted.
Smoothing	The variable used to plot the smoothing line. If NULL, nothing is plotted.
GNL	The variable used to plot the Gain, Normal and Loss color code.
Bkp	If TRUE, the breakpoints are represented by a vertical red dashed line.
labels	If TRUE, the labels of the cytogenetic banding are written.
plotband	If TRUE, the cytogenetic banding are plotted.
unit	Give the unit of the PosBase. For example if unit=3, PosBase are in Kb, if unit=6, PosBase are in Mb, ...
colDAGLAD	Color code to plot Deletion, Amplification, Gain, Lost and Normal status.
pchSymbol	A vector of two elements to specify the symbol tu be used for plotting point. pchSymbol[2] is the symbol for outliers.
colCytoBand	Color code for cytogenetic banding.
colCentro	Color code for centromere.
text	A list with the parameters to be passed to the function text.
cytoband	cytodand data. For human, cytoband data are avaibale using data(cytoband).
main	title of the plot.
ylim	range of the y-axis
...	...

Details

" "

Value

A plot

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>.

See Also

" "

Examples

```
### Cytogenetic banding information
data(cytoband)

###
data(snijders)

### Creation of "profileCGH" object
profileCGH <- as.profileCGH(gm13330)



############################################################
###
###  glad function as described in HupÃ© et al. (2004)
###
############################################################


res <- glad(profileCGH, mediancenter=FALSE,
                smoothfunc="lawsglad", bandwidth=10, round=2,
                model="Gaussian", lkern="Exponential", qlambda=0.999,
                base=FALSE,
                lambdabreak=8, lambdacluster=8, lambdaclusterGen=40,
                type="tricubic", param=c(d=6),
                alpha=0.001, msize=5,
                method="centroid", nmax=8,
                verbose=FALSE)

### cytoband data to plot chromosome
data(cytoband)

### Genomic profile on the whole genome
plotProfile(res, unit=3, Bkp=TRUE, labels=FALSE,
            Smoothing="Smoothing", plotband=FALSE, cytoband = cytoband)
```

```
### Genomic profile on the whole genome and cytogenetic banding
plotProfile(res, unit=3, Bkp=TRUE, labels=FALSE, Smoothing="Smoothing",
            cytoband = cytoband)


### Genomic profile for chromosome 1
text <- list(x=c(90000,200000),y=c(0.15,0.3),labels=c("NORMAL","GAIN"), cex=2)
plotProfile(res, unit=3, Bkp=TRUE, labels=TRUE, Chromosome=1,
Smoothing="Smoothing", plotband=FALSE, text=text, cytoband = cytoband)

### Genomic profile for chromosome 1 and cytogenetic banding with labels
text <- list(x=c(90000,200000),y=c(0.15,0.3),labels=c("NORMAL","GAIN"), cex=2)
plotProfile(res, unit=3, Bkp=TRUE, labels=TRUE, Chromosome=1,
Smoothing="Smoothing", text=text, main="Chromosome 1", cytoband = cytoband)
```

`profileCGH` *Objects of Class profileCGH and profileChr*

Description

Description of the objects `profileCGH` and `profileChr`. The last object corresponds to data of only one chromosome.

Details

`LogRatio`, `Chromosome` and `PosOrder` are compulsory.

Value

Objects `profileCGH` and `profileChr` are composed of a list with the first element `profileValues` which is a `data.frame` with the following columns names:

`LogRatio`	Test over Reference log-ratio.
`PosOrder`	The rank position of each clone on the genome.
`PosBase`	The base position of each clone on the genome.
`Chromosome`	Chromosome name.
`Clone`	The name of the corresponding clone.
`...`	Other elements can be added.

Note

People interested in tools dealing with array CGH analysis can visit our web-page http://bioinfo.curie.fr.

Author(s)

Philippe HupÃ©, <glad@curie.fr>.

See Also

glad, as.profileCGH.

Examples

```
data(snijders)
gm13330$Clone <- gm13330$BAC
profileCGH <- as.profileCGH(gm13330)
class(profileCGH) <- "profileCGH"

profileChr <- as.profileCGH(gm13330[which(gm13330$Chromosome==1),])
class(profileChr) <- "profileChr"
```

---

snijders *Public CGH data of Snijders*

Description

The data consist of 15 human cell strains with known karyotype (12 fibroblast cell strains, 2 chorionic villus cell strains, 1 lymploblast cell strain) from the NIGMS Human Genetics Cell Repository (http://locus.umdnj.edu/nigms). Each cell strain has been hybridized onto a CGH-array of 2276 BAC's spotted in triplicate.

Usage

```
data(snijders)
```

Source

http://www.nature.com/ng/journal/v29/n3/suppinfo/ng754_S1.html

References

A M Snijders, N Nowak, R Segraves, S Blackwood, N Brown, J Conroy, G Hamilton, A K Hindle, B Huey, K Kimura, S Law, K Myambo, J Palmer, B Ylstra, J P Yue, J W Gray, A N Jain, D Pinkel & D G Albertson , Assembly of microarrays for genome-wide measurement of DNA copy number, *Nature Genetics* 29, pp 263 - 264 (2001) Brief Communications.

Examples

```
data(snijders)
array <- gm13330
```

`veltman` *Public CGH data of Veltman*

Description

The data consist of 2 bladder cancer tumors obtained by Veltman et al (2003).

Usage

```
data(veltman)
```

Source

http://cancerres.aacrjournals.org/cgi/content/full/63/11/2872

References

Joris A. Veltman, Jane Fridlyand, Sunanda Pejavar, Adam B. Olshen, James E. Korkola, Sandy DeVries, Peter Carroll, Wen-Lin Kuo, Daniel Pinkel, Donna Albertson, Carlos Cordon-Cardo, Ajay N. Jain and Frederic M. Waldman. Array-based Comparative Genomic Hybridization for Genome-Wide Screening of DNA Copy Number in Bladder Tumors. *Cancer Research* 63, 2872-2880, 2003.

Examples

```
data(veltman)
P9
```

Index

∗ **classes**
  arrayCGH, 3
  profileCGH, 27
∗ **cluster**
  hclustglad, 21
∗ **color**
  myPalette, 24
∗ **datasets**
  array, 2
  cytoband, 12
  snijders, 28
  veltman, 29
∗ **hplot**
  arrayPersp, 4
  arrayPlot, 5
  ColorBar, 11
  plotProfile, 25
∗ **internal**
  GLAD-internal, 20
∗ **manip**
  as.data.frame.profileCGH, 7
  as.profileCGH, 9
  ChrNumeric, 10
∗ **math**
  kernelpen, 23
∗ **models**
  daglad, 12
  glad, 17
∗ **multivariate**
  hclustglad, 21

affectationGNL *(GLAD-internal)*, 20
array, 2
array1 *(array)*, 2
array2 *(array)*, 2
array3 *(array)*, 2
arrayCGH, 3, *4*, *6*
arrayPersp, 4, *7*
arrayPlot, *5*, 5, *11*, *24*
as.data.frame.profileCGH, 7, *9*
as.profileCGH, *8*, 9, *20*, *28*

BkpInfo *(GLAD-internal)*, 20

CheckData *(GLAD-internal)*, 20
chrBreakpoints *(GLAD-internal)*, 20
ChrNumeric, *9*, 10
cluster *(GLAD-internal)*, 20
clusterglad *(GLAD-internal)*, 20
ColorBar, 11, *24*
cytoband, 12

daglad, 12
DelRegionTooSmall *(GLAD-internal)*, 20
detectOutliers *(GLAD-internal)*, 20
dogenomestep *(GLAD-internal)*, 20

FDRThres *(GLAD-internal)*, 20
filterBkp *(GLAD-internal)*, 20
filterBkpStep *(GLAD-internal)*, 20
findCluster *(GLAD-internal)*, 20

glad, *3*, *16*, 17, *28*
GLAD-internal, 20
gm00143 *(snijders)*, 28
gm01524 *(snijders)*, 28
gm01535 *(snijders)*, 28
gm01750 *(snijders)*, 28
gm02948 *(snijders)*, 28
gm03134 *(snijders)*, 28
gm03563 *(snijders)*, 28
gm03576 *(snijders)*, 28
gm04435 *(snijders)*, 28
gm05296 *(snijders)*, 28
gm07081 *(snijders)*, 28
gm07408 *(snijders)*, 28
gm10315 *(snijders)*, 28
gm13031 *(snijders)*, 28
gm13330 *(snijders)*, 28

HaarSeg *(GLAD-internal)*, 20
HaarSegGLAD *(GLAD-internal)*, 20
HaarSegGLADCPP *(GLAD-internal)*, 20
hclustglad, 21

image, *11*, *24*

kernelpen, 23

lawsglad *(GLAD-internal)*, 20
loopRemove *(GLAD-internal)*, 20

MoveBkp *(GLAD-internal)*, 20
MoveBkpStep *(GLAD-internal)*, 20
myPalette, *4–7*, *11*, 24

OptimBkpFindCluster *(GLAD-internal)*, 20
OutliersGNL *(GLAD-internal)*, 20

P20 *(veltman)*, 29
P9 *(veltman)*, 29
par, *11*, *24*
plotCytoBand *(GLAD-internal)*, 20
plotProfile, *20*, 25
prepare.output.daglad *(GLAD-internal)*, 20
profileCGH, *12*, *17*, *20*, *25*, 27
profileChr *(profileCGH)*, 27

RecomputeGNL *(GLAD-internal)*, 20
removeBreakpoints *(GLAD-internal)*, 20
removeLevel *(GLAD-internal)*, 20

SegmentByPeaks *(GLAD-internal)*, 20
selectindex *(GLAD-internal)*, 20
snijders, 28

testBkpToMove *(GLAD-internal)*, 20
text, *25*

veltman, 29