Package 'aCGHplus'

August 20, 2007

Version 0.0.2-37

Date Aug 15, 2007

Title Tools for Analyzing Array Comparative Genomic Hybridization Data

Author Daniel P Gaile <dpgaile@buffalo.edu>, Jeff Miecznikowski <jcm38@buffalo.edu>, Lori A. Shepherd <las65@buffalo.edu>

Maintainer Lori A. Shepherd <las65@buffalo.edu>

Depends R(>= 2.3.1),DNAcopy,fields, rgl

Suggests limma, marray,Biobase, convert, vsn, GLAD

Description A collection of tools for analyzing aCGH data

License GPL version 2 or later

URL http://sphhp.buffalo.edu/biostat/research/software/acghplus/index.php

R topics documented:

`aCGH.ex1.inv` *INVENTORY FILE FOR EXAMPLE SET*

Description

This is an inventory file for the example dataset provided with Write.aCGH.ex1. It is needed when creating an aCGH roster object

Format

A comma delimited file with a single header line indicating column names. It contains the following mandatory columns: sample.ID, image.name, load.specs, image.soft, design, default.map, orientation, and PI. Some other columns such as sex, sample.type, and comment are also included.

Details

The inventory file should contain the following mandatory columns:sample.ID, image.name, load.specs, image.soft, design, default.map and PI. The sample.ID is the sample name, image.name is the name of the associated array image file for that sample, load.specs is the name of the spec file associated with that sample, image.soft is the name of the software used to create raw image files, design is the name of the chip design,default.map is the name of the RData map build file associated with the chip design and PI is the principle investigator. An inventory file may have any number of additional columns. In this example file, additional inventory information includes: sex, orientation, sample.type, and comment

Note

No Column Name or data may have use the '' symbol, as this is recognized as a comment in R

References

See Also

`Write.aCGH.ex1`, `create.roster`, `convert.old.inv.R`

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# to see the provided inventory file
read.table("aCGH.ex1.inv", sep=",")
```

---

aCGHgetIntensityMatrix

CONVERTS ACGH VALUES INTO INTENSITY MATRIX

Description

This function will take in an aCGH object and return an intensity matrix that can be used with the vsn functions. This function is similar to vsn's getIntentisyMatrix.

Usage

```
aCGHgetIntensityMatrix(aCGH,
                       sampleDX=NA,
                       bgsub=TRUE  )
```

Arguments

`aCGH`	an aCGH object
`sampleDX`	list of samples to use. If NA, uses all samples
`bgsub`	if a background subtraction should be performed

Details

This function will take in an aCGH object and make an intensity matrix. It is currently hardwired to used the Signal.Mean. The user may choose if a simple background subtraction of the Background.Mean should be performed.

Value

an intensity matrix

Note

after this function is performed fillInMissingIntensities gets rid of missing data. This is made with intensions of using vsn

Author(s)

Lori Shepherd

References

See Also

`fillInMissingIntensities`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

mm = aCGHgetIntensityMatrix(aCGH)
mm = fillInMissingIntensities(mm)

library(Biobase)
library(vsn)

mm = vsn(mm)

plot(exprs(mm), pch=".")
meanSdPlot(mm)
vsnPlotPar(mm, "factors")
vsnPlotPar(mm, "offsets")
```

aCGH *aCGH OBJECT*

Description

Format

see details

Details

The object structure is as follows (note: the class of the objects are in parenthesis):

aCGH (aCGH/list). When the aCGH object is first created it has five objects:

data.info (list) useful data and indicies

nsmpl (numeric) total number of samples that make up aCGH object

nBAC (numeric) total number of spots (BACs/Probes) used in mapping

fname.array.images (character) vector of full path names to the array.image files for each sample.

mapping.info.name (character) name of the build map file used to map samples

fname.maps (character) vector of full path names to the map file for each sample

fname.designs (character) vector of full path names to the design file for each sample if applicable.

mapping.info (data.frame) This information is attained through the mapping build file. The number of columns is variable depending if certain criteria is flagged. There are 14 standard columns:

spot.ID (character) names of the spots - BACs, Clones, etc.

loc.start (numeric) starting location of spot with respect to chromosome

loc.center (numeric) central location of spot with respect to chromosome

loc.stop (numeric) ending location of spot with respect to chromosome

loc.genome (numeric) genomic location of the spot

Chrom (character or numeric) chromosome location of the spot

iChrom (numeric) chromosome location of the spot - factored version of Chrom

Arm (character) arm location of the spot (p,q)

iArm (numeric) arm location of the spot - factored version of arm (as if chromosome are arm i.e 1p=1,1q=2,2p=3,2q=4...)

Broad.Band (character) broad band location of spot including arm information (i.e q26, q27)

iBroad.Band (numeric) broad.band location of spot

Fine.Band (character) fine band location of spot including arm infomration (i.e q26.33, q27.1)

iFine.Band (numeric) fine band location of spot

There are three optional columnse:

mapped.by (character) indicates by what means spots and locations were identified (i.e FISH, BEP)

map.flag (character or numeric) identified spots with flags. These flags could be for reliablity, quality, etc.

random (any) random column to store addtional information user deems important

Band.Aid (list) This information is created through the mapping build file.

Centromeres (data.frame) Centromere data may be missing. It is dependent if the user provided a centromere file when creating build map files. The four columns are:

Chrom (character or numeric) the chromosome location

iChrom (numeric) the chromosome location – factored version of Chrom

loc the location of the centromere with respect to the chromsome

loc.genome the location of the centromere with respect to the genome

Regions (list) useful indicies and labels for plotting.

Chrom (data.frame)

Arm (data.frame)

Broad.Band (data.frame)

Fine.Band (data.frame)

Each data.frame has the same columns.

Chrom (character or numeric) chromosome location

Label (character) label to be used on graphs

Lower (numeric) lower bounding limit of the region

Center (numeric) central location of the region

Upper (numeric) upper bounding limit of the region

log2.ratios (matrix) numeric nBAC x nsmpl. Contains the original (with smooth2D performed) log2 ratios.

inventory (data.frame) contains information about the samples. The minimum inventory contains: sample.ID, image.name, load.specs, image.soft, design, default.map, orientation, and User. More may be specified. see aCGH.ex1.inv The aCGH inventory may also have columns added depending on what aCGHplus package functions are run.

Other objects are created as certain functions are utilized within the package. The following are such examples:

log2.ratios.fitted (matrix) numeric nBAC x nsmpl. Contains the log2 ratios after circulary binary segmentation is performed if overwrite is false. If overwrite is true these values replace the values in log2.ratios

CBSfits (list) contains lists of DNAcopy object parts

data (list) of length nsmpl,contains CNA object for each sample.

(CNA/data.frame) CNA object created with the DNAcopy call. This data.frame has the columns: chrom, maploc, and Sample.1

output (list) contains output data from DNAcopy output. It is of length nsmpl and each entry is a data.frame.

(data.frame) ouput data.frame created by the DNA copy call. This data.frame has the columns: ID, chrom,loc.start, loc.end, num.mark, and seg.mean

call (list) contains call objects from DNAcopy. It is of length nsmpl and each entry is a call object.

(call) Object created by the DNAcopy function. It contains: "", x, alpha, nperm, and verbose.

segmat (matrix) nBAC x nsmpl, integer values. This is a collection of the segvec values indicating which segment the spot lies.

CBSflag (logical) vector of length nsmpl indicating if CBS was successfully performed on the sample

NAmat (matrix) nBAC x nsmp, logical indicating where NA values did and did not occur in the original (with smooth2D performed) log2 ratios

recenterVls (numeric) vector of length nBAC indicating a value that has already been added to log2.ratios and log2.ratios.fitted as part of a centering function. These values can be subtracted to get original log2.ratios and log2.ratios.fitted values

PCAinv (character) vector which keeps track of different PCA inventories associated with this aCGH object. These invetnories can be reloaded into the interactive principle component viewer.

SegMask (list) keeps track of segmental masking data

- bacDX (numeric) vector indicating which spots were used in analysis
- smplDX (numeric) vector indicating which samples were used in the analysis
- segs (matrix) contains four columns
- biomarks (numeric)
- biomin (numeric)
- biomax (numeric)
- biocnt (numeric)
- iab (matrix) length bacDX x length smplDX, numeric,
- NAmat (matrix) length bacDX x length smplDX, logical indicating where values were missing

Note

We recommend storing aCGH objects in the RData directory. The example aCGH given in the package examples is: RData/aCGH.RData

Source

References

See Also

`make.aCGH`

Examples

```
# To see an example of an aCGH object
# This object has undergone all preprocessing
#  and therefore will have CBS fitted data

Write.aCGH.ex2()
load("RData/aCGH.RData")
ls()

names(aCGH)

# subset through object using $
names(aCGH$data.info)
```

`aCGHReLoadtoLimma` *MAKES A RGList OBJECT BY RELOADING IMAGE ANALYSIS FILES*

Description

This function will reload the original software image analysis files to create a limma RGList object.

Usage

```
aCGHReLoadtoLimma(aCGH,
                  subdx=NA,
                  cc = "",
                  fcol = "Signal Mean",
                  gcol = NA,
                  fbcol ="Background Mean" ,
                  gbcol = NA,
                  vrb=TRUE,
                  knfile = NA,
                  ...)
```

Arguments

`aCGH`	an aCGH object
`subdx`	an index of samples to use
`cc`	column to be used as column argument in read.maimages or read.imagene calls of limma. May be left NA but then need to specify fcol, gcol, fbcol, and gbcol
`fcol`	name of column in image analysis files that stores the red channel foreground intensities
`gcol`	name of column in the image analysis files that stores the green channgel foreground intensities
`fbcol`	name of the column in the image analysis files that stores the red background intensities
`gbcol`	name of the column in teh image analysis files that stores the green background intensities
`vrb`	logical indicating if status messages should be printed
`knfile`	numeric either 1 or 2. Indicates if one file microarray experiemtns or 2 file microarray experiments should be used
`...`	additional arguments to be passed into the limma package read.maimages or read.imagene calls

Details

This function will create a RGList object by retrieve the original software image analysis file[s] for each sample and reloading through limma read.maimages or read.imagene functions.

Additional arguments may be passed into the limma read.maimage and read.imagene calls. columns for these calls should be specified by the cc argument. If cc is left blank, "", then the function will

construct the column argument through fcol, gcol, fbcol, and gbcol. If there are two image analysis files per sample, fcol is the name of the column in the raw image analysis file representing the foreground intensities to be used and fbcol is the name of the column in the raw image analysis file representing the background intensities to be used. If there is only one file per sample fcol is the red (cy5)channel foreground intensities, gcol is the green(cy3) channel foreground intensities, fbcol is the red (cy5) channel background intensities, and gbcol is the green (cy3) channel background intensities. The names given should match a column name in the image analysis flat file exactly.

knife indicates if samples with one image anlaysis file or with two image analysis files should be used during the load in. (It is our understanding that unlike our package, limma and marray can handle only one type at a time)

Value

RGList object

Note

required packages: limma

user may then use bioconductors convert package to make marray or bioconductor objects

Author(s)

Lori Shepherd

References

See Also

marrayWrapper, aCGHReLoadtoMarray

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH
Write.aCGH.ex1()
Write.aCGH.ex2()
load("RData/aCGH.RData")

limmaObj = aCGHReLoadtoLimma(aCGH,
                        subdx =c(1,3,5),
                        fcol = "Signal Mean",
                        fbcol ="Background Mean",
                        knfile = 2)
```

`aCGHroster` *aCGHroster OBJECT*

Description

An aCGHroster object is a list of indecies to a set of samples' array.image objects. It also contains the inventory for all samples.

Format

see details

Details

The object structure is as follows (note: class type is in parenthesis):

aCGHroster (aCGHroster,list)contains the following

fname.array.images (character). array with full path name to samples' array.images

inventory (data.frame). contains information about the samples. The minimum inventory contains: sample.ID, image.name, load.specs, image.soft, design, default.map, orientation, and User. More may be specified. see aCGH.ex1.inv

Load.Flag (data.frame)contains information on if a sample failed the load-in process. The data contained is: sample.ID, image.name, SampleLoadFlag, LoadedFlag, SampleMapFlag, and SampleDesignFlag.

Note

We recommend storing aCGHroster objects in the RData directory. The example aCGHroster given in the package examples is: RData/aCGHroster.RData

Source

References

See Also

`create.roster`

Examples

```
# To see an example of an aCGHroster
Write.aCGH.ex2()
load("RData/aCGHroster.RData")
ls()

names(aCGHroster)

aCGHroster
```

`add.inventory` *ADDS SUPPLEMENTAL INVENTORY FILE OR SPECIFY INCLUSION IN INVENTORY*

Description

This function adds a supplemental inventory file in addition to the required-default-inventory file, updates a current inventory, or adds a column to indicate inclusion in a specific inventory.

Usage

```
add.inventory(aCGHroster,
                        inventory.file="example1.add.inv",
                        inv.sep=",",
                        matchOnlyImageName=FALSE,
                        matchOnlySampleID = FALSE,
                        overwrite=FALSE,
                        colIncl=TRUE,
                        colName=NA,
                        saveNew=FALSE,
                        saveName=NA,
                        vrb=TRUE
                        )
```

Arguments

`aCGHroster`	aCGHroster object
`inventory.file`	Name of the inventory file to be associated with `aCGHroster`.
`inv.sep`	The separation character for the inventory file. Will be passed in as sep for a read.table call
`matchOnlyImageName`	logical indicating if samples should be matched by image.name and sample.ID or just by image.name. Default is F, matches by image.name and sample.ID
`matchOnlySampleID`	logical indicating if samples should be matched by just sample.ID. Default is F, matches by image.name and sample.ID
`overwrite`	Flag to overwrite existing columns. The default will not overwrite. If overwrite is tripped, if any of the columns in the inventory are already in the main inventory, those values will be replaced.
`colIncl`	a logical indicating if a column should be included flagging the samples in the given inventory
`colName`	The name of the column to add if `colIncl` is true. The default will use the name of the inventory file
`saveNew`	logical indicating if the updated aCGHroster should be saved to a file
`saveName`	Name/path to save the aCGHroster. The default, NA, will save in the RData directory as aCGHroster.RData (in most cases this will overwrite the default file)
`vrb`	a logical flag indicating if status messages should be printed

Details

This function will add a supplemental inventory file to the aCGHroster object. There may be occassions where there is more information in the provided sample inventory file than is stored in the default master inventory list. This allows this extra information to be stored for future viewing and use. Additional data is added to the aCGHroster object's inventory. The minimal supplemental inventory file will contain the sample.ID and the image.name. The overwrite will allow for updating of the current aCGHroster. If overwrite is True, any columns that are included in the supplimental inventory that are also in the aCGHroster will be updated for the samples included. `colIncl` allows for the addition of a column indicating the sample was included in the given inventory. This allows for grouping of samples for data anaylsis. Each inventory may have a different column name, this allows for samples to be flagged for inclusion in multiple data sets. The updated aCGHroster will not be saved unless the `saveNew` is tripped to True. This will save the aCGHroster object to the given `saveName` or if left as default, NA, will save to the RData directory as aCGHroster.RData.

Value

returns updated aCGHroster object

Note

The function `convert.old.inv` is useful in creating a supplemental inventory file. It will parse a provided inventory file into the required default inventory file and a supplemental inventory file.

No column name or data may use the symbot '', as it is recognized in R as a comment

Author(s)

Lori Shepherd

References

See Also

`convert.old.inv.R`,`combineInvFiles`

Examples

```
library(aCGHplus)

# This function will use the example inventory given in Write.aCGH.ex1
# and the RData objects created in Write.aCGH.ex2

Write.aCGH.ex1()
Write.aCGH.ex2()

# Since the three samples included in the aCGH.ex1.inv were used in
# the creation of RData/aCGHroster.RData usage of this function will
# create a new column in the inventory of the aCGHroster object
# indicating the three samples share a common inventory file

load("RData/aCGHroster.RData")
```

```
newroster  = add.inventory(aCGHroster,
                        inventory.file="aCGH.ex1.inv",
                        inv.sep=",",
                        overwrite=FALSE,
                        vrb=TRUE,
                        colIncl=TRUE,
                        colName=NA,
                        saveNew=FALSE,
                        saveName=NA
                        )

# to compare inventorys:
#old inventory
aCGHroster$inventory

#new inventory
newroster$inventory
```

`addPCAinv` *saves inventory from 3D PCA Viewer in aCGH object inventory*

Description

saves inventory from the 3D PCA Viewer in aCGH inventory.

Usage

```
addPCAinv(aCGH,
       sampleInfo=NA,
       PCA.inv = NA)
```

Arguments

`aCGH`	An aCGH object
`sampleInfo`	object containing data for samples
`PCA.inv`	index for an aCGH PCA inventory file

Details

`addPCAinv` will save the inventory created for all samples used in the principle component anlaysis interative viewer. The interactive PCA viewer allows for sample scoring, commenting, removal, and flagged for removal. The information for these four vectors will be added to the existing aCGH object inventory file.

The sampleInfo object is created with the call to makePCAfile. The user must load the file created by the makePCAfile. The default is the file RData/PCAtest.RData.

Value

aCGH

Note

It is called from within `runPCAfile` and `choiceAct`

`makePCAFile` must have been run

Author(s)

Lori Shepherd

See Also

`makePCAfile`,`savePCAInv`, `runPCA`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# make PCA file
makePCAfile(aCGH)

# load PCA information
load("RData/PCAtest.RData")

addPCAinv(aCGH,sampleInfo=sampleInfo, PCA.inv=NA)
```

---

add.supInv.toInvFiles

COMBINES TWO INVENTORY FILES

Description

This function will combined two inventory files into one matching only on image.name. Samples that do not match will be ignored

Usage

```
add.supInv.toInvFiles(inventory.file1,
                inventory.file2,
                inv.sep1=",",
                inv.sep2=",",
                overwrite=FALSE,
                colIncl = TRUE,
                colName=NA,
                saveNew=TRUE,
                saveName=NA,
                out.sep=",",
```

```
                returnFlag = FALSE,
                vrb=TRUE
                )
```

Arguments

`inventory.file1`	Name of the first inventory file
`inventory.file2`	Name of the second inventory file
`inv.sep1`	The separation character for the first inventory file. Will be passed in as sep for a read.table call
`inv.sep2`	The separation character for the second inventory file. Will be passed in as sep for a read.table call
`overwrite`	Flag to overwrite existing values. The default will not overwrite and use all values from the first given inventory file. If overwrite is tripped, if any of the columns in the second inventory are already in the first inventory, the values of the second inventory will be used, replacing the first inventories values.
`colIncl`	flag to include a logical column indicating if sample was included in second inventory
`colName`	name of the column to include if colIncl
`saveNew`	logical if the new combined inventory should be saved
`saveName`	name of the path/file to save new inventory
`out.sep`	seperation character for the outfile
`returnFlag`	if the new inventory, as a data frame, should be returned
`vrb`	a logical flag indicating if status messages should be printed

Details

`combineInvFiles` takes in two inventory files to combine into one inventory file. The application will determine which samples are the same in both inventories and which samples from the second inventory will be added to the first based on image.name. Next, columns will be checked. If the same samples are in both inventories, and a column name is the same in both files, the values of the first inventroy are used, unless overwrite is tripped. If overwrite is True, the values of matching samples of matching columns, will assume the value of the second inventory.Any additional columns in the second inventory will be ignored. NAs will be added where values do not exist. Next, any additional samples in the second inventory will be added to the end of the first inventory. If columns are missing, NAs will be added. An additional column will be added for the sample.IDs in the second inventory; this will be either a sample.ID, alt.ID or alt.ID.B column. There is also an option to add a column which will indicate which samples of the first inventory were included in the second inventory.

Value

A combination inventory file, and duplicate inventory if applicable,are written to the current directory. If return flag, the new inventory as a data frame is returned

Note

Author(s)

Lori Shepherd

References

See Also

`add.inventory`, `checkInv`, `combineInvFiles`

Examples

```
library(aCGHplus)

# This function will use the example inventory given in Write.aCGH.ex1

Write.aCGH.ex1()

r = read.table("aCGH.ex1.inv", sep=",", header=TRUE)
r

# we now create a fake inventory file to combined with given inventory
inv = list()
inv$sample.ID = c("randomSample", "sampleB", "test")
inv$image.name = c("HB19Kv2-sample1","HB19Kv2-sample3", "test")
inv$Test = c("test", "test", "test")
inv = as.data.frame(inv)
write.table(inv, file="combineEx.inv", sep=",", row.names=FALSE)

newInv = add.supInv.toInvFiles(inventory.file1 = "aCGH.ex1.inv",
                inventory.file2 = "combineEx.inv",
                inv.sep1=",",
                inv.sep2=",",
                overwrite=FALSE,
                colIncl = TRUE,
                colName=NA,
                saveNew=TRUE,
                saveName="CombinedInv.inv",
                out.sep="\t",
                vrb=TRUE,
                returnFlag = TRUE
                )


# to see new inventory file
newInv
```

`addTo.GenomeOneRow` *ADD VALUES TO GENOMIC PLOT*

Description

This function will take a plotting object from InitGOR and a vector of values, and plot the values on the graph

Usage

```
# Graphs bars/lines
BarsGOR(GORplt,yvec,col="red",lwd=0.25,ycntr=0.0)

# Graphs points
pointsGOR(GORplt,yvec,spotDX=NA,highLightFlag=FALSE,
          column=aCGH$mapping.info$map.flag,
          clrs= c("blue", "green", "purple"),
          normal = 0, col="black",...)
```

Arguments

`GORplt`	a plotting object (see InitGOR)
`yvec`	a vector of values equal to the number of spots or equal to the length of subset spotDX
`lwd`	width of the lines
`ycntr`	center point, start of line
`spotDX`	subset of spotIDs
`highLightFlag`	logical if flagged spots should have different color highlighting on a genomic plot
`column`	index of length equal to the total number of BACs/Probes/spot.Ids/etc. that flags certain spots
`clrs`	vector of colors to use for highlighting different flags (levels of column)
`normal`	value in column that represents a normal or acceptable value
`col`	default color for lines or color for normal spots
`...`	additional arguments to be passed into points

Details

These functions will add points or bars to a genomic plot created using InitGOR/ReInitGOR. The plotting object carries all information needed for mapping values. The vector of values must be of equal length to the number of spots in the aCGH object or to the subset of spots using spotDX.

Sometimes different softwares save flags for the spot.IDs representing quality control or some specific variable differences. If this data is known for the spot.ID set used, the flags for each spot may be passed in and the value plotted is highlighted in different colors. This utilizes the gethighlight function when plotting points.

Value

The graph desplays the vector of values

Note

This function assumes that an empty plot is already created and open. See InitGOR and ReInitGOR

Author(s)

Lori Shepherd

References

See Also

`create.GenomeOneRow`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


  # QuickTips:
  # if export.array.images has been used you may load an aCGH object with
  #         load("RData/aCGH.RData")
  # if you have gone through the process of making an aCGHroster object
  #         load("RData/aCGHroster.RData")
  #         aCGH = make.aCGH(aCGHroster)

  plt1=InitGOR(aCGH)

  vec = aCGH$log2.ratios[,1]
  BarsGOR(plt1, vec)


  plt2=InitGOR(aCGH, spotDX=10000:12000, maxXlab=10, chrmLines=TRUE,chrmLabels=TRUE)
  pointsGOR(plt2,yvec=vec, spotDX=10000:12000,pch=".")

  # or with highlighting
  ReInitGOR(plt2)
  pointsGOR(plt2,yvec=vec, spotDX=10000:12000,pch=3, highLightFlag=TRUE)

graphics.off()
```

`addTo.multiGenomePlot` *ADD VALUES TO GENOMIC PLOT*

Description

This function will take a plotting object from pltG, and an aCGH object, and plot values on the graph

Usage

```
pointspltG(pltList,
           aCGH,
           ismpl = 1,
           yvec1 = NA,
           yvec2 = NA,
           yvec3 = NA,
           fitVls = TRUE,
           vrb=TRUE,
           fitvec1 = NA,
           fitvec2 = NA,
           fitvec3 = NA,
           vl.col = "gray63",
           fit.col = "red",
           vl.size = 0.5,
           fit.size = 0.5,
           vl.pch = 3,
           fit.pch =3,
           ...)
```

Arguments

`pltList`	plotting object created by pltG
`aCGH`	an aCGH object
`ismpl`	if a vecotr of points to graph is not indicated by specifing yvec1 below, ismpl is the number of the sample in the aCGH object whose log2.ratios/fitted log2.ratios are used.
`yvec1`	a vector of values to graph equal to the spot index of the first graph. The default, NA will use the log2.ratios of the ismple specified
`yvec2`	a vector of values to graph equal to the spot index of the second graph. The default, NA will use the log2.ratios of the ismple specified
`yvec3`	a vector of values to graph equal to the spot index of the third graph. The default, NA will use the log2.ratios of the ismple specified
`fitVls`	logical indicating if each graph has a second vector of values to plot. If this is a single value it is assumed the same for all graphs, otherwise this should be of equal length to the number of graphs
`vrb`	a logical indicating if status messages should be shown

`fitvec1`	a vector of values to graph equal to the spot index of the first graph. If fitVls is true the default will graph the fitted log2.ratios of the ismpl specified
`fitvec2`	a vector of values to graph equal to the spot index of the second graph. If fitVls is true the default will graph the fitted log2.ratios of the ismpl specified
`fitvec3`	a vector of values to graph equal to the spot index of the third graph.If fitVls is true the default will graph the fitted log2.ratios of the ismpl specified
`vl.col`	The color to plot the yvec vectors. This is a single entry for color. All graphs will have the same color
`fit.col`	The color to plot the fitvec vectors. This is a single entry for color. All graphs will have the same color
`vl.size`	The size to plot the points of yvec vectors. This is a single numeric entry. All graphs will have the same size points
`fit.size`	The size to plot the points of fitvec vectors. This is a single numeric entry. All graphs will have the same size points
`vl.pch`	The plotting character to plot the points of yvec vectors. This is a single entry for pch. All graphs will have the same pch
`fit.pch`	The plotting character to plot the points of fitvec vectors. This is a single entry for pch. All graphs will have the same pch
`...`	Additional argumrnts to be passed into points. May not be a duplicate of any of the arguments listed above(i.e col, cex, pch)

Details

This function will add points to a genomic plot created using pltG or ReInitpltG. The plotting list carries important information regarding the plot. The plotting list will contain plotting objects for each of the graphs plotted. The different yvec arguments allow for a different set of points to be plotted other than the default aCGH objects log2 ratios for the samples specified in ismpl. yvec1 is for the first graph, yvec2 the second and yvec3 for a third (if applicable). These vectors must be of equal length to the number of spot.IDs/BACs specified. –(these may be found by searching for which of the xloc are not NA in the plotDF of an object in pltList$pltObj) --The same logical is true for the fitvec. Fitvecs are plotted if fitVls for that graph

Value

points are plotted on each of the graphs of a plot created by pltG or ReInitpltG

Note

pltG must have been called first

Author(s)

Lori Shepherd

References

points function of R graphics package

See Also

`create.multiGenomePlot`,`create.GenomeOneRow`, `plotGenome`, `GenomeImage`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

  # QuickTips:
  # if export.array.images has been used you may load an aCGH object with
  #         load("RData/aCGH.RData")
  # if you have gone through the process of making an aCGHroster object
  #         load("RData/aCGHroster.RData")
  #         aCGH = make.aCGH(aCGHroster)

  # The follow will create a plot similar to plotGenome
  pltList = pltG(aCGH)

  # This will plot the first samples log2.ratios and if applicable fitted.log2.ratios
  pointspltG(pltList, aCGH)

graphics.off()
```

AgilentQCflags — *COMBINES QUALITY CONTROL FLAGS OF AGILENT INTO ONE OVERALL SPOT QUALITY FLAG*

Description

This function will combined all the spot quality control output from a sample's given analysis software into one spot quality flag. The spot is excluded if it failed in any of the criteria.

Usage

```
AgilQCFlag(aCGHroster,
           vrb=TRUE,
           inventory.Name = "load.specs",
           options="config.files/Agil.load.specs.csv")
```

Arguments

aCGHroster	an aCGHroster object
vrb	logical indicating if status messages should be printed
inventory.Name	name of the column in the inventory that will be checked for agilent samples
options	list of options indicating agilent sample in inventory.Name or the aCGHroster inventory

Details

It is recommended to use the spot quality control columns given from the software analysis files. It is neccessary then to combined all desired columns into a single spot quality flag for analysis using aCGHplus. This function uses the following quality control columns from the outfiles of feature extraction: gIsFeatNonUnifOL, gIsFeatPopnOL, gIsSaturated, gIsBGNonUnifOL,gIsBGPopnOL,IsManualFlag, gIsFound, gIsPosAndSignif, and gIsWellAboveBG. These columns are stored during the load in process if our recommended agil.load.specs.csv file is used. The aCGHroster samples are checked based on the given inventory.Name, representing a column in the aCGHroster inventory, and a set of options. If a sample has a value listed in options for the column inventory.Name the sample is determined to be included in the agilent set. The dafault uses the load.specs inentory column and searches for config.files/agil.load.specs.csv to indicate agilent samples. The application then cycles through all agilent files, loading their array.image file, combining the quality control flags, and storing one overall qualtiy control flag, spotFlag in the array.image objects list of matricies.

Value

Each agilent samples' array.image file is updated to include a spotFlag matrix. This matrix flags spots based on all quality control parameters. If a spot failed in one of the stored measurments it is excluded.

Note

This function is hard coded using structures created from our recommended Agil.load.specs.csv file see vignette for more details on our recommended load script for agilent

If the user wishes to customize agilent load-in script of not to use all the quality measurments this application provides, it is highly recommended to create a spot flag matrix based on user desired criteria.

Author(s)

Lori Shepherd

References

See Also

`create.roster`

Examples

`array.images` *ARRAY.IMAGE OBJECT*

Description

Each sample has an array.image object file that store information and indicies for that sample. It stores such information as mapping data, smoothing data, CBS data, and indicies for map and design files.

Format

see details

Details

The object structure is as follows (note: the class of the objects are in parenthesis):

array.images (list).

When the array.image object is first created it has three objects:

info (list) gives useful data and indicies for sample objects

cyinfo (list) gives names of image analysis flat file[s]

cyroot (character) gives the root name of the file[s]. The name without extension.

cyfile (character) the full path name to image analysis file. Note: cyfile is used if there is only one file associated with a sample. If there are two files associated with the sample, such as Imagene output, cyfile is replaced with two entries

cy3file (character) full path name to the cy3 image analysis flat file

cy5file (character) full path name to the cy5 image analysis flat file

image.dir (character) path/name of where array.image files containing the array.image objects

design.name (character) name of the chip design

image.soft (character) name of the image analysis software used to create image analysis flat files

load.specs (character) name of the load specs file for this sample

Lspecs (data.frame) a data.frame version of the load specs file. This is a two column data.frame.

full.design.name (character) full path name to the design file if it exists

map.name (character) name of the map file created holding the map.image object.

full.map.name (character) full path name to the map file.

mapping.info.name name of the build map file used for the chip

matrix (list)

cy3 (list) all objects under this level are matrices. The objects that are created are variable depending on what is specified in the sample's load.spec file. Only objects that are specified in the load.specs are created. The following are the objects created with the provided load.spec file in the package example:

Flag (matrix) spot quality flag. These are the flag values at every spot from the image analysis flat file

Background.Mean (matrix) the values at every spot from the image analysis flat file that contain the mean background signal.

Background.Stdev (matrix) the values at every spot from the image analysis flat file that contain the background standard deviation

Signal.Mean (matrix) the values at every spot from the image analysis flat file that contain the mean signal

Signal.Stdev (matrix) the values at every spot from the image analysis flat file that contain the standard deviation

cy5 (list) The same is true for the cy5 object as the cy3 object. Only objects specified in the load.spec file are created. Generally we would recommend any objects created for the cy3 object should be created for the cy5 and visa versa. The following are the objects created with the provided load.spec file in the package example:

Flag (matrix) spot quality flag. These are the flag values at every spot from the image analysis flat file

Background.Mean (matrix) the values at every spot from the image analysis flat file that contain the mean background signal.

Background.Stdev (matrix) the values at every spot from the image analysis flat file that contain the background standard deviation

Signal.Mean (matrix) the values at every spot from the image analysis flat file that contain the mean signal

Signal.Stdev (matrix) the values at every spot from the image analysis flat file that contain the standard deviation

Grid (matrix) contains mapping information for each spot on the chip. This mapping is the Meta.Row/Meta.Col data.

raw (list) contains usefule data concerning chip mapping and design

nCol (numeric) maximum number of chip columns over the entire chip

nRow (numeric) maximum number of chip rows over the entire chip

mstr.Row (numeric) vector indicating row location for each spot with reference to the entire chip

mstr.Col (numeric) vector indicating column location for each spot with reference to the entire chip

spot.file (character) vector giving names of the spots [Clone,BAC,Probe etc]

chip.design (numeric) vector of length four. This gives the maximum values for Meta.Row, Meta.Col, Row and Col information for the chip as opposed to masterRow and masterCol data

As furthur preprocessing is undergone additional objects are stored in the array.image object such as:

smooth2D (list) contains parameters and values of running the smooth2D functions. Note: The actual name of this object is variable depending on parameter in the function call to make the object.

info (list) stores values passed in as parameters

fname.array.images (character) full path name to the sample's array.image file

map.name (character) name of the map file holding map.image object

thetas (numeric) vector of all values of theta used. Thetas were tested as smoothing value.

cvLevel (numeric) number indicating the percentage of data points that were removed before smoothing and used as a reference check in a cross validation technique.

LossF (character) Either L1 or L2 indicating which data from Smooth.Image.CV was used. L1 represents using the sum of the absolute difference of the observed values and smoothed values. L2 represents using the sum of the difference of observed values and the smoothed values squared.

useResid (logical) were residues used

BCoption (character) Either none, raw or smoothed indicating which background correction was made.

lambdas (numeric) vector representing the different percentages of background correction to use were tested.

nlambda (numeric) The total number of lambda values tested

DesignFlag (logical) indicates if there was an attempt to correct for design

lib.loc (character) path name to a local library if necessary

cyC.label (character) path/name to what was used as the sample's control signal values

cyT.label (character) path/name to what was used as the sample's tumor signal values

cyC.BC.label (character)path/name to what was used as the sample's control background signal values

cyT.BC.label (character)path/name to what was used as the sample's tumor background signal values

exclude.rule (character) indicates a rule used for excluding points

output.label (character) what the output label objects is called

LambdaList (list)if saveAll was true in the parameters when the smoothing function was run, this object stores values of certain situations. All items are lists.

MiscMats.original.list (list)

Mxy.original.list (list)

Axy.original.list (list)

Mxy.loess.list (list)

Mxy.smooth.list (list)

MxyFit.loess.list (list)

MxyFit.smooth.list (list)

Mxy.final.list (list)

Mxy.noDesign.list (list)

Mxy.noDesign.default (matrix) smoothed values without design correction

Mxy.default (matrix) smoothed values with design correction

CBS (list) contains data created when running CBS functions. Note: The actual name of this object is variable depending on parameter in the function call to make the object.

log2.ratios (numeric) vector of original log2.ratio values, before CBS, at each spot

log2.ratios.fitted (numeric) vector of log2.ratio values after CBS at each spot

segvec (integer) vector indicating which segment the spot lies

CBSfit (DNAcopy) object from DNAcopy function

data (CNA/data.frame) CNA object created with the DNAcopy call. This data.frame has the columns: chrom, maploc, and Sample.1

output (data.frame) ouput data.frame created by the DNA copy call. This data.frame has the columns: ID, chrom,loc.start, loc.end, num.mark, and seg.mean

call (call) Object created by the DNAcopy function. It contains: "", x, alpha, nperm, and verbose.

data.info (list) some useful indicies

mapping.info.name (character)name of the build map file used for sample

fname.maps (character)full path name to the map file containing map.image object for the sample

fname.designs (character)full path name to the design file, if applicable, containing design.list for the sample

These additional objects may have variable names depending on user specifications in associated functions. The subobjects, however, should have the same structure/names.

Note

array.image object files holding the array.images object are located in the array.image subdirectory images. The example images object file given in the package is: array.images/images/HB19Kv2-sample1.RData

Source

References

See Also

`create.array.images`

Examples

```
# To see an example of a array.image file object
Write.aCGH.ex2()
load("array.images/images/HB19Kv2-sample1.RData")
ls()

names(array.images)

# subset through object using $
names(array.images$info)
```

---

buildMap *BUILD MAP OF CHROMOSOME AND BAND DATA*

Description

This function takes in a files containing chromosome and band information. It may also take in a file containing centromere location information. It creates a file containing two objects: a mappingInfo data.frame and a Band.Aid object. These files are used in creating roster objects and in graphing

Usage

```
buildMap( chrom.band.file,
                      map.label,
                      spot.ID.lbl,
                      loc.start.lbl,
                      loc.center.lbl,
                      loc.stop.lbl,
                      Chrom.lbl,
                      Fine.Band.lbl,
                      Chrom.labels,
                      chrom.band.file.sep=",",
                      mapped.by.lbl=NA,
```

```
                    map.flag.lbl=NA,
                    random.lbl=NA,
                    genome.loc.lbl=NA,
                    rm.spotID.lbls=NA,
                    maxnChrom = 1:24,
                    XchromNum = 23,
                    YchromNum = 24,
                    centromere.file=NA,
                    centromere.loc.lbl="location",
                    centromere.file.sep="\t")
```

Arguments

`chrom.band.file`
: name or path to file containing spotID, chromosome and fine band information

`map.label`
: name for created RData map file, this should be unique for each map

`spot.ID.lbl`
: name of the column in chrom.band.file representing spot labels (e.g. Clone, Probe.ID)

`loc.start.lbl`
: name of the column in chrom.band.file representing start locations

`loc.center.lbl`
: name of the column in chrom.band.file representing center locations

`loc.stop.lbl`
: name of the colum in chrom.band.file representing stop locations

`Chrom.lbl`
: name of the column in chrom.band.file representing chromosomes information

`Fine.Band.lbl`
: name of the column in chrom.band.file representing band information

`Chrom.labels`
: labels indicating how the chromosomes in the chromosome column of the chrom.band.file are labeled

`chrom.band.file.sep`
: seperation character for chrom.band.file, used as sep in call to read.table

`mapped.by.lbl`
: name of the column in chrom.band.file representing a mapped by field, this may be NA

`map.flag.lbl`
: name of the column in chrom.band.file representing map flag, see flags in details, this may be NA

`random.lbl`
: name of the column in chrom.band.file representing any additional field to be stored, in some Agilent files this field would be useful for storing a flag for spotIDs with a random label, this may be NA

`genome.loc.lbl`
: name of the column in chrom.band.file representing genomic location, this may be NA

`rm.spotID.lbls`
: list of spotID names to be removed from map file if they occur, this may be NA (e.g. c=("EMPTY", "H2O", "HK134K"))

`maxnChrom`
: a numeric vector of 1 to the total number of chromosomes. The default is the human chromosome, 1:24.

`XchromNum`
: The number of the maxnChrom that represents the X Chromosome. The default is for a human chromosome, 23.

`YchromNum` The number of the maxnChrom that represents the Y Chromosome. The default is for a human chromosome, 24.

`centromere.file`
name or path to file containing centromere locations. The default, NA, is there is no centromere file

`centromere.loc.lbl`
name of the column in the centromere.file representing location

`centromere.file.sep`
seperation character for centromere file, used as sep in call to read.table

Details

`buildMap` loads in a file `chrom.band.file`.It stores information about the spotID, chromosome, band, and start,center,and stop locations in a data.frame. A second file `centromere.file` is optionally loaded. This file must contain the locations of the centromeres. The band information, and the centromere information if provided, is stored in an object Band.Aid. Both the data.frame and the Band.Aid object are stored as a .RData file in the local RData directory under the name `map.label`. This object contains mapping information needed when creating roster objects and in graphics functions.

The `chrom.band.file` file must contain a single header line indicating column names. It must minimally containing columns for the following: spotID names (i.e. Clones, Probes), start locations, center locations, stop locations, chromosomes names, and bands. Bands will be parsed into fields referencing chromosome, arm, broad.band, and fine.band. The genomic location may be given or if NA will be estimated. All other fields are optional. This data frame object takes on the name specified in `map.label`.

The `centromere.file` file, if present, must contain the location information for the centromeres. The locations must be given in order. All other data in this file is ignored.

The data.frame object `map.label` that is created contains the following fields:spotID, loc.start, loc.center, loc.stop, loc.genome, Chrom, iChrom, Arm, iArm, Broad.Band, iBroad.Band, Fine.Band, iFine.Band, mapped.by, map.flag, and random. spotID, loc.start, loc.center, loc.stop, Chrom, Fine.Band, and optionally loc.genome are defined directly from columns in the input file. If loc.genome is not specified, it along with iChrom, Arm, iArm, Broad.Band, iBroad.Band, and iFine.Band are calculated within the function. mapped.by, map.flag, and random are optionally defined columns from the input file.

This list `Band.Aid` consists of a data.frame `Centromeres` containing the data from the centeromere file if it was given, and a list `Regions`. This list contains a data.frame for Chrom, Arm, Broad.Band, and Fine.Band. Each indicates the Chromosome location, the label to be used (useful when graphing), and the lower,center, and upper bound locations.

Value

creates a .RData file in the RData directory with the name specified as `map.label`. This file contains two objects: a data frame object of the name `map.label` and a list object `Band.Aid`. See details

Note

The file loaded must meet the requirements described in the details section. If data is missing or not in the correct format, one must create a new file with the requirements.(often the case with agilent)

This may involve minimal bioinformatics/R programming knowledge to create either a new file or objects of the correct format. See also examples

Author(s)

Lori Shepherd

References

See Also

`Write.aCGH.ex1`, `Centromeres`,`HB19Kv2.HG18.map`

Examples

```
#
# Example1:  we have provided HB19Kv2.HG18.map and Centromeres files
#            these files are created and stored in the  working
#            directory with the call Write.aCGH.ex1()
#            To run any and all examples, Write.aCGH.ex1() must have been
#            called
#

library("aCGHplus")
# initializes directories and data needed for running any and all examples
Write.aCGH.ex1()

# indicates how the chromosomes in the chromosome column of the
# HB19Kv2.HG18.map.csv file are labeled: needed for factoring
ch.label = c("chr1","chr2","chr3","chr4","chr5","chr6","chr7","chr8","chr9","chr10","chr1

# this is run with HB19K_v2_HG18 mapping
# This contains Centromere location data
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
         map.label="HB19Kv2.HG18",
         spot.ID.lbl="Clone",
         loc.start.lbl="start",
         loc.center.lbl="Center",
         loc.stop.lbl="Stop",
         Chrom.lbl="Chromosome",
         Fine.Band.lbl="Band",
         Chrom.labels= ch.label,
         chrom.band.file.sep=",",
         mapped.by.lbl="Mapped.by",
         map.flag.lbl="Flag",
         random.lbl=NA,
         genome.loc.lbl="g\_loc",
         rm.spotID.lbls=c("EMPTY","H2O"),
         maxnChrom = 1:24,
         XchromNum = 23,
         YchromNum = 24,
         centromere.file =  "Centromeres.txt",
         centromere.loc.lbl="location",
         centromere.file.sep="\t")
```

`CBSBatch` *PERFORM CIRCULAR BINARY SEGMENTATION ON BATCH OF SAMPLES IN aCGH OBJECT*

Description

These functions perform circular binary segmentation on samples in an aCGH object in batch mode.

Usage

```
CBS.Batch(aCGH,
          overwrite=FALSE, BatchDX=NA,nbatchjobs=2,ibatchjob=1,
          time.order=FALSE,alpha=0.025,nperm=100,outlier.mad.cut=5,CBS.label="CB

Assemble.CBS.Batch(aCGH,
                   CBS.label="CBS",overwrite.log2.ratios=FALSE,saveFlag=FALSE,
                   fileName="RData/aCGHwCBS.RData",vrb=TRUE,alpha=0.025,nperm=1
```

Arguments

`aCGH`	aCGH object
`overwrite`	flag to overwrite existing array.image files. If this is of length one, then it will be as specified for all samples. If this is of length of number of samples, each will be as specified. Default is to not overwrite files
`BatchDX`	index of aCGH samples to be run in batch mode
`nbatchjobs`	number of batch jobs
`ibatchjob`	the batch job turn in this call of CBS.Batch
`time.order`	logical if should order samples by time stamp
`alpha`	significance levels for the test to accept change-points. passed into DNAcopy's segment function
`nperm`	number of permutations used for p-value computation. passed into DNAcopy's segment function
`outlier.mad.cut`	
`CBS.label`	name of the object to store CBS data in the array.image and aCGH object
`vrb`	a logical flag indicating if status messages should be printed
`saveFlag`	logical indicating if new aCGH object should be saved to a file
`fileName`	name of the file to save if saveFlag is T
`overwrite.log2.ratios`	logical indicating if log2.ratios should be overwritten. If False, a new fields log2.ratios.fitted will be added to the aCGH object. Default is to not overwrite.

Details

Value

Each samples' array.image image file is updated to include a CBS component. An aCGH object with CBS data is returned when Assemble.CBS.Batch is called and is saved to a file if saveFlag is T

Note

Requires DNAcopy package CBS.Batch.ismpl is called from within the CBS.Batch function

Author(s)

Lori Shepherd

References

See Also

`fit.CBS`, `fitCBS.sample`, `CBS.papply`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# this will run the first batch job
# since there is a total of two jobs (nbatchjobs=2)
# this will run every other sample in the aCGH object beginning with
# the first element
# this may be run in one R session

CBS.Batch(aCGH,overwrite=FALSE, BatchDX=NA,
            nbatchjobs=2,ibatchjob=1, time.order=FALSE,
            alpha=.025, nperm=100, outlier.mad.cut=5,
            CBS.label="CBS", vrb=TRUE)

# this will run the second batch job
# since there is a total of two jobs (nbatchjobs=2)
# this will run every other sample in the inventory beginning with
# the second element
# this may be run in a second session of R

CBS.Batch(aCGH,overwrite=FALSE, BatchDX=NA,
            nbatchjobs=2,ibatchjob=2, time.order=FALSE,
            alpha=.025, nperm=100, outlier.mad.cut=5,
            CBS.label="CBS", vrb=TRUE)

# Now all the array.image files for the samples listed
# in the aCGH object have been updated, it is possible to created an
# aCGH object with the estimated values
```

```
aCGH = Assemble.CBS.Batch(aCGH, CBS.label="CBS", overwrite.log2.ratios=FALSE,
vrb=TRUE,saveFlag=FALSE)


#####
# to do one large batch job try

# CBS.Batch(aCGH, nbatchjobs=1, ibatchjob=1)
# aCGH = Assemble.CBS.Batch(aCGH)
```

`CBSExtras.Rd` *EXTRA HELPER FUNCTIONS AFTER CBS IS RIN*

Description

These functions may be used after Circulary Binary Segmentation is run on an aCGH object. nInaSegment returns the number of spotIDs/BACs in a given segment. segmentsBelowThreshold will return every segment that has fewer spots/BACs then a given threshold for a specific sample

Usage

```
nInaSegment(aCGH,
            sampleDX,
            segmentDX,
            vrb=TRUE)

segmentsBelowThreshold(aCGH,
                       sampleDX,
                       threshold=2,
                       vrb=TRUE)
```

Arguments

`aCGH`	an aCGH object
`sampleDX`	a single numeric value indicating which sample is in question
`segmentDX`	a single numeric value indicating which segment is in question
`vrb`	a logical flag indicating if status messages should be printed
`threshold`	a single numeric value indicating the minimum number of spots/BACs a segment should hold. The default is 2

Details

After CBS is applied to an aCGH object, these functions allow easy access to segment information.

Value

numeric vector

The return value of nInaSegment is a single numeric indicating the total number of spots in a given sample's (sampleDX) segment (segmentDX)

The return value of segmentsBelowThreshold is a numeric vector representing the numbers of all the segements of a sample (sampleDX) that are below a given threshold

Note

These are to be used after CBS has been used

Author(s)

Lori Shepherd

References

See Also

`fit.CBS`, `fitCBS.sample`, `CBSBatch`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


#this runs CBS
CBS.Batch(aCGH,overwrite=FALSE, BatchDX=NA,
            nbatchjobs=1,ibatchjob=1, time.order=FALSE,
            alpha=.025, nperm=100, outlier.mad.cut=5,
            CBS.label="CBS", vrb=TRUE)

aCGH = Assemble.CBS.Batch(aCGH, CBS.label="CBS", overwrite.log2.ratios=FALSE,
vrb=TRUE)


#the following will return the number of spots in the first aCGH sample's third
#segment

nSpots = nInaSegment(aCGH, sampleDX=1, segmentDX=3, vrb=TRUE)


#the following will return the number of all the first aCGH sample's
# segments that are below a threshold of 3 spots

segs = segmentsBelowThreshold(aCGH, sampleDX=1, threshold=3, vrb=TRUE)
```

`CBS.papply` *PERFORM CIRCULAR BINARY SEGMENTATION OF SAMPLES' OF AN aCGH OBJECT ON CLUSTER*

Description

These functions perform circular binary segmentation on samples in an aCGH object utilizing papply. This function will use papply to send multiple jobs to nodes on a cluster for efficiency.

Usage

```
CBS.papply(roster.file="RData/aCGHroster.RData",
           alpha=0.025,
           nperm=1000,
           outlier.mad.cut=5,
           CBS.label="CBS",
           overwrite=FALSE,
           vrb=TRUE)
```

Arguments

`roster.file`	path name of file containing the aCGHroster object to be made into an aCGH object
`alpha`	significance levels for the test to accept change-points. passed into DNAcopy's segment function
`nperm`	number of permutations used for p-value computation. passed into DNAcopy's segment function
`outlier.mad.cut`	
`CBS.label`	name of the object to store CBS data in the array.image and aCGH object
`overwrite`	flag to overwrite existing array.image files. If this is of length one, then it will be as specified for all samples. If this is of length of number of samples, each will be as specified. Default is to not overwrite files
`vrb`	a logical flag indicating if status messages should be printed

Details

This function is designed to work with a cluster. Therefore, cluster nodes have to be reserved and the following commands run at the linux command line: module load lam lamboot -v R

Load the package (aCGHplus, Rmpi and papply libraries must be loaded).

CBS.papply uses the papply function to send out multiple cbsWrap calls to different nodes. The cbsWrap function will perform circular binary segmentation on an individual sample's log2 ratios. The CBS.papply function will also Assemble the CBS data with the aCGHplus function Aseemble.CBS.Batch once all individual calls are completed. The aCGH object with CBS data is returned.

Value

CBS.papply returns an aCGH object

Note

This function uses the papply package.

Author(s)

Lori Shepherd

References

See Also

`fit.CBS`, `fitCBS.sample`,`CBSBatch`

Examples

`Centromeres` *EXAMPLE CENTROMERE LOCATION DATA*

Description

This data set gives the centromere location for each chromosome.

Format

A tab delimited text file containing two columns “chromosome” and “location”.

chromosome	chromosome number
location	location with respect to chromosome

Details

A centromere file must contain the centromere locations with respect to the associated chromosome, in chromosomal order.

Note

Source

References

See Also

Write.aCGH.ex1,buildMap

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# to see the centromere file we have provided
read.table("Centromeres.txt", sep="\t")
```

---

check.Band.Aid *CHECKS BAND AID AND MAPPING INFO OBJECTS CORRECT*

Description

Function to correct for legacy mapping code. This function will correct errors in the mapping.info genomic location and the Band.Aid locations.

Usage

```
check.Band.Aid(aCGH,
               fileName="RData/aCGH.RData")
```

Arguments

aCGH	an aCGH object
fileName	path name to save the corrected aCGH object if applicable

Details

Older legacy code had an error in the genomic mapping. To ensure that the genomic locations and band locations are correct, check.Band.Aid will compare all mappings and update as needed.

Value

an aCGH object

Note

Author(s)

Lori Shepherd

References

See Also

Examples

```
# This example assumes an aCGH object has already been created

load("RData/aCGH.RData")
aCGH = check.Band.Aid(aCGH)
```

---

checkImages *CHECK IF ARRAY IMAGE OBJECT IS CREATED OFF INVENTORY AND IF FLAT FILE EXISTED FOR FILES*

Description

These functions provide additional checks to see if objects were created. checkImagesMade will run through the inventory and see if the object was created. checkFlatFilesExist will take a list of files and return whether they have any flat files associated with them.

Usage

```
checkImagesMade(inventory,
                sep=";",
                path="array.images/images/")

checkFlatFilesExist(files,
                    path = "arrays/",
                    patterns =c("_635.txt","_532.txt",".txt"))
```

Arguments

`inventory`	name or path to an inventory file
`sep`	seperation character for the inventory file. The default is a semi-colon
`path`	Path to images. In checkImagesMade, this is the path to the array image files created with a create.array.image or create.roster call. In checkFlatFilesExist, this is the path to the image analysis flat files.
`files`	A list of files to check against a directory. This should be a root name if you would like an accurate count on how many files were found for sample. (i.e HBmyfile$_5$32$.txt, shouldbeHBmyfile)Listofpatternstoremove$

Details

patterns

checkImagesMade is a check to see if create.array.images or create.roster ran correctly. It will cycle through all samples of a given inventory and see if there is an associated image file in the path specified. This assumes an image has the same root name of the image.name of the inventory. It keeps a count of the files that failed and those that were created successfully, as well as the image.name and the inventory entry for the files that failed.

checkFlatFiles can be used with the above checkImageMade call or on its own. checkFlatFiles takes in a list of image root names and a path to a given directory; this should be the directory holding the raw image analysis flat files. The function will get a list of all the files in the specified directory removing patterns listed in the patterns argument. It then matches the list of files to the directory indicating how many files are found for each sample.

checkFlatFiles will indicate if there was a problem excuting a call to create.array.images or if there is an outside error such as an incorrect inventory name or missing files. When used together, one would pass in the checkImageMade object's slot for filesFailed into the files argument of checkFlatFiles.

Value

checkImagesMade will return an object out that contains slots success, failed, filesFailed, and inventory. Success is a numeric of how many of the inventory samples were created successfully, Failed is a numeric of how many samples were not created, while filesFailed lists the root name of the samples that failed. Inventory is a data frame of the inventory entries of the samples that failed.

checkFlatFiles will return a data frame l. This data frame will list the name of the file, if an image analysis file exists, and the number of files found for the sample.

Note

Author(s)

Lori Shepherd

References

See Also

checkObjects, create.array.images, create.roster, reportMissingInvEntries, reportMissingArrays

Examples

```
library(aCGHplus)

# creates directories and files needed for running example
Write.aCGH.ex1()

# a build file for each of the chip designs used in analyzing the
# images must be built. For our example this is the HB19K_v2_HG18
# this builds that mapping information file
 buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
```

```
            map.label="HB19Kv2.HG18",
           spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
           loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
           Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
               "chr7","chr8","chr9","chr10","chr11","chr12",
               "chr13","chr14","chr15","chr16","chr17","chr18",
               "chr19","chr20","chr21","chr22","chrX","chrY"),
           chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
           map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
           rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
           YchromNum = 24, centromere.file =  "Centromeres.txt",
           centromere.loc.lbl="location", centromere.file.sep="\t")

# this will create an aCGHroster object off the example inventory file
# aCGH.ex1.inv

aCGHroster=create.roster.R(array.dir="arrays",
            roster.file="RData/aCGHroster.RData",
            inventory.file="aCGH.ex1.inv", image.dir="array.images",
            inv.sep=",", overwrite=FALSE, returnFlag=TRUE, BatchDX=NA, vrb=TRUE)

checkCreated = checkImagesMade(inventory="aCGH.ex1.inv", sep=",",
path="array.images/images/")

checkCreated

# load inventory
inv = read.table("aCGH.ex1.inv", sep=",",header=TRUE)
imageFiles = inv$image.name

checkFlatFiles = checkFlatFilesExist(files=imageFiles, path = "arrays/", patterns =c("_63

checkFlatFiles
```

`checkInv` *CHECKS FOR AND REMOVES DUPLICATE ENTRIES*

Description

This function will search a given inventory file for duplicate entries. Duplicate entries are not allowed and therefore removed

Usage

```
checkInv(file,
         sep,
         vrb=TRUE,
         outfile=NA,
         duplicate.file="Duplicate.inv",
         appendFile=TRUE)
```

Arguments

`file`	inventory file to check
`sep`	seperation character to read in inventory file
`vrb`	logical indicating if status messages should be printed
`outfile`	name of the new inventory file
`duplicate.file`	file name to save duplicate entries for manual edit
`appendFile`	logical indicating if file already exists should new data be appended

Details

The file will be checked for duplicate entries, which is not allowed.Image.name should be unique; If an image.name has more than one entry, all entries are removed and places in a duplicate inventory file. If inclusion of any of these images is desired, the user must manually chose which of the duplicate entries to use and place them in the new inventory created.The duplicate file and the new inventory will be saved.

Value

An updated inventory file, a copy of the original inventory file, and a duplicate inventory if applicable,are saved

Note

Author(s)

Lori shepherd

References

See Also

`add.inventory`, `combineInvFiles`

Examples

`checkObjects` *CHECKS IF AN OBJECTS HAS BEEN CREATED WITHIN AN aCGH or aCGHRoster OBJECT OR ASSOCIATED ARRAY IMAGE FILES*

Description

These functions will check all files of all of a given object's samples to see if an object (the result of a process) has been created.

Usage

```
checkObjectsFromACGH(aCGH,
                     object="CBS",
                     path="array.images",
                     vrb=TRUE,
                     saveFile=FALSE,
                     saveName="checkCBSarrayimage" )

checkObjectsFromRoster(aCGHroster,
                       object="smooth2D",
                       path="array.images",
                       vrb=TRUE,
                       saveFile=FALSE,
                       saveName="checkSmootharrayimage")
```

Arguments

`aCGH`	an aCGH object
`aCGHroster`	an aCGH roster object
`object`	name of desired object or data set in question
`path`	path to get to object
`vrb`	a logical indicating if status messages should be printed
`saveFile`	logical indicating if an output file should save results of check for each sample
`saveName`	Name of file to store results of check

Details

These functions allow for a check over all samples in an aCGH or aCGHroster object to see if an object has been created (indicates if a process has been run or where a process failed). checkImagesFromACGH's default is to search for a CBS object, if CBS has been performed a CBS object will exist in the samples array.image file. checkImageFromRoster's default is to search for a smooth2D object,if smooth2D has been performed a smooth2D object will exist in the sample's arry.image file. The object in question must be specified, as well as the path through other objects to the object in question. The default for checkImageFromACGH has an object="CBS" and path="array.images". If the user wished to see if CBS has been assembled and a log2.ratios fitted object exists, object="log2.ratios.fitted" and path="aCGH". If The user wished to see if the object Grid was created, object="Grid" and path="array.images$matrix$".$Thea$

Value

returns an object out with the number of files that successfully contain the object, the number of files that failed and do not have the object, and an index of the failed samples in the given aCGH or aCGHroster object. If saveFile this object is also saved as a file

Note

Author(s)

Lori Shepherd

References

See Also

checkInv

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# it is important to note here that the example aCGH object
# has already had CBS performed on it.

out = checkObjectsFromACGH(aCGH)
out

# should see success =6, failed = 0, idxfiles = 0
# if CBS had not been run yet success=0, failed=6, idxfiles=1 2 3 4 5 6
```

---

choiceAct *makes interactive legend; used within selectPt; user never calls*

Description

This function is used within selectPt for interactive 3D principle component analysis viewer. This give details concerning the interactive legend options

Usage

```
choiceAct(aCGH,
       PCtest,
       sampleInfo,
       factorInfo,
       plotInfo,
       pt2,
       gen,
       heat)
```

Arguments

aCGH	An aCGH object
PCtest	object containing principle component analysis data
sampleInfo	object containing data for samples
factorInfo	object containing factor information
plotInfo	object containing information for graphical devices
heat	flag for activated heat map. options are "cy", "background" or "correction" heat maps.
pt2	flag for last point chosen before empty space.
gen	flag for genome graph. options are "PCA" or "gen".

Details

choiceAct is used to interrogate principle component data and aCGH samples. It is called from within selectPt. There are 12 options a user may select from on the interactive legend: back to PCA pt selection, change factor type, toggle heat maps, toggle PCA/genome plot, comment sample, score sample, flag for removal, remove all flagged, recalculate PCA, save inventory, save aCGH, exit all. back to PCA pt selection returns the user to the 3D PCA graph for selection of a new sample. Change factor type utilizes the factorInfo object created in initFactor. The user is able to cycle through all factors specified when creating this object. The legend and graph colors are automatically updated. There are currently three sets of heat maps the user may toggle when selecting toggle heat maps: cy3 and cy5 chips, background cy3 and cy5, and mxy before correction and mxy after correction. They are cycled in that order. The genome plot may be toggled by selecting toggle PCA/genome plot. This allows the user to flip between a genome plot for the selected sample and a PCA plot ove the genome. Comment sample will store a character string for the object. The user may, therefore, make notes while viewing the sample genome and chips. If the user has already made a comment for the sample, additional comments are added to the end of the current string and are seperated with a space. The user enters the comment on the R command line. When finished the user should press enter. Score sample allows the user to give a numerical score to the current select sample. If a score was already given, the new score replaces the old score. The user enters the score on the R command line. When finsihed the user should press enter. Flagging sample will flag the sample for removal. The index of the sample is stored in an array exclude. All flagged samples may be removed by selected removing flagged samples. All removed samples are stored in an index removed. Sample scores and comments, and the arrays exclude and removed are stored in the sampleInfo object. The sampleInfo object is created in makePCAfile If a user wishes to see how removed sample[s] effect the principle component analysis, the user may select to recalculate PCA. The user will have an option to continue with the current re-calculated principle component anaylsis or to go back to the previous principle component anaylsis. If the user decides to continue with the new PCA, there will also be an option to save the old PCA before

continuing. Saving inventory will save the principle component data, as well as the sample and factor information. The user will be asked to enter a name for the new inventory file on the R command prompt. When the user if finished entering the name they should press enter. If the user does not specify a name, the date and time are used as the inventory name. The file is saved in the PCA.inventory directory. The aCGH object is also updated with an object PCAinv. This is a list of all inventory files associated with that aCGH object from old to new. The user will have the option to replace the previously saved inventory file. This will not delete the old file in the PCA.inventory, however it will delete the file name from the aCGH objects PCAinv. This is a way for the user to continuously save work. Saving aCGH will save the aCGH object. This allows the changes made to the aCGH object when saving inventory files to be saved. The user will have the option to replace the previous aCGH file. If the user chooses not to replace the old file, they will be asked to enter a file name on the R command prompt. The user should press enter when finished. If the user does not specify a name, the date and time are used as the new aCGH file name. The final option the user has on the legend is to exit all. The user will have the options to save the inventory and the aCGH object before exiting. All graphics windows will close automatically.

Value

Note

The user should never use this call. It is internally called by selectPt

Author(s)

Lori Shepherd

See Also

`runPCAFile`, `selectPt`

Examples

---

cmean *MAKES INTERACTIVE MEAN PLOT*

Description

This function will make a mean plot of all the sample the values in an aCGH object. This plot is then interactive.

Usage

```
cmean(aCGH,
      rmNA = TRUE,
      maxNA=0.5,
      fitVls=TRUE,
      recenter=TRUE,
      bacwin=10,
```

```
grp.lbls = NA,
orientation.flipped=1
)
```

Arguments

aCGH	an aCGH object
rmNA	a logical indicating if NA should be removed from the calculation of mean across samples
maxNA	a numerical indication of the percentage of sample data that must be present for the sample to be included
fitVls	logical value indicating if the fitted log2 ratios are used or the raw log 2 ratios are used initially in all graphs
recenter	logical value indicating if the log2 ratios should be recentered
bacwin	numeric value indicating the number of bacwins (+/-) that will be shown on subsequent graph of surrounding area of a selected point
grp.lbls	character vector of factor objects (see makeFactorObject)
orientation.flipped	value in orientation field of aCGH inventory representing flipped/dye swaps. Any sample with this value for the orientation will have their log2 ratios multipled by a -1 for analysis

Details

makeMeanPlot will create an interactive plot of the aCGH object's samples' mean log2 ratio values. The user may indicate whether the fitted log2 ratios or the raw log2 ratios are used to create the mean plot by setting fitVls. fitVls default is to use the fitted values. The user may toggle between fit and raw values interactivly after the plot is drawn. If no fit values are found (CBS not run) the apllication will use the raw values automatically. The user may also specify whether the data should be recentered. The default is to recenter data. The default will recenter data based on autosomes removing NAs. If other recenter methods are desired, the aCGH object should be recentered before use in this function and the function recenter should be set to F.

There is a cutoff range for the amount of data that must be observed for a sample to be included. maxNA is initially set to .5; half the data must be observed for the sample to be included. This value may be changed interactivly.

If there are dye swaps/orientation flips, samples will have log ratios opposite than what is normal. To correct for this effect we multiple a -1 on all sample values whose orientation suggests flipped.

The program will initially open with two blank windows, a mean graph, and a legend window. The legend window provides users withthe following options: select point, select BAC/spotID, raw, fitted, see table, write table to file, change NA cutoff, or exit all. Clicking on select point will activate the mean graph for interaction. The user may then click upon any sample point in the graph of the means to view an area more closely. The selected point will be circled and the section surrounded by blue vertical lines. The two initially blank windows will plot two different graphs. One will be a flanking graph, showing sample log2 ratios for +/- the number of bacwins (default +/- 20) surrounding the selected spot/BAC. The sample at the BAC/spotID will be in a bold color, the flanking spots in faded color. Data plotted as a + character indicates observed values. Data plotted as an open square indicates missing/estimated values. The color indicates a factor type. The second graph window shows the sample log2 ratios at the selected BAC/spotID. The names of the samples are used as plotting character. Bold colors indicate real values, faded indicate observed/estimated values. In order for a user to select a different point on the frequency plot they must first re-select

Select point on the legend device. Clicking on select BAC/spotID allows the user to enter the name of a specific BAC or spot.ID in question. This must match exactly to the name of a spot.ID in the aCGH object. Capitalization and spacing matters; type carefully. It will activate the graphs similarly to the select point circling the specified spotID and creating additional graphs. Clicking on raw or fitted will alter the log 2 ratio values. All three plots will be updated if the user has activated the two subsequent graphs. Raw will update to raw log 2 ratios and fitted will graph the fitted log 2 ratios if available. See table will show a table of all the samples for the selected spot.ID, the samples' log2 ratios, the samples' fitted log 2 ratios (if available), and the factor group they belong to. Write table will write the table viewed using see table to a file. The file will be saved in a new directory created called ValuesAtspotIDs. The NA cutoff is changed on the R command line prompt. The user will be directed to enter a numeric value corresponding to the percentage of data observed that must be present for a sample to be included in the graph. Clicking on exit all will close all windows and exit the application. The user may click on any of the legend options at anytime.

Value

An interactive plot of the mean log2 ratio

Note

`makeFactorObject` should be run before running makeMeanPlot. The main difference between meanPlot and cmean is cmean uses an aCGH object, meanPlot will load in files. called in meanPlot

Author(s)

Lori Shepherd

References

See Also

`meanPlot`, `makeFactorObject`, `getMean`,`recenterData`, `setCutoff`, `makeFlank`, `makeSample`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# runs makeFactor Object

  aCGH = makeFactorObject(aCGHloaded=TRUE, invloaded = TRUE, factorBy="sex")

  load("RData/vizMean.RData")

  # because this is interactive the code has been commented out

  # cmean(aCGH,rmNA = TRUE,maxNA= .5,fitVls= FALSE,recenter= FALSE,bacwin=10,grp.lbls=grp
```

`combinedACGH` *COMBINED TWO aCGH OBJECTS*

Description

This function will combined two aCGH objects

Usage

```
combinedACGH(aCGH1,
             aCGH2,
             saveFile=TRUE,
             fileName="RData/newACGH.RData",
             vrb=TRUE)
```

Arguments

`aCGH1`	an aCGH object or the pathname to a file containing aCGH object
`aCGH2`	an aCGH object or the pathname to a file containing aCGH object
`saveFile`	logical if the new combined aCGH object should be saved
`fileName`	if saveFile, the pathname to save the object
`vrb`	logical indicating if status messages should be printed

Details

This function will take in two aCGH objects. As long as both aCGH objects were mapped with the same mapping object, they will be combined into a single aCGH.

Value

aCGH object

Note

Author(s)

Lori Shepherd

References

See Also

`make.aCGH`, `subsetACGH`

Examples

`combineInvFiles` *COMBINES TWO INVENTORY FILES*

Description

This function will combined two inventory files into one.

Usage

```
combineInvFiles(file1,
                file2,
                sep1,
                sep2,
                outfile="MasterInv.inv",
                outsep="\t",
                vrb=TRUE,
                overwrite=FALSE,
                duplicate.file="Duplicate.inv")
```

Arguments

`file1`	Name of the first inventory file
`file2`	Name of the second inventory file
`sep1`	The separation character for the first inventory file. Will be passed in as sep for a read.table call
`sep2`	The separation character for the second inventory file. Will be passed in as sep for a read.table call
`outfile`	Name/path to save the updated combine inventory file. The default will save in the current directory as MasterInv.inv
`outsep`	The seperation character to use in writing the new combine inventory file. This will be passed in as sep in a write.table call
`vrb`	a logical flag indicating if status messages should be printed
`overwrite`	Flag to overwrite existing values. The default will not overwrite and use all values from the first given inventory file. If overwrite is tripped, if any of the columns in the second inventory are already in the first inventory, the values of the second inventory will be used, replacing the first inventories values.
`duplicate.file`	file name to save duplicate entries for manual edit

Details

`combineInvFiles` takes in two inventory files to combine into one inventory file. First the application will determine which samples are the same in both inventories and which samples from the second inventory will be added to the first based on sample.ID AND image.name. Next, columns will be checked. If the same samples are in both inventories, and a column name is the same in both files, the values of the first inventroy are used, unless overwrite is tripped. If overwrite is True, the values of matching samples of matching columns, will assume the value of the second inventory.Any additional columns in the second inventory will be added to the first inventory. NAs

will be added where values do not exist. Next, any additional samples in the second inventory will be added to the end of the first inventory. If columns are missing, NAs will be added. Finally, the file will be checked for duplicate entries, which is not allowed.Image.name should be unique; If an image.name has more than one entry, all entries are removed and places in a duplicate inventory file. If inclusion of any of these images is desired, the user must manually chose which of the duplicate entries to use and place them in the new combine inventory created.The duplicate file and the new combined inventory will be saved.

Value

A combination inventory file, and duplicate inventory if applicable,are written to the current directory

Note

Author(s)

Lori Shepherd

References

See Also

`add.inventory`, `checkInv`, `add.supInv.toInvFiles`

Examples

```
library(aCGHplus)

# This function will use the example inventory given in Write.aCGH.ex1

Write.aCGH.ex1()

# we now create a fake inventory file to combined with given inventory
inv = list()
inv$sample.ID = c("sampleA", "sampleB", "test")
inv$image.name = c("HB19Kv2-sample1","HB19Kv2-sample3", "test")
inv$Test = c("test", "test", "test")
inv = as.data.frame(inv)
write.table(inv, file="combineEx.inv", sep=",", row.names=FALSE)

combineInvFiles(file1 = "aCGH.ex1.inv",
                file2 = "combineEx.inv",
                sep1=",",
                sep2=",",
                outfile="CombinedInv.inv",
                outsep="\t",
                vrb=TRUE,
                overwrite=FALSE,
                duplicate.file = "duplicate.inv")
```

```
# to see new inventory file
read.table("CombinedInv.inv", sep="\t", header=TRUE)
```

`convert.old.inv.R` *CREATES REQUIRED INVENTORY FILE AND SUPPLEMENTAL INVENTORY FILE*

Description

This function will take in a provided inventory file, and parse it into a required master inventory file and a supplemental inventory file.

Usage

```
convert.old.inv.R(fname.input,
                          fname.output,
                          input.inv.sep=",",
                          output.inv.sep=",",
                          default.load.specs,
                          default.image.soft,
                          default.design,
                          default.orientation,
                          default.map,
                          default.User=NA,
                          reqd.cols=c("sample.ID","image.name","hyb.date",
                                      "load.specs","image.soft","design",
                                      "default.map","sex","orientation",
                                      "User","sample.type","comment"))
```

Arguments

`fname.input` provided inventory file that will be converted

`fname.output` root file name for the two output files,which will have the suffixes .rqd.inv (required file) and .add.inv (supplemental)

`input.inv.sep`
character that separates fields in the provided inventory, will be passed as sep to a read.table call

`output.inv.sep`
character to separate fields in the output files,will be passed as sep to a write.table call

`default.load.specs`
The name of the default load.specs file if there is no load.specs column in the provided inventory. i.e. "Imagene.load.specs" ; see `Imagene.load.specs`

`default.image.soft`
The name of the default image software used for analysis if there is no image.soft column in the provided inventory i.e. "ImageneV6" or "FeatureExtraction"

`default.design`
The name of the default design file if there is no design column in the provided inventory i.e. "HB19Kv2.design"; see `HB19K2.design`

`default.orientation`
: The default chip orientation if there is no orientation column in the provided inventory

`default.map`
: The name/path to the default map file if there is no default.map column in the provided inventory i.e."RData/RP11.19Kv2.HG18.RData"; see `buildMap`

`default.User`
: The name of the default principle investigator if there is no User column in the provided inventory

`reqd.cols`
: Name of the columns to be included in the master inventory file. Any additional columns in the provided inventory will be in the supplemental inventory file.

Details

The master inventory must minimally have the following columns: sample.ID, image.name, load.specs, image.soft, design, orientation, default.map, and User. Any additional columns specified in `reqd.col` will also be included in the master inventory file. All additional columns in the provided inventory file will be included in the supplemental inventory file. Only the required inventory will be added to the aCGHroster object initially. The supplimental inventory file, or any other additional inventory files may be added to the aCGHroster object using the `add.inventory` function

Value

Two inventory files will be created in the working directory: `fname.output`.rqd.inv and `fname.output`.add.inv. `fname.output`.rqd.inv represents the master inventory file and `fname.output`.add.inv represents the additional supplemental inventory file.

Note

Author(s)

Lori Shepherd

References

See Also

`add.inventory`

Examples

```
library(aCGHplus)

# this will use the provided inventory aCGH.ex1.inv
Write.aCGH.ex1()

# This will take the given inventory aCGH.ex1.inv and split
# into an add inventory and a required inventory
# The required inventory will contain all columns listed in
# the argument reqd.cols
```

```
# In this example we removed sex from reqd.cols; therefore the add
# inventory will contain sampleID, image.name and sex while the
# remaining columns will be in the rqd inventory

convert.old.inv.R(fname.input = "aCGH.ex1.inv",
                        fname.output= "SplitInvEx",
                        input.inv.sep=",",
                        output.inv.sep=",",
                        default.load.specs ="Imagene" ,
                        default.image.soft ="Imagene",
                        default.design ="HB19K",
                        default.orientation =0,
                        default.map ="RData/HB19Kv2.HG18.RData",
                        default.User=NA,
                        reqd.cols=c("sample.ID","image.name","hyb.date",
                                    "load.specs","image.soft","design",
                                    "default.map","orientation",
                                    "User","sample.type","comment"))
```

---

create.array.images

CREATE ARRAY IMAGE FILES FROM RAW FILES FROM IMAGE ANALYSIS SOFTWARE

Description

This function will take in a raw image analysis file name, and an associated load.specs file (see details) and create image file in the specified directory.

Usage

```
create.array.images(cyroot,
                            load.specs="Imagene.load.specs.csv",
                            array.dir="arrays",
                            image.dir="array.images",
                            design.name=NA,
                            image.soft=NA,
                            vrb=TRUE)
```

Arguments

cyroot	The root name of the image to load. see details.
load.specs	the name or path to a comma delimited text or table file which holds information about the columns in the chip image analysis files. The default is the name of the spec file given as part of the example dataset; It is an example Imagene spec file. See details
array.dir	the name or path to the directory containing the raw chip image files from image analysis software
image.dir	the name of the directory that will store created images
design.name	the chip design label
image.soft	the name or identifier for image software used
vrb	a logical flag indicating if status messages should be printed

Details

cyroot is the root name of the image to load. It is an object which indicates the file name without associate suffix (i.e .txt, $_6 35_{,5}\, 32$). $The root name of Sample1.txt is Sample1.$

$The \boldsymbol{load.specs} file is a comma delimited text or table. It contains information about the names of column headers in the$

$This function takes in a cyroot object holding file information for a sample and an assocated load.spec file containing in$

Value

A .RData file with the root name of the sample is stores an array.images object in the directory **image.dir**/images/

Note

Author(s)

References

See Also

`Write.aCGH.ex1` `Imagene.load.specs`

Examples

```
#
# this example uses output from Imagene software
#

library(aCGHplus)

# loads in data for examples
Write.aCGH.ex1()

cyinfo="HB19Kv2-sample1"

create.array.images(cyroot=cyinfo, load.specs="Imagene.load.specs.csv",
                    array.dir="arrays", image.dir="array.images",
                    design.name=NA, image.soft=NA, vrb=TRUE)
```

`create.GenomeOneRow`

CREATE AN EMPTY GENOMIC PLOT

Description

This function will make the shell of a genomic plot for an aCGH object

Usage

```
#To initialize graph:
InitGOR(aCGH, spotDX=NA, maxXlab=NA,
                  ylim=c(-1,1),
                  main="",xlab="",ylab="",Yaxis=TRUE,
                  zeroLine=TRUE,sectionLines=TRUE, chrmLines = FALSE, chrm.lty=
                  largeChrLabels=FALSE, chrmLabels=FALSE,
                  cex.axis=0.5,chrom.label.col="sienna",chrom.label.cex=2,addCh

#To redraw graph:
ReInitGOR(GORplt,...)
```

Arguments

`aCGH`	an aCGH object
`spotDX`	index of specific spots in the aCGH object to use; The default or if NA will use all spots
`maxXlab`	maximum number of x labels the graph will have; this is used in deciding which labels to use (chromsome,arm,broad.band,or fine.band). The default or if NA will use 24 (if entire genome=chromosome)
`ylim`	the bounds of the y axis, default is -1 to 1
`main`	The title of the graph
`xlab`	The x axis label
`ylab`	The y axis label
`Yaxis`	logical flag for if the y axis should be plotted
`zeroLine`	logical flag for if a horizontal line should be drawn at zero
`sectionLines`	logical flag for if vertical lines should be drawn at each of the labels
`chrmLines`	logical flag for if vertical lines should be drawn at each chromosome.
`chrm.lty`	type of line to be drawn if chrmLines=T. Default is a single continuous line
`largeChrLabels`	logical indicating if large chromosome labels should be drawn at the top of the graph
`chrmLabels`	logical if chromosome labels should be drawn on the X-axis
`chrom.label.col`	color of the chromosome labels
`chrom.label.cex`	size of the large chromosome labels

addChrmToSec	logical if chromosome number should be attached to x-axis section labels. If section labels are already chromosome, function will automatically flag false
cex.axis	axis scale
bffr	ignore this argument it is not functional at this time
...	additional arguments that will be passed into a plot call
GORplt	The plotting object created as output from InitGOR

Details

This function will take an aCGH object and create an empty genomic plot. The spots used for the x axis can be subset by specifing a spotDX. This will take only the spots indentified in the spotDX for plotting. The graph will automatically determine the best labelling possiblites based on the specified maximum number of x labels, maxXlab. Depending the the spotDX, the labels will be either chromosome number, arm, broad.band, or fine.band. Section lines will plot vertical lines at each of the automatically generated labels. zeroLines will plot a horizontal line where y=0. Bffr adds a buffer of white space at the beginning and end of each section for better viewing.

Value

The shell of a genomic plot.

InitGOR: A plotting object that stores all information needed to re-draw the empty graph.

Note

Author(s)

Lori Shepherd

References

See Also

addTo.GenomeOneRow, make.aCGH

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


  # QuickTips:
  # if export.array.images has been used you may load an aCGH object with
  #         load("RData/aCGH.RData")
  # if you have gone through the process of making an aCGHroster object
  #         load("RData/aCGHroster.RData")
  #         aCGH = make.aCGH(aCGHroster)
```

```
  # This will draw an entire genome plot with chromosome numbers as labels
  # section lines and zeroLines are also plotted

  plt1=InitGOR(aCGH)

  # This will draw a genome plot with arms as labels
  # section lines, zeroLines, and chromosome lines/labels are included

  plt2=InitGOR(aCGH, spotDX=10000:12000, maxXlab=10)

  #This will redraw the initial full genome plot

  ReInitGOR(plt1)


graphics.off()
```

`create.multiGenomePlot`

PLOTS MULTIPLE GENOMIC PLOTS

Description

This function will plot 1 to 3 genomic plots of varing vectors and sizes.

Usage

```
pltG(aCGH,
     numberOfRows = 3,
     minChromInRow = c(1,5,12),
     maxChromInRow = c(4,11,24),
     vrb=TRUE,
     chrmLabels = TRUE,
     maxXlab=NA,
     maintitle = "",
     ttl = "",
     xlab = "",
     ylab ="log2 T/C",
     ...)

ReInitpltG(pltList)
```

Arguments

`aCGH`	an aCGH object
`numberOfRows`	the number of plots desired. This is a numeric equal to 1,2, or 3.

`minChromInRow`
: The minimum chromosome number to begin plotting. This should be of length equal to the numberOfRows. The default will begin plotting chromosome 1 on the first graph, chromosome 5 on the second graph, and chromosome 12 on the third graph.

`maxChromInRow`
: The maximum chromosome number to begin plotting. This should be of length equal to the numberOfRows. The defaults will plot to and including the fourth chromosome on the first graph, the eleventh chromosome on the second graph, and the 24 chromosome(Y) on the third graph.

`vrb`
: a logical indicating if status messages should be printed

`chrmLabels`
: logical indicating if the label for the X axis should be just chromosome numbers

`maxXlab`
: If chrom.lab is false, this is the maximum number of x labels the graph will have; this is used in deciding which labels to use (chromsome,arm,broad.band,or fine.band. If maxXlab is not NA and the numberOfRows does not equal the length of the maxXlab, the first maxXlab is used for all graphs

`maintitle`

`ttl`
: The title of the graph

`xlab`
: The x axis label. This should be of length equal to numberOfRows. The first entry corresponds to the first graph and so on.

`ylab`
: The y axis label. This should be of length equal to numberOfRows.The first entry corresponds to the first graph and so on.

`...`
: Additional arguements to be passed into GenomeOneRow besides maxXlab, main, xlab, and ylab.

`pltList`
: A plotting object created with the call pltG

Details

This function creates genomic plots. It is similar to plotGenome but each row of the graph may be specified. In other words, each row of the graph is a unique graph that is over a given genomic range. Each graph may have different labels. The graphs only depict ranges over entire chromosomes. Therefore part of a chromosome may not be viewed.

Value

creates empty genomic plots

pltList: a plotting object that stores all information needed to re-draw the empty graph[s]. This is returned only by pltG.

Note

uses GenomeOneRow, eval.js

Author(s)

Lori Shepherd

References

See Also

`create.GenomeOneRow`, `plotGenome`, `GenomeImage`, `addTo.multiGenomePlot`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

  # QuickTips:
  # if export.array.images has been used you may load an aCGH object with
  #         load("RData/aCGH.RData")
  # if you have gone through the process of making an aCGHroster object
  #         load("RData/aCGHroster.RData")
  #         aCGH = make.aCGH(aCGHroster)

  # The follow will create a plot similar to plotGenome
  pltList = pltG(aCGH)

  # The following will plot two plots split evening across the Genome
  pltList2 = pltG(aCGH,numberOfRows=2, minChromInRow=1)

  # The following will plot three graphs
  # The first is the entire genome
  # The second a plot from the 7th to 9th chromosome
  # The thire the 21 to the Y chromosome

  pltList3 = pltG(aCGH, minChromInRow = c(1,7,21), maxChromInRow = c(24,9,24))

  # The following will reinitialize the second plot graphed of a two row
  # of equal length

  ReInitpltG(pltList2)


graphics.off()
```

`create.roster` *CREATES ACGH ROSTER OBJECT*

Description

This function uses an inventory file to load in all sample arrays. An array image file is created and a mapping is performed for each sample. The object saved as a .RData file in the RData directory.

Usage

```
create.roster.R(array.dir="arrays",
                        roster.file="RData/aCGHroster.RData",
```

```
                    inventory.file="aCGH.ex1.inv",
                    image.dir="array.images",
                    inv.sep=",",
                    overwrite=FALSE,
                    returnFlag=FALSE,
                    BatchDX=NA,
                    vrb=TRUE)
```

Arguments

`array.dir`	name or path to the directory holding the raw data files from image software
`roster.file`	the name or path to save the roster object. The default is to save it as aCGHroster.RData in the RData directory (i.e. RData/aCGHroster.RData)
`inventory.file`	Name of the inventory file listing samples that will make up the aCGHroster. The default uses the inventory given as part of the example dataset. (see also example `aCGH.ex1.inv`)
`image.dir`	name of the directory to store image array files and map files. The subdirectories images, maps, and designs will be created within this directory
`inv.sep`	The separation character for the inventory file. Will be passed in as sep for a read.table call
`overwrite`	flag to overwrite existing array.image files. If this is of length one, then it will be as specified for all samples. If this is of length of number of samples, each will be as specified. Default is to not overwrite files
`returnFlag`	flag to return the aCGHroster object once it is created. Default is false; the aCGHroster object will be saved but not returned
`BatchDX`	Flag for running create.roster in batch mode. Dafault is NA and will run all nsmpls. If an index is specified batch mode will section off pieces of the inventory BatchDX long to run.
`vrb`	a logical flag indicating if status messages should be printed

Details

This function requires an inventory file (see note).

CheckInv should be performed on the inventory file before running create roster.

This function will load in raw data files and create array.image files for each sample specified in the inventory file. A design file is created for each sample's assocaited chip designs and a mapping for each image file to a default map file is performed. Both the sample chip design and the default map are specified in the inventory file. If a BatchDX is indicated, the above process is done in batch mode. An aCGHroster object is created indexing all samples of the inventory. This object is saved as a .RData file.

Value

if returnFlag is T, returns object aCGHroster of class aCGHroster
aCGHroster is saved as specified `roster.file`

Note

No column or data in the inventory file may contain the symbol '', as it is recognized as a comment in R

create.roster is powered by the given inventory file. The inventory file determines how many samples are in the given object. The inventory file should contain the following mandatory columns:sample.ID, image.name, load.specs, image.soft, design, and default.map. The sample.ID is the sample name, image.name is the name of the associated array image file for that sample, load.specs is the name of the spec file associated with that sample, image.soft is the name of the software used to create raw image files, design is the name of the chip design, and default.map is the name of the RData map build file associated with the chip design. An inventory file may have any number of additional columns.In the example file, additional inventory information includes: sex, orientation, PI, sample.type, and comment (see `aCGH.ex1.inv`).

The map files should already be build for each array.image design listed in the inventory (see `buildMap` and `map.array.images`)

This function uses `create.array.images`, `design.array.images`, and `map.array.images`

In order for the design file to be built, a file containing design data must be present. please refer to `design.array.images` or `HB19Kv2.design`

Author(s)

Lori Shepherd

References

See Also

`buildMap`,`checkInv`, `create.array.images`,`design.array.images`, `map.array.images`, `Write.aCGH.ex1`, `aCGH.ex1.inv`, `RosterBatch`, `Roster.papply`

Examples

```
library(aCGHplus)

# creates directories and files needed for running example
Write.aCGH.ex1()

# a build file for each of the chip designs used in analyzing the
# images must be built. For our example this is the HB19K_v2_HG18
# this builds that mapping information file
 buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
          map.label="HB19Kv2.HG18",
         spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
         loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
         Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
             "chr7","chr8","chr9","chr10","chr11","chr12",
             "chr13","chr14","chr15","chr16","chr17","chr18",
             "chr19","chr20","chr21","chr22","chrX","chrY"),
         chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
         map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
         rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
```

```
          YchromNum = 24, centromere.file =  "Centromeres.txt",
          centromere.loc.lbl="location", centromere.file.sep="\t")

# this will create an aCGHroster object off the example inventory file
# aCGH.ex1.inv
aCGHroster=create.roster.R(array.dir="arrays",
             roster.file="RData/aCGHroster.RData",
             inventory.file="aCGH.ex1.inv", image.dir="array.images",
             inv.sep=",", overwrite=FALSE, returnFlag=TRUE, BatchDX=NA, vrb=TRUE)
```

`design.array.images` *CREATES DESIGN FILE FOR THE CHIP THE SAMPLE WAS RUN ON*

Description

Stores useful information about the chip design

Usage

```
design.array.images(fname.array.images,
                            design.data,
                            vrb=TRUE)
```

Arguments

`fname.array.images`	file name for the array.image
`design.data`	name of file holding design data
`vrb`	a logical flag indicating if status messages should be printed

Details

This function will read in the chip design of a sample, load in the design file for the chip if present, and store useful design variables. The function checks if the design.data name passed in as an argument matches the design name stored for the sample's array image. If the two do not match, the array image file design name is updated to match the design file given.

Value

A .RData file is stored in the array.image subdirectory designs with the name of the design

Note

This file is not unique to the samples, it is unique to the chip design. If more than one sample has the same chip design, the file is not re-created. The sample will use the same design file as the previous sample's. After the array image design file is updated (if necessary), if the file already exists the function will exit.

Author(s)

Lori Shepherd

References

See Also

Write.aCGH.ex1, create.array.images,HB19Kv2.design

Examples

```
  library(aCGHplus)
  Write.aCGH.ex1()

# a sample's array image must have been created
# this will build an array image file for sample1
  cyroot="HB19Kv2-sample1"
  cy3file="arrays/HB19Kv2-sample1_532.txt"
  cy5file="arrays/HB19Kv2-sample1_635.txt"
  cyinfo=list(cyroot=cyroot,cy3file=cy3file,cy5file=cy5file)
  create.array.images(cyroot=cyinfo, load.specs="Imagene.load.specs.csv",
                  array.dir="arrays", image.dir="array.images",
                  design.name=NA, image.soft=NA, vrb=TRUE)

 # with the array image file we now can create a design file for
 # sample1's chip design
 # This will be a design file for the HB19Kv2 chip design
  design.array.images(
      fname.array.images="array.images/images/HB19Kv2-sample1.RData",
      design.data = "HB19Kv2.design",
      vrb=TRUE)
```

---

design.list *DESIGN OBJECT*

Description

A design list object is a list containing information about the chip design. This file gets created if a design file was provided by the user.

Format

see details

Details

The object structure is as follows (note: the class of the objects are in parenthesis):

design.list (list).

design.levels (list) information on levels of certain design criteria specified in the design file. These are therefore variable. We will list the objects created based on the example file:

MasterRow the list of all rows for the chip

MasterCol the list of all columns for the chip

MetaRow	the list of grid rows
MetaCol	the list of grid columns
GridRow	the list of rows within one grid - print tip
GridCol	the list of columns within one grid - print tip
Pin	the list of pins used to spot chip
Plate	the list of plates used to spot chip
PlateRow	the list of rows of the plate
PlateCol	the list of columns of the plate
Repetition	the list of how many replicate spots are on the chip
BAC	the names of the spots - BAC names, Clone names, etc

design.matrix (list) these are all matricies. The data matricies created is dependent on design criteria specified in the design file. These are therefore variable. For every matrix in this list there is an entry in the design.levels with the levels for that matricies. The matrix objects created in our given example therefore are the same as above:

MasterRow	row location with respect to the chip
MasterCol	column location with respect to the chip
MetaRow	row location with respect to the grid
MetaCol	column location with respect to the grid
GridRow	row location within one grid - print tip
GridCol	column location within one grid - print tip
Pin	pin used to spot location
Plate	plate used to spot location
PlateRow	row from the plate which spots location
PlateCol	column from the plate which spots location
Repetition	replicate spot
BAC	the names of the spot - BAC names, Clone names, etc

Note

Again the objects created are variable depending on the information provided in the design file.

design object files holding the design.list object are located in the array.image subdirectory designs. The example design object file given in the package is: array.images/designs/HB19Kv2.design.RData

Source

References

See Also

`HB19Kv2.design`,`design.array.images`

Examples

```
# To see an example of a design file object
Write.aCGH.ex2()
load("array.images/designs/HB19Kv2.design.RData")

names(design.list)

# use the $ to subset objects

names(design.list$design.matrix)
```

---

dianosticRosterCheck

RANDOM CHECKS FOR ROSTER

Description

This function will perform a check on the roster object to ensure dimensions and mappings were stored correctly.

Usage

```
diagnosticRosterCheck(aCGHroster,
                      quickCheck=TRUE,
                      vrb=TRUE,
                      saveTofile = TRUE,
                      fileName="diagnosticRosterCheck" )
```

Arguments

`aCGHroster`	an aCGHroster object
`quickCheck`	logical if a quick roster check or a more detailed roster check should be performed
`vrb`	logical indicating if status messages should be printed
`saveTofile`	logical if results of checks should be outputted to a file
`fileName`	file name to save output

Details

This application has a quick version and a detailed version controlled by specifing quickCheck. The quick check will check the order of the image names with the order of the inventory to ensure that inventory entries match up with the correct sample number. The detailed check will still check the inventory versus the image name list. It will then load each samples' array image file and check the inventory entries for the roster with the mapping file, design file, the actual used design/mapping file, and the name of the sample.

Value

a list of flags for which samples passed and which failed

a file with details for each sample is created if saveTofile. It is given the name fileName.

Note

Author(s)

Lori Shepherd

References

See Also

`create.roster`, QC1report

Examples

```
library(aCGHplus)

# We make sure an aCGHroster object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see create.roster

Write.aCGH.ex2()
load("RData/aCGHroster.RData")

out = diagnosticRosterCheck(aCGHroster, quickCheck=TRUE, saveTofile=FALSE)


out = diagnosticRosterCheck(aCGHroster, quickCheck=FALSE, saveTofile=TRUE,fileName="diagn
```

DiagPlot.Smooth2D *INTERACTIVE PLOT OF SMOOTH2D OUTPUT AND SPOT TO CHIP LOCATION*

Description

This function creates a user interactive plot. The user can click on any point in a chromosome plot to see that sample's spot locations on the chip, and before and after smooth2D graphs.

Usage

```
DiagPlot.Smooth2D(aCGH,
                  ismpl=1,
                  heatMats.object.labels=c("array.images$smooth2D$LambdaList$Mxy
                            "array.images$smooth2D$LambdaList$Mxy.loess.list[[1
                            "array.images$smooth2D$LambdaList$MxyFit.loess.list
                            "array.images$smooth2D$LambdaList$Axy.original.list
                            "array.images$matrix$cy5$Signal.Mean",
                            "array.images$matrix$cy3$Signal.Mean",
                            "array.images$matrix$cy5$Background.Mean",
                            "array.images$matrix$cy3$Background.Mean",
```

```
                "array.images$matrix$cy5$Signal.Stdev",
                "array.images$matrix$cy3$Signal.Stdev",
                "array.images$matrix$cy5$Background.Stdev",
                "array.images$matrix$cy3$Background.Stdev"),
    heatMatsLbls=c("raw log2(T/C) values",
                "loess corrected log2(T/C) values",
                "fitted loess log2(T/C) values",
                "yprod",
                "cy5mat",
                "cy3mat",
                "Back.cy5mat",
                "Backcy3mat",
                "sd.cy5mat",
                "sd.cy3mat",
                "sd.Back.cy5mat",
                "sd.Back.cy3mat"),
    Mxy.original.label="array.images$smooth2D$LambdaList$Mxy.origi
    Axy.original.label="array.images$smooth2D$LambdaList$Axy.origi
    FlagMat.label="array.images$matrix$cy3$Flag",
    MxyFit.loess.label="array.images$smooth2D$LambdaList$MxyFit.lo
    Mxy.loess.label="array.images$smooth2D$LambdaList$Mxy.loess.li
    Mxy.noDesign.label="array.images$smooth2D$LambdaList$Mxy.noDes
    Mxy.Smooth2D.label="array.images$smooth2D$LambdaList$Mxy.smoot
    GridMat.label="array.images$matrix$Grid",
    dbg=TRUE
    )
```

Arguments

`aCGH`	an aCGH object
`ismpl`	an index of which sample to begin viewing in the aCGH object
`heatMats.object.labels`	list of paths to matricies that should be able to be viewed through the heatmap toggle
`heatMatsLbls`	labels to use as titles for when the above list of heatMats.object.labels are displayed
`Mxy.original.label`	path to values of the difference of log2 tumor and log2 control
`Axy.original.label`	path to values of the addition of log2 tumor and log2 control
`FlagMat.label`	path to matrix of spot flags
`MxyFit.loess.label`	path to
`Mxy.loess.label`	path to
`Mxy.noDesign.label`	path to
`Mxy.Smooth2D.label`	path to

`GridMat.label` path to

`dbg` logical indicating if status messages should be printed

Details

Value

interactive plots

Note

Author(s)

Lori Shepherd

References

See Also

`SmoothArray`, `smooth2D.papply`, `SmoothBatch`

Examples

`DNAcopyWrapper` *CONVERTS ACGH TO DNAcopy OBJECT*

Description

This function will take in an aCGH object and return a list of DNAcopy objects

Usage

```
makeDNAcopyObjs(aCGH,
                subdx=NA,
                smoothCNA=TRUE,
                segmentCNA=TRUE,
                vrb=TRUE)
```

Arguments

aCGH	an aCGH object
subdx	index of samples to use. If NA, all samples are used
smoothCNA	logical indicating if smooth.CNA should be called on the CNA object
segmentCNA	logical indicating if segment should be called on the CNA object
vrb	logical indicating if status messages should be printed

Details

This function will take in an aCGH object and make DNAcopy objects. This function will get the necessary arguments to call DNAcopy functions CNA, smooth.CNA, and segment.

Value

an list object containing DNACopy objects.

Note

Author(s)

Lori Shepherd

References

See Also

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

library(DNAcopy)

DNA = makeDNAcopyObjs(aCGH)

plot.DNAcopy(DNA$segment, plot.type="w")
```

`eval.js` *A JAVASCRIPT-LIKE EVAL FUNCTION*

Description

This function evaluates expressions.

Usage

```
eval.js(expr,
        envir=parent.frame(),
        enclos=if(is.list(envir)||is.pairlist(envir)) parent.frame())
```

Arguments

`expr`	character string of an expression to evaluate
`envir`	passed to eval function. see eval
`enclos`	passed to eval function. see eval

Details

Value

will return the evaluated expression

Note

uses function eval from base package

Author(s)

Lori Shepherd

References

See Also

`eval`

Examples

```
df = list()
df$a = rep(1,5)
df$b = rep("one",5)
df = as.data.frame(df)

#for comparison view
```

```
df

eval.js("df$new = NA")
df

###########

# or

load("RData/aCGH.RData")
load(aCGH$data.info$fname.array.images[1])

# this will create an object lgT
Tmat = "array.images$matrix$cy5$Signal.Mean"
eval.js(paste("lgT=log2(",Tmat,")",sep=""))
```

`export.array.images` *EXPORT aCGH OBJECT AND ASSOCIATED FILES*

Description

This function takes an aCGH object, saves the aCGH object as a file, and creates a tar of the aCGH object with all associated files

Usage

```
export.array.images(aCGH,
                            fname.tgz="aCGH.data.tgz",
                            fname.aCGH="RData/aCGH.RData",
                            vrb=TRUE)
```

Arguments

`aCGH`	aCGH object
`fname.tgz`	name of the tar file; should end in .tgz
`fname.aCGH`	path/name to save aCGH object; we recommend saving in the RData directory and therefore to begin with RData/; should end in .RData
`vrb`	a logical flag indicating if status messages should be printed

Details

This function saves an aCGH object. It will create a list of all files associated with the aCGH object, save the list, and create a tar ball of the files.

Value

A portable version of the aCGH object and array.images as a tar file

Note

The aCGH files will overwrite any existing files if they have the same name. It is therefore advisable to enter values for fname.tgz and fname.aCGH rather than leaving defaults.

Author(s)

Lori Shepherd

References

See Also

`make.aCGH`, `Export.sample`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

   # QuickTips:
   # if export.array.images has been used you may load an aCGH object with
   #         load("RData/aCGH.RData")
   # if you have gone through the process of making an aCGHroster object
   #         load("RData/aCGHroster.RData")
   #         aCGH = make.aCGH(aCGHroster)


   # the aCGH object is stored as a .RData file in RData directory
   # a tar is created

   export.array.images(aCGH)
```

Export.sample *CREATE SAMPLE FILE WITH SPOT.ID LISTING AND LOG RATIOS*

Description

This function will create a file for a sample[s] with the spot.IDs, The log2 ratios at each spot, the fitted (CBS) log2 ratios at each spot, and spot flags.

Usage

```
Export.sample.j(j,
                aCGHroster,
                CBS.label = "array.images$CBS"
                )

ExportSamples.papply(aCGHroster,
                     smplDX=NA,
                     export.dir="test.export",
                     CBS.label = "array.images$CBS",
                     export.sep=","
                     )
```

Arguments

`j`	index of individual sample to export
`aCGHroster`	an aCGH or aCGHroster object
`CBS.label`	the path name to the CBS object created during circular binary segmentation. This path should be one level above the log2.ratios and log2.fitted ratios for the sample
`smplDX`	an index of all samples that should be exported
`export.dir`	name of the directory that will contain all output files
`export.sep`	seperation character for the output files

Details

ExportSamples.papply is designed to work with a cluster. Therefore, cluster nodes have to be reserved and the following commands run at the linux command line: module load lam lamboot -v R Load the package (aCGHplus, Rmpi and papply libraries must be loaded). ExportSamples.papply utilizes the internal fuction XportWraps. XportWrap takes an individual sample entry and creates the sample's file containing spot.IDs, spot.flags, and log2.ratios by calling the function Export.sample.j. ExportSamples.papply uses the papply function to send out the multiple XportWrap calls to different nodes. The files are generated with the sample image.name as the file name in the directory specified as export.dir.

Export.sample.j takes in an individual sample index and an aCGH or aCGHroster object. The sample's array.image file is loaded which should contain CBS data. The path to the CBS object in the array.image object should be given as CBS.label. This path should lead to the structure that contains log2.ratios and log2.fitted ratios. This data along with the spot.IDs and any spot flags are combined into a data.frame. This data.frame is returned by the function.

Value

Export.sample.j returns a data.frame of the sample information

ExportSamples.papply returns nothing but the appropriate files are created for each sample desired in the given directory.

Note

This assumes CBS has already been performed on data.

ExportSamples.papply uses the papply package.

Author(s)

Lori Shepherd

References

See Also

`export.array.images`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# This will create a data frame of data for sample 3
smplDF = Export.sample.j(3, aCGH, "array.images$CBS")
```

---

factorObj — *performs factoring/leveling and store useful information in factorInfo object*

Description

This function takes in an aCGH object and the name of a column in the aCGHH inventory. The samples are then factores/leveled in accordance with the specified column. It stores all factor information in a factorInfo object

Usage

```
factorObj(aCGH,
       smpldx = NA,
       factorType = NA)
```

Arguments

`aCGH`	An aCGH object
`smpldx`	a vector indicating which samples to use in calculation. The defualt uses all samples in the aCGH object
`factorType`	The name of the column within the aCGH inventory to be used as a factor. The default creates an all inclusive type, essentially making no factor type

Details

`factorObj` takes in a specific aCGH object to factor inventory. The user may indicate which samples within the aCGH to factor. The default uses all aCGH samples. FactorType is the name of desired factor in aCGH inventory. The defualt will create a unique factor "all" which is inclusive of all samples. All factor information is stored in a factorInfo object. This includes: facIdx, Ftype, iFtype, and ntype. facIdx is the location of the factor type in the aCGH inventory Ftype is the vecotr of samples factored by the current factor type iFtype is the numeric version of Ftype ntype is the total number of different types of the current factor

Value

factorInfo

Note

The difference between initFactor and factorObj is initFactor stores information for cycling through a list of factors, factorObj factors by a single value in the inventory. factorType must also be of type character.

Author(s)

Lori Shepherd

See Also

`initFactor`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# makes all inclusive factor type
fac1 = factorObj(aCGH)
fac1

fac2 = factorObj(aCGH, factorType="sex")
fac2
```

---

fillInMissingIntensities

FILLS IN MISSING INTENSITIES OF INTENSITY MATRIX

Description

This function fills in missing values of an intensity matrix.

Usage

```
fillInMissingIntensities(intensity,
                         value=0)
```

Arguments

`intensity`	an intensity matrix
`value`	value to be filled in for missing

Details

This function should be performed after the aCGHgetIntensityMatrix function is called. The vsn function does not allow missing data, this function will place in value for missing values.

Value

an intesnity matrix

Note

to be used after aCGHgetIntensityMatrix

Author(s)

Lori Shepherd

References

See Also

`aCGHgetIntensityMatrix`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

mm = aCGHgetIntensityMatrix(aCGH)
mm = fillInMissingIntensities(mm)

library(Biobase)
library(vsn)

mm = vsn(mm)

plot(exprs(mm), pch=".")
meanSdPlot(mm)
vsnPlotPar(mm, "factors")
vsnPlotPar(mm, "offsets")
```

`fit.CBS` *PERFORM CIRCULAR BINARY SEGMENTATION ON aCGH OBJECT*

Description

This function takes in an aCGH object and performs circular binary segmentation over each sample.

Usage

```
fitCBS(aCGH,
       alpha=0.025,nperm=100,outlier.mad.cut=5,vrb=TRUE)
```

Arguments

`aCGH`	aCGH object
`alpha`	significance levels for the test to accept change-points. passed into DNAcopy's segment function
`nperm`	number of permutations used for p-value computation. passed into DNAcopy's segment function
`outlier.mad.cut`	
`vrb`	a logical flag indicating if status messages should be printed

Details

Value

an aCGH object with DNAcopy object and log2.ratios.fitted

Note

Requires DNAcopy package – use CBSBatch and fitCBS.sample over fitCBS

Author(s)

Lori Shepherd

References

See Also

`CBSBatch`, `fitCBS.sample`, `CBS.papply`

Examples

```
#
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

 aCGH = fitCBS(aCGH)

# this was the same as running the defaults
# aCGH = fitCBS(aCGH,alpha=0.025,nperm=100,outlier.mad.cut=5,vrb=TRUE)
```

---

fitCBS.sample *PERFORMS CIRCULAR BINARY SEGMENTATION ON A SAMPLE*

Description

This function takes in an aCGH object and sample number, and performs circular binary segmentation on that sample

Usage

```
CBS.Batch.ismpl(aCGH,ismpl,
                alpha=0.025,nperm=100,outlier.mad.cut=5,
                CBS.label="CBS",overwrite=FALSE,vrb=TRUE)
```

Arguments

aCGH	an aCGH object
ismpl	sample in aCGH object to work on
alpha	significance levels for the test to accept change-points. passed into DNAcopy's segment function
nperm	number of permutations used for p-value computation. passed into DNAcopy's segment function
outlier.mad.cut	
CBS.label	name of object to store CBS data in array.images
overwrite	logical indicating if samples' have already undergone CBS should they be overwritten.
vrb	a logical flag indicating if status messages should be printed

Details

Value

The sample's array image file is updated to include CBS data

Note

requires package DNAcopy

Author(s)

Lori Shepherd

References

See Also

`fit.CBS`, `CBSBatch`, `CBS.papply`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


#
# CBS.Batch.ismpl is called from within the CBS.Batch function
# However, if it is necessary to run CBS on a single aCGH object
# sample it may be used
# the following would run on aCGH second sample

CBS.Batch.ismpl(aCGH,ismpl=2,
                alpha=0.025,nperm=100,outlier.mad.cut=5,CBS.label="CBS",vrb=TRUE)

# note since this uses the defaults it may also have been run as
# CBS.Batch.ismpl(aCGH, ismpl=2)
```

---

flankNA.CBS *ESTIMATES MISSING VALUES*

Description

This function takes in an aCGH object and fills in missing values with flanking fitted values.

Usage

```
flankNA.CBS(aCGH,
            NAmatFlag=TRUE,
            saveFlag=FALSE,
            fileName="RData/aCGHwNmat.RData")
```

Arguments

aCGH	an aCGH object
NAmatFlag	a logical indicating if an NAmat should be created
saveFlag	a logical indicating if new aCGH object should be saved as a file
fileName	name of file to save to if saveFlag=T

Details

This function takes in an aCGH object and estimates any missing log2.ratio values with flanking fitted values. It takes an average using the flanking values if available

Value

aCGH with missing log2.ratios and log2.ratios.fitted estimated. If NAmatFlag is true, the NAmat of missing values is also saved. if saveFlag=T a file is also created saving aCGH object

Note

CBS should have been run so that aCGH has log2.ratios.fitted

Author(s)

Lori Shepherd

References

See Also

CBSBatch, fit.CBS

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


 # performs CBS
 aCGH = fitCBS(aCGH)
```

```
# estimates missing values
aCGH = flankNA.CBS(aCGH)

# this was the same as running the defaults
#  aCGH = flankNA.CBS(aCGH, NAmatFlag=TRUE, saveFlag=FALSE)
```

`freqPlot` *CREATES FREQUENCY PLOT*

Description

This function will create frequency plot(s). It also has an option to save the plots to a file.

Usage

```
makeFqPlot(aCGH,
           smplDX=NA,
           maxNA=.25,
           fitVls=TRUE,
           ylims=c(-1,1),
           cutoffs=c(-.15,.15),
           chrm = NA,
           saveToFile = FALSE,
           fileName="freq.plot",
           orientation.flipped=1)
```

Arguments

`aCGH`	an aCGH object
`smplDX`	index of samples to use to calculate frequency
`maxNA`	maximum percentage of data that can be missing for sample inclusion
`fitVls`	logical indicating if log2 ratios fitted (T) or raw log 2 ratios (F) should be used
`ylims`	min and max value for the y axis
`cutoffs`	min and max aberration.
`chrm`	numeric list of chromosomes to plot. NA(default) will only plot entire genome
`saveToFile`	logical indicating if plot(s) should be saved as a file
`fileName`	if saveToFile, the name of the file. Files are automatically generated in the plots directory as a postscript and pdf.
`orientation.flipped`	value in orientation field of aCGH inventory representing flipped/dye swaps. Any sample with this value for the orientation will have their log2 ratios multipled by a -1 for analysis

Details

This function will make a frequency plot of the aCGH object's log ratios. The user may specify if only certain samples should be included for analysis, as well as the maximum percentage of missing data a sample may have to be included. fitVls determines the the fitted or raw log2 ratios should be used. Ylims and cutoffs apply ranges to the data. cutoffs determine the aberrant samples. Any sample above the maximum cutoff is considered a gain; any sample belove the minimum cutoff is considered a loss. The cutoffs are therefore the range of 'normal' values. Ylims determines the yaxis labels and cutoffs. Any values above these ranges will be rounded to the bound. Chromosome is a numeric list of chromsomes that should be plotted. For every chromosome in the list, a plot of the frequency for that chromsome will be plotted. If not saving to a file, all graphs will open in new windows. If there are dye swaps/orientation flips, samples will have log ratios opposite than what is normal. To correct for this effect we multiple a -1 on all sample values whose orientation suggests flipped. It is possible to save all graphs to a file, fileName. saveToFile is the arguement to trip saving a file. The file will be saved in the plots directory as a postscript and pdf file.

Value

a frequency plot of the entire genome and of any specified chromosomes. These plots are saved in a file is saveToFile is tripped

Note

Author(s)

Lori Shepherd

References

See Also

`plotMean.vs.freq`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# this makes only the genome wide plot
makeFqPlot(aCGH)

# this makes the genome plot and a closer view of chromosomes 5,8,and 12
makeFqPlot(aCGH, chrm=c(5,8,12))

graphics.off()
```

`Genome3d` *CREATES A 3D VERSIONS OF GENOME PLOT*

Description

This function takes in an aCGH object and sample number and creates a 3d genomic pixilated plot

Usage

```
GenomeRGL(aCGH,
          ismpl,
          nxcut=500,
          nycut=100,
          ttl="",
          col.graph= "lightblue")
```

Arguments

`aCGH`	an aCGH object
`ismpl`	number of the sample in the aCGH object to graph
`nxcut`	Number of x cuts
`nycut`	Number of y cuts
`ttl`	The title of the graph produced
`col.graph`	The col of the graph to produce

Details

This function creates a pixilated like genomic plot of the given aCGH sample (ismpl). The number of cuts to determine the number of sections is determined by nxcut and nycut. The graph will view each section relative to the number of points located within that particular section. This function utilizes persp3d of the RGL package

Value

A 3d Genomic plot

Note

This function required the RGL package

Author(s)

Lori Shepherd

References

RGL

See Also

create.GenomeOneRow, GenomeImage, plotGenome, RGL

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

GenomeRGL(aCGH,ismpl=1,
          nxcut=500, nycut=100, ttl="", col.graph= "lightblue")

# may also have been run as GenomeRGL(aCGH, ismpl=1)



rgl.close()
```

GenomeImage *CREATE GENOMIC PIXILATED-LIKE IMAGES*

Description

These functions will create a pixilated genomic image

Usage

```
InitGI(y,
       aCGH,
       bacDX=NA,
       nxcut=500,
       nycut=100,
       ylim=c(-2,2),
       ylab="log2 T/C",
       cex.axis=0.5,
       main="",
       xlab="",
       Yaxis=TRUE,
       Xaxis=TRUE,
       zeroLine=TRUE,
       sectionLines=TRUE,
       bffr=0,
       maxXlab=NA,
       ...)

ReInitGImg(GImg,...)
```

```
subsetGImg(aCGH,
           GImg,
           maxXlab=NA,
           subDX=NA,
           ...)
```

Arguments

y	vector of values that must be bacDX or nBAC long (length of number of spotIDs used)
bacDX	vector of which spotID/BACs to use
aCGH	an aCGH object
nxcut	The number of x breaks that will determine the number of horizontal sections for pixilated graph
nycut	The number of y breaks that will determine the number of vertical sections for pixilated graph
ylim	scale of y values to use
ylab	the y axis label
cex.axis	the axis size
main	main title of the graph
xlab	x label of the graph
Yaxis	logical indicating if the y axis should be plotted
Xaxis	logical indicating if the x axis should be plotted
zeroLine	logical indicating if a horizontal line should be drawn at 0
sectionLines	logical indicating if vertical lines should be drawn at section divides (chrom, arms, bands)
bffr	a cushion to seperate sections
maxXlab	maximum number of labels to have on the x axis. Will automatically determine based on the number of spots used whether to use chromosome, arm or bands
GImg	a GenomeImage Object created by InitGI and/or subsetGImg
subDX	an index to subset a GenomeImage object by with respect to the number of spotIDs/BACs
...	additional arguments to be passed into plot call

Details

`InitGI` will initialize a pixilated-like genomic graph of an aCGH object. The user must specify a vector of y values, a vector to subset the spot.IDs by, and an aCGH object. If a bacDX is not specified, the dafault (NA) is to use all spot.IDs. The x axis labels are automatically determined based on the maximum number of x labels specified by maxXlab. nxcut and nycuts will determine the number of sections the pixilated graph will contain. nxcuts represents the number of vertical cuts, nycuts the number of horizonal cuts. This will make a nxcut by nycut graph. The graph will be generated by shading the above sections based on the total number of points within the area. See R's graphics package `image` function. This reduces the number of points generated to aid in graphing large datasets, such as Agilent 244K data. The output is a genomic graph and a GImg object containing all information and data needed to recreate the genomic plot.

subsetGImg will take a GImg object (created by InitGI or this function) and subset based on a given subDX. If no subDX is given, The default, NA, uses the previous bacDX saved in the GImg object. It uses all the saved information in the GImg object for labels and cuts. The genomic plot is generated in the same procedure as InitGI and also returns a GImg object.

ReInitGImg will redraw a pixilated-like genomic plot using a GImg object created by InitGI or subsetGImg.

Value

InitGI and subsetGImg will return a GImg object containing all information for creating and graphing the generated pixilated graph. All three functions will create pixilated-like genomic plots.

Note

R function image from the graphics package is used

Author(s)

Lori Shepherd

References

R function image from the graphics package is used

See Also

GenomeZoom

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


  # This will graph sample 1's entire genomic plot

  y = aCGH$log2.ratios[,1]
  bacDX = 1:aCGH$data.info$nBAC
  GImg = InitGI(y,aCGH,bacDX,
                nxcut =500, nycut=100, ylim =c(-2,2),
                ylab="log2 T/C", cex.axis=.5, main="", xlab="", Yaxis=TRUE, Xaxis=TRUE,
                zeroLine = TRUE, sectionLines = TRUE, maxXlab=24, bffr=0)

  ## since this mainly uses the defaults it may also have been run using
  #  InitGI(y, aCGH)

  # this will subset the first plot to spots 1:1000

  subGI = subsetGImg(aCGH,GImg,maxXlab= NA,subDX=1:1000)


   # This will Re draw the original plot
```

```
   ReInitGImg(GImg)


graphics.off()
```

GenomeZoom *CREATE INTERACTIVE GENOMIC PIXILATED-LIKE IMAGES*

Description

These functions will create an interactive pixilated-like genomic image

Usage

```
GenomeZoom(aCGH,
           smpl,
           ylim=c(-2,2),
           zeroLine=TRUE,
           sectionLines=TRUE,
           maxXlab=NA,
           ptLimit=20,
           spotLimit=20,
           spotLines=FALSE)


GImgZoom(GImg,
         aCGH,
         maxXlab=NA,
         ptLimit=20,
         spotLimit=20,
         spotLines=FALSE)
```

Arguments

aCGH	an aCGH object
smpl	the number of the aCGH object's sample to use
ylim	scale of y values to use
zeroLine	logical indicating if a horizontal line should be drawn at 0
sectionLines	logical indicating if vertical lines should be drawn at section divides (chrom, arms, bands)
maxXlab	maximum number of labels to have on the x axis. Will automatically determine based on the number of spots used whether to use chromosome, arm or bands
ptLimit	The minimum number of points to graph. During subsetting (halving), once the number of points is less than the ptLimit subsetting is haulted

`spotLimit`	The minimum number of points for spotID/BACs graph to plot. During subsetting (halving) , once the number of points is less than the spotLimit a second graph showing the spotIDs/BACs for all available points is drawn
`spotLines`	logical indicating if lines should be drawn at all the spotIDs/BACs in the graph
`GImg`	a GenomeImage Object created by InitGI and/or subsetGImg

Details

`GenomeZoom` takes in an aCGH object and sample number to create an interactive genomic pixilated-like plot of that sample. Initially a single graph of the entire genome is displayed and two empty windows will appear. Once a region/point is selected by left clicking, One of the windows will remain blank, the other will display the arm of the region/point selected. The graph of the arm is now the active plot. A user may continue to select regions/points on this graph. Once a point/region is selected by left clicking, the graph will be halved, taking a region surrounding the point, the first half of the graph, or the last half of the graph. Halving will continue until a minimum number of points is reached which is determines by the specified ptLimit. If the user right clicks, the previous graph will be displayed. The user therefore may naviate backwards any number of times until the original graph is reached. The original graph will be reactivated when the top level is reached. The third graph will remain blank until a threshold of points specified by spotLimit is reached. Once this limit is reached the third graph will display the surrounding area with spotID or BAC labels.

`GImgZoom` works the exact same way as `GenomeZoom` except it begins with a GImg object created by InitGI or subsetGImg, and begins immediately with halving not a zoom into an arm.

To exit close all graphics windows

Value

creates interactive genomic graph

Note

This function calls InitGI, ReInitGImg, and subsetGImg

Author(s)

Lori Shepherd

References

See Also

`GenomeImage`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
```

```
load("RData/aCGH.RData")



# because these are user intereactive graphs they have
# been commented out

  # This will graph sample 1's entire genomic plot

 # GenomeZoom(aCGH,1)



## This is using the defaults: could also have been run as:
# GenomeZoom(aCGH,1,ylim=c(-2,2),zeroLine=TRUE,sectionLines=TRUE,
#            maxXlab=NA,ptLimit=20,spotLimit=20,spotLines=FALSE)



# first create GImg
#    this will use sample 3 of the aCGH object

  GImg = InitGI(y= aCGH$log2.ratios[,3], aCGH, bacDX=1:10000, maxXlab=10)

#  GImgZoom(GImg, aCGH)

## This is using the defaults: could also have been run as:
#   GImgZoom(GImg,aCGH,maxXlab=NA,ptLimit=20,spotLimit=20,
#            spotLines=FALSE)



graphics.off()
```

`getBlockMap` *GET BEFORE CORRECTION HEAT IMAGE WITH MAPPED BLOCKS*

Description

These are a utility functions to determine if chip images were mapped and blocks specified correctly

Usage

```
getBlockMap(aCGH,
            smplDX=NA,
            fileName="BlockQC",
            pdfFlag=TRUE,
            ...)

makeBlockMap.image(array.image,
                   chipMat= "array.images$smooth2D.noDesign$LambdaList$Mxy.origi
                   gridMat= "array.images$matrix$Grid",
```

```
                    FlagMat= "array.images$matrix$cy3$Flag",
                    autoFlag = NA,
                    blockPath = "array.images$matrix$Block",
                    ttl = NA,
                    ...)

labelBlocks(array.image, blockPath)
```

Arguments

aCGH	an aCGH or aCGHroster object
smplDX	an index of samples to plot. If NA, all samples will be used
fileName	name of the file that will store images. This file will be created in the plots directory
pdfFlag	flag indicating if a pdf version of the file should be created
array.image	path name to a sample's array.image file
chipMat	path to the before correction matrix of values
gridMat	path to the grid matrix values
FlagMat	path to the matrix of flagged spots. This matrix spots have failed if values are not equal to zero
autoFlag	path to a generated autoFlag matrix, where spots are flagged by having a value equal to zero
blockPath	path to the matrix of Block values at spots
ttl	title for plot
...	additional arguments for the image.acgh.chip function call

Details

There are certain file types that will provide block information with within grid dimension information. The aCGHplus load in process can load in files with this information. Information however, must be provided on the block dimensions of the chip and whether the chip was filled by row major or column major. In some cases this data may not be known or may be unclear. These functions allow the user to view before correction chip images with grid markings and where the blocks are located according to the information provided. If there were mistakes with the loadin the plots will be problematic.

getBlockMaps will use an aCGH or aCGHroster object and loop over any number of given samples.It is a wrapper for makeBlcokMap.images.

makeBlockMap.image will output a plot for a single sample. It utilizes labelBlocks.

labelBlocks will print block labels for a given array.image object assuming block information has been saved.

Value

Either a chip image of the before correction chip image with grid lines and labels for blocks or a file containing multiple versions of this plot for any number of specified samples.

Note

array image files for samples must have been created. The files must have had block information saved. Smooth2D must also have been performed

Author(s)

Lori Shepherd

References

See Also

`create.roster`, `make.aCGH`, `create.array.images`,`image.acgh.chip`

Examples

```
# the provided example will not work properly as they do not contain
# block information in the flat file

# these function will work with online example set 1
```

`getFailed` *DETERMINES (AND REMOVES) SAMPLES THAT ARE MISSING ALL OF THEIR (log2) DATA*

Description

Finds samples that have failed for one reason or another and subsets them out of aCGH object

Usage

```
getFailed(aCGH,
          removeFlag = TRUE,
          vrb =TRUE,
           ...)
```

Arguments

`aCGH`	an aCGH object
`removeFlag`	logical if failed samples should be removed from the aCGH object
`vrb`	logical indicating if status messages should be printed
`...`	additional arguments for the subsetACGH function

Details

This function is designed as a housekeeper. During the load-in process and the processing(smoothing/CBS), place holders for the samples may be included; this leads to full columns of missing(NA) data for samples' log ratios in the aCGH object. There are certain functions and applications that will not work correctly if there is missing data. There is a function flankNA.CBS that will fill in missing values, however, if an entire column is NA it will not fill these values in and NAs will occur. This function works by finding the full columns of NAs. These columns are flagged as failures. If removeFlag is tripped these samples will be exclude from the aCGH object using the subset function. An aCGH object of 'okay' samples is returned. If removeFlag is not tripped a new aCGH object is not created, instead the logical vector indicating which samples were okay (T) and which had NA columns (F) is returned.

Value

if removeFlag, an aCGH object of samples that did not fail if not removeFlag, a logical vector indicating which samples failed and which are okay

Note

Author(s)

Lori Shepherd

References

See Also

`subsetACGH`, `flankNA.CBS`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# return aCGH object
aCGH = getFailed(aCGH, removeFlag=TRUE)

#or

#returns logical vector
whichFailed = getFailed(aCGH, removeFlag=FALSE)
```

`gethighlight` *CREATE COLOR VECTOR HIGHLIGHTING POINTS OF FLAGGED SPOTS*

Description

This function will create a color shceme to highlight values of flagged spot.IDs. It is used with a single row genomic plot created by InitGOR

Usage

```
gethighlight(column,
             GORplt,
             clrs = c("blue", "green", "purple"),
             normal = 0, ncol="black" )
```

Arguments

`column`	vector of length equal to the total number of spots in the aCGH object. This vector contains values that flag specific spots.
`GORplt`	a plotting object created by the InitGOR call
`clrs`	a list of colors that will be used as the highlight color. Each different flag in the column vector will recieve a different color. If more flags occur that colors specified, colors will be repeated.
`normal`	Teh value in the vector of columns that is considered normal or unflagged.
`ncol`	the default color for normal values

Details

gethighlight was created as a helper function that is internally called by pointsGOR. There should be no reason to call gethighlight directly; pointsGOR does require specifing the arguments to be passed into the gethighlight function. For this reason we will discuss the function and arguments further.

Column is a vector of flags, flagging spots used on the chip. It will be of length equal to the number of spots used on the chip NOT the index/number of spots for a particular plot. There may be a number of different flags. Each flag will be given a color based on the argument clrs. This will be the color of the point on the plot created at any spot that is of that type of flag. We assume that normals are given some common value; this value should be specified as normal. For example: say we gave a column with values 0=normal, 1=QC flag, 2=failure. normal would equal 0 (normal=0). any spots flaged with a 1 will have a color of blue and any spots flagged with a 2 will have a color of green. If ncol is specified normals will be of that color, the default is black.

Value

a vector of length equal to the number of spots used in the aCGH object. It will contain the colors for each spot.

Note

Used by pointsGOR

Author(s)

Lori Shepherd

References

See Also

pointsGOR, InitGOR

Examples

```
#
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


GORplt = InitGOR(aCGH)
cc = gethighlight(aCGH$mapping.info$map.flag, GORplt)

# The pointsGOR function automatically calls this function and subsets appropriatly

graphics.off()
```

---

getLGRraw *GET LOG2 RATIOS*

Description

This functions loads an array image file, gets the signal values, and computes a log2 ratio. The log2 ratios are returned as a vector.

Usage

```
getLGRraw(fname.array.images,
                  Tmat="array.images$matrix$cy5$Signal.Mean",
                  Cmat="array.images$matrix$cy3$Signal.Mean",
                  flag.mat=NA
                  )
```

Arguments

`fname.array.images`	array image file name/path to load
`Tmat`	path to tumor signal value
`Cmat`	path to control signal value
`flag.mat`	logical indicating values to exclude, T means exclude, this is passed in to image2vec call

Details

Value

LGR a vector of log2 ratios

Note

Author(s)

Lori Shepherd

References

See Also

`create.array.images`

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# this builds a map build
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
        map.label="HB19Kv2.HG18",
       spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
       loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
       Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
           "chr7","chr8","chr9","chr10","chr11","chr12",
           "chr13","chr14","chr15","chr16","chr17","chr18",
           "chr19","chr20","chr21","chr22","chrX","chrY"),
       chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
       map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
       rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
       YchromNum = 24, centromere.file =  "Centromeres.txt",
       centromere.loc.lbl="location", centromere.file.sep="\t")


# this will build an array image file for sample1
```

```
 cyroot="HB19Kv2-sample1"
 cy3file="arrays/HB19Kv2-sample1_532.txt"
 cy5file="arrays/HB19Kv2-sample1_635.txt"
 cyinfo=list(cyroot=cyroot,cy3file=cy3file,cy5file=cy5file)
 create.array.images(cyroot=cyinfo, load.specs="Imagene.load.specs.csv",
                array.dir="arrays", image.dir="array.images",
                design.name=NA, image.soft=NA, vrb=TRUE)

# with the build file and array image file we now can map sample 1
 map.array.images(
     fname.array.images="array.images/images/HB19Kv2-sample1.RData",
     mapping.data="RData/HB19Kv2.HG18.RData",
     vrb=TRUE)

# loads sample1's array image file and gets raw log2 ratios
file.name="array.images/images/HB19Kv2-sample1.RData"
LGR =  getLGRraw(fname.array.images=file.name)
```

`getMean` *RETRIEVE MEAN VALLUE ACROSS ALL SAMPLES*

Description

This function will take in an aCGH objectand return a vector of the means of sample values for the aCGH object.

Usage

```
getMean(aCGH,
        fitVls,
        rmNA,
        orientation.flipped=1)
```

Arguments

`aCGH`	an aCGH object
`fitVls`	logical indicating if fitted log 2 ratios are used. Default is T
`rmNA`	logical if NA values should be excluded from calculating mean across samples
`orientation.flipped`	value in orientation field of aCGH inventory representing flipped/dye swaps. Any sample with this value for the orientation will have their log2 ratios multipled by a -1 for analysis

Details

This function will take in an aCGH object. It will get an average value at each spot location of either the log2 ratios or the smoothed log2 ratios depending on fitVls. If rmNA is T all NA values are excluded from the analysis. If there are dye swaps/orientation flips, samples will have log ratios opposite than what is normal. To correct for this effect we multiple a -1 on all sample values whose orientation suggests flipped.

Value

a vector of mean values for each spot of an aCGH object

Note

Author(s)

Lori Shepherd

References

See Also

`meanPlot`, `plotMean.vs.freq`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


yvec = getMean(aCGH, fitVls=TRUE, rmNA=TRUE)

# to graph

makeMeanGraph(aCGH, yData=yvec, clr="blue")



graphics.off()
```

---

getNmat *EVALUATE A MATRIX?*

Description

Usage

```
getNmat(Icol,Irow,ncol,nrow)
```

Arguments

`Icol`
`Irow`
`ncol`
`nrow`

Details

Value

Note

uses Fortran code

Author(s)

Lori Shepherd

References

See Also

Examples

`GetSNR` *CALCULATE SIGNAL TO NOISE*

Description

This function caluculates the signal to noise ratio for each sample.

Usage

```
GetSNR(aCGH)
```

Arguments

`aCGH`	an aCGH object

Details

This function gets the median of the difference of log2 ratios and log2 ratios fitted as a noise calculation. The signal calculation is obtained by the absolute value of the median of log2 ratios on the X chromosome. A Signal to noise ratio is also calculated. These values are stored in the aCGH object's inventory as SNR (signal to noise), SNR.Signal (signal calculation), and SNR.Noise (noise calculation)

Value

an aCGH object with updated inventory entries

Note

This function assumes circulary binary segmentation has been performed and that log2.ratios were not overwritten with fitted values.

It is recommended that the aCGH object is recentered before utilizing this function

Author(s)

Lori Shepherd

References

See Also

SNR.interactive

Examples

```
load("RData/aCGH.RData")
aCGH = GetSNR(aCGH)
```

GLADwrapper *CONVERTS ACGH TO GLAD OBJECT*

Description

This function will take in an aCGH object and return a list of GLAD objects that can be used with GLAD functions

Usage

```
aCGHtoGLAD(aCGH,
           ismpl=1,
           fitted=TRUE,
           vrb=TRUE,
           profileBysub=TRUE,
           performGlad=TRUE,
            ...)
```

Arguments

aCGH	an aCGH object
ismpl	number of the sample to use. single value
fitted	logical if fitted or raw log2 ratios should be used
vrb	logical indicating if status messages should be printed
profileBysub	logical indicating if profileCGH object should be made with a glad function call or by subsetting the aCGH object. Default subsets the aCGH object
performGlad	logical if glad function should be run to get a profileCGH object and glad object
...	additional arguments to the glad function call

Details

This function will take in an aCGH object and make a glad profileCGH object, as well as a glad object if glad is performed. There are two options for making a profileCGH object, althrough they both yield the same output, we left it up to the user to decide if the object should be made purely on current aCGH values or if a call to glad's as.profile should be used. The user may or may not want glad to be run on the profileCGH object. This is controlled by the function's performGlad argument.

Currently the function does not make an arrayCGH object. This is coming soon.

Value

an list object containing profileCGH object, a glad object if glad was performed, and the information for ismpl and fitted.

Note

This function currently does not make an arrayCGH object. It is coming soon

Author(s)

Lori Shepherd

References

See Also

makeGLADplts

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

library(GLAD)
```

```
GLADobj = aCGHtoGLAD(aCGH)

#or

GLADobj2 = aCGHtoGLAD(aCGH, ismpl=3, fitted=FALSE, profileBysub=FALSE)
```

`graphPCA` *creates 3D PCA Viewer*

Description

takes in an aCGH object and associated principle component analysis information and factor information to create a 3D PCA Viewer

Usage

```
graphPCA(aCGH,
       PCtest,
       factorInfo,
       interactive = TRUE)
```

Arguments

`aCGH`	An aCGH object
`PCtest`	object containing principle component analysis data
`factorInfo`	object containing factor information
`interactive`	logical if window display choice actions for interactive graph version

Details

`graphPCA` takes in a specific aCGH object and associated principle component anaylsis and factor information. The application uses this information to create a 3D PCA Viewer to interrogate sample data. It creates a plotInfo object containing information about graphical devices and sample colors.

plotInfo has two slots: clrs and deviceInfo clrs is a vector of clrs to use for sample coloring based on factoring deviceInfo contains the names of all open windows and there index

Value

plotInfo

Note

an interactive version of these graphs may be created with `runPCAfile`

In order for this to run properly, `CBS.Batch`, `flankNA.CBS`, `PCAfile`, and initFactor or `factorObj` should all be run

Author(s)

Lori Shepherd

See Also

runPCAFile, PCAfile, factorObj, factorObj, rgl

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


# it is important to note here that the example aCGH object
# has already had CBS performed on it. CBS must be performed to
# use this function
# see ?CBSBatch , ?flankNA.CBS

#remove NA columns
aCGH = getFailed(aCGH)

# make PCA file
makePCAfile(aCGH)

# load PCA information
load("RData/PCAtest.RData")

# make Factor object
factorInfo=initFactor(aCGH,smpldx=sampleInfo$smpldx,factorList=c("sex", "orientation", NA

plotInfo = graphPCA(aCGH, PCtest, factorInfo, interactive=FALSE)


graphics.off()
rgl.close()
```

HB19Kv2.design *EXAMPLE CHIP DESIGN FILE*

Description

This data gives the deisgn of the chip used. In must contain a MasterRow and MasterCol column, which is followed by other design variable.

Format

This file must be a comma delimited file with a single header line containing the names of the columns.

Details

This design file is for the HB19K chip. It contains the following columns: MasterRow, MasterCol, Pin, Plate, Platerow, Platecol, Repetition, and Time.

Note

The file must contain columns for MasterRow and MasterCol

Source

References

See Also

`Write.aCGH.ex1`, `design.array.images`

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# to see the first ten columns of the design file provided
dd = read.table("HB19Kv2.design", sep=",")
dd[1:10,]
```

HB19Kv2.HG18.map *EXAMPLE MAPPING INFORMATION FOR HB19K$_v$2$_H$G18*

Description

Information for the HB19K chip design

Format

A comma seperated table with the following columns: Clone, Chromosome, start, Stop, Center, g$_l$$oc, Band, Mappedby, and$

Details

This type of file must contain information similar to the columns: Clone, Chromosome, start, Stop, Center, and Band.

Note

Source

References

See Also

Write.aCGH.ex1, buildMap

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# to see the first ten columns of the map file provided
dd = read.table("HB19Kv2.HG18.map.csv", sep=",")
dd[1:10,]
```

HBdesign.inventory *ASSIGNS DEFAULT DESIGN NAME*

Description

This function is a RPCI specific function to assign default design name by subsetting the chip image name

Usage

```
HBdesign.inventory(fname.input,
                           input.inv.sep=","
                  )
```

Arguments

fname.input	inventory file to be updated with design name
input.inv.sep	character that separates fields in the provided inventory, will be passed as sep to a read.table call

Details

Value

The inventory file passed in as an argument will be overwritten with the updated inventory file

Note

Author(s)

Lori Shepherd

References

See Also

`add.inventory`, `convert.old.inv.R`

Examples

`image2vec` *CONVERT IMAGE MATRIX TO VECTOR*

Description

This function takes in a matrix associated with a specified image mapping. It will use the mapping to convert the matrix into a vector.

Usage

```
image2vec(img.mat,
                map.images,
                min.rep=1,
                flag.mat=NA)
```

Arguments

`img.mat`	matrix to map to a vector
`map.images`	associated map image file
`min.rep`	number of replicates to include
`flag.mat`	logical indicating values to be excluded, T means to exclude

Details

This will take a matrix associated with an array image and use that array image's map to convert the matrix into a vector.

Value

out a matrix of values

Note

Creation of an array image object and mapping of that array image must have occurred.

Author(s)

Lori Shepherd

References

See Also

map.array.images

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# this builds a map build
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
        map.label="HB19Kv2.HG18",
       spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
       loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
       Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
           "chr7","chr8","chr9","chr10","chr11","chr12",
           "chr13","chr14","chr15","chr16","chr17","chr18",
           "chr19","chr20","chr21","chr22","chrX","chrY"),
       chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
       map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
       rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
       YchromNum = 24, centromere.file =  "Centromeres.txt",
       centromere.loc.lbl="location", centromere.file.sep="\t")

# this will build an array image file for sample1
  cyroot="HB19Kv2-sample1"
  cy3file="arrays/HB19Kv2-sample1_532.txt"
  cy5file="arrays/HB19Kv2-sample1_635.txt"
  cyinfo=list(cyroot=cyroot,cy3file=cy3file,cy5file=cy5file)
  create.array.images(cyroot=cyinfo, load.specs="Imagene.load.specs.csv",
                  array.dir="arrays", image.dir="array.images",
                  design.name=NA, image.soft=NA, vrb=TRUE)

 # with the build file and array image file we now can map sample 1
  map.array.images(
      fname.array.images="array.images/images/HB19Kv2-sample1.RData",
      mapping.data="RData/HB19Kv2.HG18.RData",
      vrb=TRUE)

  # load in sample1's cy5 signal mean matrix
  load("array.images/images/HB19Kv2-sample1.RData")
  mat = array.images$matrix$cy5$Signal.Mean

  # load in associated array image map
  mapName = array.images$info$full.map.name
  load(mapName)

  # creates a vector version of the signal mean matrix
  sigMean = image2vec(img.mat=mat, map.images=map.images, min.rep=1, flag.mat=NA)
```

---

image.aCGH *MAKES CHIP IMAGES*

Description

These functions visualize chip images.

Usage

```
image.acgh.chip(chipMat,gridMat=NA,FlagMat=NA,autoFlag=NA,
                useRanks=TRUE,ttl="title",xlim=NA,ylim=NA,...)

image.acgh.rlgnd(chipMat,gridMat=NA,FlagMat=NA,autoFlag=NA,
                 ttl="title",xlim=NA,ylim=NA,...)

Timage.acgh.rlgnd(chipMat,gridMat=NA,FlagMat=NA,autoFlag=NA,
                  ttlA="titleA",ttlB="titleB",xlim=NA,ylim=NA,
                  mirrorFlag=FALSE,...)

image.acgh.lgnd(chipMat,gridMat=NA,FlagMat=NA,autoFlag=NA,
                ttl="title",xlim=NA,ylim=NA,...)
```

Arguments

`chipMat`	matrix of values to be converted into a chip image
`gridMat`	matrix used to create yellow grid lines on chip images, if NA no grid will be plotted
`FlagMat`	matrix or vector indicating spots not to be included; utilizes the rule of which FlagMat != 0; if NA all spots are used
`autoFlag`	matrix or vector indicating spots not to be included; utilizes the rule of which autoMat == 0; if NA all spots are used
`useRanks`	logical indicating if rank should be used on `chipMat` values before image is created
`ttl`	title of chip image
`xlim`	x axis limits, if NA (default) will use the dimension of the `chipMat`
`ylim`	y axis limits, if NA (default) will use the dimension of the `chipMat`
`ttlA`	title of chip image
`ttlB`	title of histogram graph
`mirrorFlag`	logical indicating if the transpose of the mirror image should be used
`...`	additional arguments

Details

These functions are designed to produce visualization of chip images from matricies of values using red/green to represent flourescence.

Value

visualization of chip images

Note

Author(s)

Lori Shepherd

References

See Also

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

load(aCGH$data.info$fname.array.images[1])

chipMat = array.images$matrix$cy3$Signal.Mean
gridMat = array.images$matrix$Grid
exclude.rule="array.images$matrix$cy3$Flag!=0"
eval.js(paste("excludeMat=",exclude.rule))
ttl = "cy3"

image.acgh.rlgnd(chipMat=chipMat,gridMat=gridMat,
                      FlagMat=excludeMat,
                      autoFlag=NA,useRanks=TRUE,
                      ttl=ttl)

image.acgh.chip(chipMat=chipMat, ttl = ttl)

Timage.acgh.rlgnd(chipMat=chipMat, ttlA=ttl)



graphics.off()
```

Imagene.load.specs *EXAMPLE IMAGENE SPECS*

Description

This data gives a sample spec file for Imagene software output

Format

A comma delimited table file containing a columns for what names should store data and a column for what the header column name is for associated data in the imagene file.

Details

All files of this type must contain a numfile field and a cy.suffix (cy5.suffix,cy3.suffix). Columns recognizing spot.ID, Row, Column, cy3/cy5 background mean, cy3/cy5 signal mean, cy3/cy5 standard deviations are included. If the user wishes to store additional information from the raw data files, they must specify what they are to be called/where they are to be stored in the new object as well as what column gives reference to the data in the file.

Note

The file must contain a fields numfile, cy3.suffix and cy5.suffix

Source

References

See Also

`Write.aCGH.ex1`, `create.array.images`

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# to see the example spec data we have provided
read.table("Imagene.load.specs.csv", sep=",")
```

---

initFactor *performs factoring/leveling and store useful information in factorInfo object*

Description

This function takes in an aCGH object and a list of factors for leveling the aCGH's inventory. It stores all information in a factorInfo object

Usage

```
initFactor(aCGH,
       smpldx = NA,
       factorList = NA)
```

Arguments

aCGH	An aCGH object
smpldx	a vector indicating which samples to use in calculation. The defualt uses all samples in the aCGH object
factorList	a numeric or character vector associated with the names of aCGH inventory to be used as factors. The default uses all the inventory names

Details

initFactor takes in a specific aCGH object to factor inventory. The user may indicate which samples within the aCGH to factor. The default uses all aCGH samples. FactorList is a numeric or character vector containing the index of names or names of desired factors. This is used to cycle through different factor types for comparison. The defualt will create a list of all inventory names. All factor information is stored in a factorInfo object. This includes: ntype, Ftype, iFtype, facIdx, factorList, listType, and listInd. facIdx is the index of the factor in the aCGH inventory Ftype is vector of the samples factored by the current factor iFtype is a numeric version of Ftype ntype is the total number of different factors of the current factor type factorList is the complete list of factors to cycle through listType indicates if the factorList is of type character or numeric listInd is the current location of the factor type in factorList

Value

factorInfo

Note

calls factorObj

Author(s)

Lori Shepherd

See Also

factorObj, runPCAFile, choiceAct

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

iniFac = initFactor(aCGH)

## or give a factor list

iniFac2 = initFactor(aCGH, factorList = c("sex", "orientation", "User",
NA))
## The above may also have been run by:
#   initFactor(aCGH, factorLis = c(8,9,10,NA))
```

`internalDataSets` *INTERNAL DATASETS*

Description

These objects and datasets are used by the writeExample functions. They are never seen or used by the user.

Format

Details

The object data.list contains a list of all the files created during the Write.aCGH.ex1 function execution. All the data of the files are kept in this list.

fnames and nfiles are helper objects that aided in the construction of ex2.RData.full.names and ex2.RData.names. fnames is a list containing the names of all .RData objects that should be created when executing Write.aCGH.ex2. This includes all objects in the array.images/ subdirectories and the RData directory. nfiles is the length of the number of .RData objects that will be created. ex2.RData.full.names is a list of the paths of the objects where the package example was created. This gave the path of the object that should have been copied over to the package. ex2.RData.names is a list containing the names of the objects that should have been created in the package.

Note

These objects are never seen by the user. They are helper objects containing data that is given when writing the package examples.

Source

References

See Also

`writeExample`

Examples

`internalFunctions` *INTERNAL FUNCTIONS*

Description

These are internal functions never called by the user.

Usage

Arguments

Details

WriteExample(data.list,target.dir,vrb=TRUE) used by Write.aCGH.ex1 and Write.aCGH.ex2

check.V1(i,V1,mtch) used by create.array.images

choiceAct(aCGH, PCtest, sampleInfo,factorInfo, plotInfo, pt2, gen, heat) used by selectPt - PCA Viewer

rosterWrapismpl used by Roster.papply

smoothWrap(ismpl) used by smooth2D.papply

cbsWrap(ismpl) used by CBS.papply

XportWrap(j) used by ExportSamples.papply

QC1Wrap(j) used by QCreport1.papply

cntOut(i) used by QCreport1.papply

makeSmooth2D() used by DiagPlot.Smooth2D

makeHeat(add=FALSE) used by DiagPlot.Smooth2D

makeChrom() used by DiagPlot.Smooth2D

makeMessage() used by DiagPlot.Smooth2D

makeWait() used by DiagPlot.Smooth2D

makeMenu() used by DiagPlot.Smooth2D

doClicks(tpnt,npnts) used by DiagPlot.Smooth2D

GetLGR(j) used by getLGRreps

CountSegs(i,iab,SegMat) used by armMask

Value

Note

Author(s)

Lori Shpeherd

References

See Also

`WriteExample`, `choiceAct`

Examples

`loadPCAInv` *loads existing aCGH with PCA inventory to interactive 3D PCA Viewer*

Description

This application loads existing aCGH objects with associated PCA inventory files and creates an interactive 3D PCA graph of samples

Usage

```
loadPCAInv(aCGHfileName,
       invFileName = NA)
```

Arguments

`aCGHfileName`	The name of the aCGH file containing aCGH object
`invFileName`	The name of the PCA inventory file to be loaded. The default is the last saved PCA inventory for the aCGH object

Details

`loadPCAInv` loads the specified aCGH file. If an inventory file name is given, the data from that identified saved session is restored. The default will load the last saved inventory file.

Value

Creates a 3D PCA graph. Inventory files and aCGH files created.

Author(s)

Lori Shepherd

See Also

`runPCA`, `makePCAfile`, `initFactor`, `makeInvDir`, `graphPCA`, `selectPt`

Examples

```
### This assumes interactive 3D viewer already has been run
###  see runPCAFile


# because this is also interactive it must be commented out

# loadPCAInv("PCAtest")
```

`make.aCGH` *MAKE aCGH OBJECT*

Description

This function subsets,if necessary, an aCGHroster object to create aCGH object

Usage

```
make.aCGH(aCGHroster,
                  sampleDX=NA,
                  mapping.data="RData/HB19Kv2.HG18.RData",
                  LGR.label="array.images$smooth2D$Mxy.default",
                  Flagmat.label=NA,
                  min.rep=1,
                  vrb=TRUE,
                  saveFlag=FALSE,
                  saveName="RData/aCGH.RData")
```

Arguments

`aCGHroster`	an aCGHroster object
`sampleDX`	index of samples to use from the aCGHroster object, subsetting index. If NA (default), all samples are used
`mapping.data`	character name mapping.info build file or list of mapping.info objects. The default uses the map built as part of the example dataset.
`LGR.label`	path/name of matrix storing the [smoothed] log2 ratios
`Flagmat.label`	path/name of the matrix storing flag spots for exclusion
`min.rep`	minimum number of spot replicates required
`vrb`	a logical flag indicating if status messages should be printed
`saveFlag`	logical if the aCGH object should be saved
`saveName`	name to save aCGH object if saveFlag

Details

The `aCGHroster` object is used to create an aCGH object.

The map data is loaded using the `mapping.data` information. The `mapping.data` argument may be a list or a character. If the mapping.data argument is a list then we assume that it has elements: [[1]] mapping.info , [[2]] Band.Aid. If the mapping data is a character then we assume that is a filename corresponding to an .RData file containing a mapping.info variable with the same name and a Band.Aid object

A list containing aCGHroster data, log2 ratios, and inventory is then created. The data is subset based on the sampleDX given. For each sample desired, the mapping data is loaded and checked, as well as the log2 ratios.

Value

returns an aCGH object

Note

The aCGHroster object must have an inventory file in order to make an aCGH object; see `add.inventory`.

Author(s)

Lori Shepherd

References

See Also

`create.roster`,`add.inventory`

Examples

```
library(aCGHplus)

# creates directories and files needed for running example
Write.aCGH.ex1()
Write.aCGH.ex2()

# a build file for each of the chip designs used in analyzing the
# images must be built. For our example this is the HB19K_v2_HG18
# this builds that mapping information file
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
        map.label="HB19Kv2.HG18",
       spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
       loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
       Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
           "chr7","chr8","chr9","chr10","chr11","chr12",
           "chr13","chr14","chr15","chr16","chr17","chr18",
           "chr19","chr20","chr21","chr22","chrX","chrY"),
       chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
       map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
       rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
```

```
        YchromNum = 24, centromere.file =  "Centromeres.txt",
        centromere.loc.lbl="location", centromere.file.sep="\t")

# we now load the provided aCGHroster object. This assumes the
# aCGHroster object already be created.
# see   create.roster

load("RData/aCGHroster.RData")

# The following would creates an aCGH object using all the samples
aCGH = make.aCGH(aCGHroster,
                 sampleDX=NA,
                 mapping.data="RData/HB19Kv2.HG18.RData",
                 LGR.label="array.images$smooth2D$Mxy.default",
                 min.rep=1,
                 vrb=TRUE)

# note the above could have been run as aCGH = make.aCGH(aCGHroster)

# this creates an aCGH object off the aCGHroster object
# This will subset out the first and third sample of the aCGHroster object

aCGH =  make.aCGH(aCGHroster,
                 sampleDX=c(1,3))
```

`makeFlank` *MAKES GRAPH OF FLANKING POINTS AROUND SELECTED SPOT*

Description

This function was designed as a helper function for cmean and meamPlot. It will take information about a selected point and graph the flanking points. makeFactorObject must have been run.

Usage

```
makeFlank(aCGH,
          loc,
          bacwin,
          fitVls,
          grp.lbls,
          mydev=NA)
```

Arguments

`aCGH`	an aCGH object
`loc`	numeric indicating which BAC is selected
`bacwin`	numeric value indicating the number of bacwins (+/-) that will be shown on subsequent graph of surrounding area of a selected point

fitVls	logical value indicating if the fitted log2 ratios are used or the raw log 2 ratios are used initially in graph
grp.lbls	character vector of factor objects (see makeFactorObject)
mydev	name or number of device to plot to; default opens new X11 window

Details

makeFlank was designed as part of meanPlot to display flanking points of a selected spot/BAC; It can however be used on its own. makeFactorObject must have been run. The function will take the index of a spot (loc) and graph +/- (bacwin) for all samples in the aCGH object. The points of the graph are color coded according to their factor type (grp.lbls). The user may change whether fit or raw log2 ratios are used by (fitVls). The function is set up with a default of 6 different colors for factor types. If more are used the colors will repeat. Faded colors indicate flanking values/samples while bold colors indicate the selected point for all samples (loc). The plotting character is a '+' for real values and 'o' for values that are missing/estimated.

Value

a graph of falnking spots to a selected spot

Note

called by cmean and meanPlot

Author(s)

Lori Shepherd

References

See Also

cmean,meanPlot

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# runs makeFactor Object

  aCGH = makeFactorObject(aCGHloaded=TRUE, invloaded = TRUE, factorBy="sex")

  # loads aCGH and grp.lbls from makeFactorObject
  load("RData/vizMean.RData")

  makeFlank(aCGH, loc=1000, bacwin=5, fitVls=FALSE, grp.lbls=grp.lbls)
```

```
graphics.off()
```

`makeGenome3Row` *MAKE GENOMIC PLOTS*

Description

This function will create a file of all specified samples' genomic profiles.

Usage

```
makeGenome3Row(aCGH,
               smpls = NA,
               fileName = "Genome3Row",
               pdfFlag = FALSE,
               fitFlag=TRUE,
                ... )
```

Arguments

`aCGH`	an aCGH object
`smpls`	numeric list corresponding to the samples in the aCGH object. Genomic plots of these samples will be genreated. NA(default) will plot alls amples.
`fileName`	name of file to save plots
`pdfFlag`	logical indicating if the postscript file should be converted to pdf
`fitFlag`	logical indicating if the fit values should also be drawn on the graph. This plotting function graphs raw log 2 ratios by default.
`...`	Any additional arguements to be passed into the plotGenome call

Details

This function will create genomic plots for all specified samples of an aCGH object. If no samples are specified genomic plots for all samples are generated. The raw log2 ratios are used for the graphs. If fitFlag is tripped the fit log2 ratios, if applicable, are also plotted for every graph.

This function automatically saves all graphs to a file, fileName, in the plots directory. The format is a postscript file. The user may specify pdfFlag if the postscript file should be converted to a pdf after generation.

Value

A file in the plots directory that has all specified samples' genomic plots

Note

This function uses the function plotGenome to graph samples

Author(s)

Lori Shepherd

References

See Also

`plotGenome`, `create.GenomeOneRow`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# graphs all samples
makeGenome3Row(aCGH)

# or

# graph the first 2 samples
makeGenome3Row(aCGH, smpls=1:2)
```

`makeGLADplts` *MAKES GLAD PLOTS*

Description

This function is used with the aCGHtoGLAD function. It will make profile plots using GLAD's plotProfile call.

Usage

```
makeGLADplts(GLADobj,
             chrmList=1:24,
             saveToFile=TRUE,
             fileName="GLADplots",
             vrb=TRUE,
             ...)
```

Arguments

GLADobj	output from aCGHtoGLAD
chrmList	List of individual chromosomes to plot. This may be left NA to only graph over the entire genome
saveToFile	logical if plots should be saved to a file or should appear in current window
fileName	path name to save graphs if saveToFile
vrb	logical indicating if status messages should be printed
...	additional arguments to glad's plotProfile call

Details

This function was made as a convience function to graph genomic profiles using glad objects and glad's plorProfile function.

It will graph the entire genome and any chromosome listed in chrmList.

Value

graphs of profile

Note

Author(s)

Lori Shepherd

See Also

GLADwrapper

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()

load("RData/aCGH.RData")
library(aCGHplus)
library(GLAD)

GLADobj = aCGHtoGLAD(aCGH)

makeGLADplts(GLADobj, nit=3, Bkp=TRUE, labels=TRUE,Smoothing="Smoothing", plotband=FALSE)
```

`makeHeatmaps` *MAKE HEATMAPS*

Description

This function will create a file containing all selected samples' heatmaps

Usage

```
makeHeatmaps(aCGH,
             smpls = NA,
             type="correction",
             fileName = "Heatmaps",
             pdfFlag = FALSE )
```

Arguments

`aCGH`	an aCGH object
`smpls`	numeric list indicating samples of the aCGH object
`type`	type of heatmaps to generate. May be "background", "correction" or "mean". default is correction
`fileName`	name of file to save plots
`pdfFlag`	logical indicating if postscript file should be converted to a pdf

Details

This is a function to generate a file containing samples' heatmaps. Two heatmaps are generated per page. Each page is a sample. The pairing depends on the type of heatmap selected. There are three options: "correction" will give the before and after correction slides, "background" will give the cy3/cy5 intensity backgrounds, and "mean" will give the cy3/cy5 intensity. If anything else is specified the function will default to mean.

The file is generated in the plot directory as a postscript file. If pdfFlag is tripped the postscript file will also be converted to a pdf.

Value

file in the plot directory containing graphs of all selected samples' heatmaps

Note

utilizes image.acgh.chip function

Author(s)

Lori Shepherd

References

See Also

image.aCGH

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# makes "correction" heatmaps for all samples
makeHeatmaps(aCGH)

# or
#makes "mean"(cy3/cy5) for first 2 samples
makeHeatmaps(aCGH, smpls=c(1:2), type="mean")
```

---

makeInvDir *creats directory PCA.inventory*

Description

creats needed directory PCA.inventory to store inventory files created while utilizing 3D PCA Viewer

Usage

```
makeInvDir()
```

Arguments

Details

`makeInvDir` is a function that checks to see if the needed directory PCA.inventory is created to store inventory files while utilizing the 3D PCA Viewer

Value

creates the directory PCA.inventory in current directory

Note

This is called within runPCA

Author(s)

Lori Shepherd

See Also

runPCA, addPCAinv

Examples

```
makeInvDir()
```

---

make.map.images *MAKES AND STORES MAP.IMAGES*

Description

This function uses an array image file, mapping file and information concerning image software and chip design to map a sample

Usage

```
make.map.images(array.images,
                mapping.info,
                Band.Aid,
                mapping.info.name,
                design.name,
                image.soft,
                map.name,
                full.map.name,
                returnFlag=FALSE,
                saveFlag=TRUE,
                vrb=TRUE)
```

Arguments

array.images	an array.image object
mapping.info	A mapping.info object
Band.Aid	A Band.Aid object
mapping.info.name	name of the mapping.info object
design.name	array.image design name
image.soft	array.image image software name
map.name	unique name for mapping using mapping.info.name, design.name, and image.soft names
full.map.name	name to be used for the mapping file, suggested is map.name with .RData extension
returnFlag	flag to return the map.image object once it is created. Default is false
saveFlag	flag to save the map.image object once it is created. Default is true
vrb	logical flag indicating if status messages should be printed

Details

This function loads a mapping information file and a sample's array image file and performs a mapping of spot.IDs. Information on the mapping and four additional matrices are stored. A mapping from the matrices to the mapping object, a mapping from the mapping object to the matrices, a column matrix, and a row matrix are the four additional matrices created. The object is stored in the image.dir directory specified when creating array images in a subdirectory maps if `saveFlag` is true. The map.image object is returned if `returnFlag` is true.

Value

If returnFlag is true, returns map.image object. A list containing software and chip information, mappings, useful indices,spot.IDs, mapping.info object, and Band.Aid object.

Note

Used by map.array.images, user should not have to call this function

Author(s)

Lori Shepherd

References

See Also

`map.array.images`, `create.array.images`, `buildMap`

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

# a mapping information file must have been built
# this will build a mapping information file
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
        map.label="HB19Kv2.HG18",
       spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
       loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
       Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
           "chr7","chr8","chr9","chr10","chr11","chr12",
           "chr13","chr14","chr15","chr16","chr17","chr18",
           "chr19","chr20","chr21","chr22","chrX","chrY"),
       chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
       map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
       rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
       YchromNum = 24, centromere.file =  "Centromeres.txt",
       centromere.loc.lbl="location", centromere.file.sep="\t")

# load mapping data
load("RData/HB19Kv2.HG18.RData")

# create array image for sample1
```

```
cyinfo="HB19Kv2-sample1"
create.array.images(cyroot=cyinfo, load.specs="Imagene.load.specs.csv",
                  array.dir="arrays", image.dir="array.images",
                  design.name=NA, image.soft=NA, vrb=TRUE)

# load array image for sample 1
load("array.images/images/HB19Kv2-sample1.RData")

design.name=array.images$info$design.name
image.soft=array.images$info$image.soft
if(is.na(design.name)) design.name=array.images$info$cyinfo$cyroot
if(is.na(image.soft)) image.soft=array.images$info$cyinfo$cyroot

# hard coded = mapping info name for these samples HB19Kv2.HG18
mapping.info.name = "HB19Kv2.HG18"


map.name=paste(mapping.info.name,"\_",design.name,"\_",image.soft,sep="")
full.map.name=paste(array.images$info$image.dir,"/maps/",map.name,".RData",sep="")

 make.map.images(array.images,
                mapping.info= HB19Kv2.HG18,
                Band.Aid=Band.Aid,
                mapping.info.name=mapping.info.name,
                design.name=design.name,
                image.soft=image.soft,
                map.name=map.name,
                full.map.name=full.map.name,
                returnFlag=FALSE,
                saveFlag=TRUE,
                vrb=TRUE)
```

`makeMeanGraph` *MAKES A MEAN PLOT GIVEN AN aCGH OBJECT AND Y-VALUES*

Description

This function takes in an aCGH object and graphs the means across samples.

Usage

```
makeMeanGraph(aCGH,
              yData,
              mydev = NA,
              clr = "red")
```

Arguments

`aCGH`	an aCGH object
`yData`	a vector of y values to plot
`mydev`	number or name of the device to plot to. Default, NA, will open a new window
`clr`	color of the points to plot

Details

This function should be used in correlation with the getMean function. It takes in an aCGH object and a vector of mean values (see `getMean`) and creates the genomic plot.

Value

a plot of the mean across samples

Note

used with getMean function. This function is called by cmean and makeMeanPlot

Author(s)

Lori Shepherd

References

See Also

`cmean`, `meanPlot`, `makeMeanPlot`, `getMean`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

yvec = getMean(aCGH, fitVls=TRUE, rmNA=TRUE)

# to graph

makeMeanGraph(aCGH, yData=yvec, clr="blue")


graphics.off()
```

---

makeSample *MAKES GRAPH OF ALL SAMPLES ON A SELECTED SPOT/BAC*

Description

This function was designed as a helper function for cmean and meamPlot. It will take information about a selected point and graph the data for all samples on that selected point. makeFactorObject must have been run.

Usage

```
makeSample(aCGH,
           loc,
           fitVls,
           grp.lbls,
           mydev=NA)
```

Arguments

aCGH	an aCGH object
loc	numeric indicating which BAC is selected
fitVls	logical value indicating if the fitted log2 ratios are used or the raw log 2 ratios are used initially in graph
grp.lbls	character vector of factor objects (see makeFactorObject)
mydev	name or number of device to plot to; default opens new X11 window

Details

`makeSample` was designed as part of meanPlot to display values of samples at a selected spot/BAC; It can however be used on its own. `makeFactorObject` must have been run. The function will take the index of a spot (loc) and graph the log2 ratios for all samples at that location. The points of the graph are color coded according to their factor type (grp.lbls). The user may change whetherfit or raw log2 ratios are used by (fitVls). The function is set up with a default of 6 different colors for factor types. If more are used the colors will repeat. Faded colors indicate missing/estimated values while bold colors indicate real.values. A jitter effect is added to the x axis for better viewing; since all values would have the same x value this attempts to avoid overlapping.

Value

a graph of all the samples of an aCGH object at a specified spot/BAC

Note

called by cmean and meanPlot

Author(s)

Lori Shepherd

References

See Also

`cmean`,`meanPlot`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# runs makeFactor Object

  aCGH = makeFactorObject(aCGHloaded=TRUE, invloaded = TRUE, factorBy="sex")

  # loads aCGH and grp.lbls from makeFactorObject
  load("RData/vizMean.RData")

  makeSample(aCGH, loc=1000, fitVls=FALSE, grp.lbls=grp.lbls)



graphics.off()
```

---

map.array.images *USES MAP FILE TO BUILD MAPS FOR ARRAY.IMAGE FILES*

Description

This function uses an array image file and a mapping file to map a sample

Usage

```
map.array.images(fname.array.images,
                         mapping.data,
                         vrb=TRUE)
```

Arguments

`fname.array.images`	file name for the array.image
`mapping.data`	character name mapping.info build file or list of mapping.info objects. The default uses the map built as part of the example dataset.
`vrb`	logical flag indicating if status messages should be printed

Details

This function loads a mapping information file and a sample's array image file and performs a mapping of spot.IDs. It will determine the image software and design name for the chip used, create a map name, and check to see if the map has already been created. The array.image object is saved with a map.name and full.map.name to the appropriate map file.

The `mapping.data` argument may be a list or a character. If the mapping.data argument is a list then we assume that it has elements: [[1]] mapping.info , [[2]] Band.Aid. If the mapping

data is a character then we assume that is a filename corresponding to an .RData file containing a mapping.info variable with the same name and a Band.Aid object

Value

An array.image object is saved in the array.dir with its associated map file name. Subsequently,a map file stored in the image.dir subdirectory maps is saved with a call to make.map.images

Note

An image file for the sample you wish to map must have already been created. A mapping for the chip used must also have been created.

A call to make.map.images creates the matrices. When this is called, information on the mapping and four additional matrices are stored. A mapping from the matrices to the mapping object, a mapping from the mapping object to the matrices, a column matrix, and a row matrix are the four additional matrices created. The object is stored in the image.dir directory specified when creating array images in a subdirectory maps.

Author(s)

Lori Shepherd

References

See Also

`Write.aCGH.ex1`, `create.array.images`,`buildMap`, `make.map.images`

Examples

```
  library(aCGHplus)
  Write.aCGH.ex1()

# a mapping information file must have been built
# this will build a mapping information file
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
        map.label="HB19Kv2.HG18",
       spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
       loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
       Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
           "chr7","chr8","chr9","chr10","chr11","chr12",
           "chr13","chr14","chr15","chr16","chr17","chr18",
           "chr19","chr20","chr21","chr22","chrX","chrY"),
       chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
       map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
       rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
       YchromNum = 24, centromere.file =  "Centromeres.txt",
       centromere.loc.lbl="location", centromere.file.sep="\t")

# create array images for sample 1
cyinfo="HB19Kv2-sample1"
create.array.images(cyroot=cyinfo, load.specs="Imagene.load.specs.csv",
```

```
                    array.dir="arrays", image.dir="array.images",
                    design.name=NA, image.soft=NA, vrb=TRUE)


  # with the build file and array image file we now can map sample 1
    map.array.images(
        fname.array.images="array.images/images/HB19Kv2-sample1.RData",
        mapping.data="RData/HB19Kv2.HG18.RData",
        vrb=TRUE)
```

`mapByBlock` *CREATES META.ROW AND META.COLUMN FOR MAPPING PURPOSES*

Description

This function creates meta.row and meta.column for mapping images based on block number and the row and column of spots in the given print tip group.

Usage

```
mapByBlock(file,dimen, major="row",Row ="Row",
           Column="Column",Block ="Block")
```

Arguments

`file`	loaded image analysis flat file. Should be of type data.frame
`dimen`	dimensions of the chip by block (how arrayer spots chip).This should be in the form c(row, column).
`major`	either row or column; did the arrrayer spot the blocks in the dimensions, dimen, by row, and therefore row major or by column and therefore column major.
`Row`	column name of the specified file that holds row information. In this case row should be the row dimension of the pin (print tip) group.
`Column`	column name of the specified file that holds column information. In this case column should be the column dimension of the pin (print tip) group
`Block`	name of the column of the specified file that holds block information. In this case, block should be a number from one to max(block)

Details

This package assumes that certain data in a particular format be located in the image analysis flat files. This function aims at extracting and creating such information when it did not exist specifical in the file. Microarray chips are spotted using an arrayer. This arrayer has a group of pins that spot the chip. This creates a gird within a grid environment. The inner grid represents the pins; it will be the dimension of the pin group. This will be referred to as the pin dimensions. The outter grid represents the group of pins as a whole, and where the arrayer spots the chip. We say the pin group as a whole as a 'Block'. This will be referred to as the block dimensions. Therefore each Block of the block dimension, consists of the pin dimension. As a descriptive example, say the array spotted a chip in a 12 x 2 block dimension and had a pin group of dimension 12 x 18. In order to load data

one of three situations need to occur regarding how row and column are listed in the image analysis flat file:

1. meta.row, meta.column, row and column are all present in the file. Meta.row and meta.column refer to the block dimension. The will have numeric values from one to twelve and one to two respectively. row and column, in this case, refer to the pin dimension and will be numbers one to twelve and one to 18 respectively.

2. Row and column occur in the file. Row and column in this case refer to the master row and master column. Master row and master column are with repect to the entire chip. Row would have values from one to (12x12) 144. Column would have values from one to (2x18)36.

3. Row and Column occur in the file representing the pin dimension. A column block indicating which block the spot occurs also is present. Block in our example case would be a number one to 24 (12x2). It is in this third example that we utilize this mapByBlock function.

The mapByBlock function will take a file, the block dimension, in our example case c(12,2), the name of the columns containing row, column and block (see situation three for what row and column should represent in this case), and the name of how the arrayer spotted the blocks ("row" for row major, "col" for column major). The function will calculate the meta.row and meta.column for the file.

Value

a matrix of values representing row, column, meta.row and meta.column.

Note

This function is utilized within the load-in process for creating images. The arguments of this function are therefore located in a sample's load.spec file. (see load.specs web.example for clarification)

Author(s)

Lori Shepherd

References

See Also

web.example for load.specs.

Examples

```
# In order to run this example, please download
#   load.in.example1 from the website

# file = read.table("GSM185333.txt", sep="\t", header=T)

# these example files were created by row major and the
# block dimension was 12 x 6

#output = mapByBlock(file,dimen=c(12,6) , major="row",Row ="Row",
#           Column="Column",Block ="Block")
```

`map.images` *MAPPING OBJECT*

Description

A map.image object is a list containing information about mapping a sample to its chip.

Format

see details

Details

The object structure is as follows (note: the class of the objects are in parenthesis):

map.images (list).

- info (list) useful indixies and info - all are single values
  - map.name (character) name of the map file containing map image
  - full.map.name (character) full path to the map file containing map image
  - mapping.info.name (character) name of the build file used to map the same
  - design.name (character) name of the chip design
  - image.soft (character) name of the software used to analyze chip
  - nCol (numeric) maximum number of columns on chip
  - nRow (numeric) maximum number of rows on chip
- maps (list)
  - mat2map (matrix) indicies to map from a matrix to the chip location
  - map2mat (matrix) indicies to map from the chip image to a matrix
- indices (list)
  - col (matrix) indicies of columns on the chip for spots
  - row (matrix) indicies of rows on the chip for spots
- spotID (list)
  - file (character) vector of names of spot.IDs from given flat files
  - aCGH (character) vecotr of names of spot.IDs from map build file
- mapping.info (data.frame) This information is attained through the mapping build file. The number of columns is variable depending if certain criteria is flagged. There are 14 standard columns:
  - spot.ID (character) names of the spots - BACs, Clones, etc.
  - loc.start (numeric) starting location of spot with respect to chromosome
  - loc.center (numeric) central location of spot with respect to chromosome
  - loc.stop (numeric) ending location of spot with respect to chromosome
  - loc.genome (numeric) genomic location of the spot
  - Chrom (character or numeric) chromosome location of the spot
  - iChrom (numeric) chromosome location of the spot - factored version of Chrom
  - Arm (character) arm location of the spot (p,q)
  - iArm (numeric) arm location of the spot - factored version of arm (as if chromosome are arm i.e 1p=1,1q=2,2p=3,2q=4...)
  - Broad.Band (character) broad band location of spot including arm information (i.e q26, q27)

iBroad.Band (numeric) broad.band location of spot

Fine.Band (character) fine band location of spot including arm infomration (i.e q26.33, q27.1)

iFine.Band (numeric) fine band location of spot

There are three optional columnse:

mapped.by (character) indicates by what means spots and locations were identified (i.e FISH, BEP)

map.flag (character or numeric) identified spots with flags. These flags could be for reliablity, quality, etc.

random (any) random column to store addtional information user deems important

Band.Aid (list) This information is created through the mapping build file.

Centromeres (data.frame) Centromere data may be missing. It is dependent if the user provided a centromere file when creating build map files. The four columns are:

Chrom (character or numeric) the chromosome location

iChrom (numeric) the chromosome location – factored version of Chrom

loc the location of the centromere with respect to the chromsome

loc.genome the location of the centromere with respect to the genome

Regions (list) useful indicies and labels for plotting.

Chrom (data.frame)

Arm (data.frame)

Broad.Band (data.frame)

Fine.Band (data.frame)

Each data.frame has the same columns.

Chrom (character or numeric) chromosome location

Label (character) label to be used on graphs

Lower (numeric) lower bounding limit of the region

Center (numeric) central location of the region

Upper (numeric) upper bounding limit of the region

Note

mapping object files holding the map.image object are located in the array.image subdirectory maps. The example mapping object file given in the package is: array.images/maps/HB19Kv2.HG18$_H B19Kv2.design_I mageneV6.R1$

Source

References

See Also

`make.map.images`

Examples

```
# To see an example of a mapping file object
Write.aCGH.ex2()
load("array.images/maps/HB19Kv2.HG18_HB19Kv2.design_ImageneV6.RData")
ls()
names(map.images)

map.images

#subset through object using $
names(map.images$info)
```

---

marrayReload — *MAKES A MARRAYRAW OBJECT BY RELOADING IMAGE ANALYSIS FILES*

Description

This function will reload the original software image analysis files to create a limma RGList object that will be converted into an marray marrayRaw object. Additional information for the marrayRaw object will be added if applicable.

Usage

```
aCGHReLoadtoMarray(aCGH,
                   subdx=NA,
                   cc = "",
                   fcol = "Signal Mean",
                   gcol = NA,
                   fbcol ="Background Mean" ,
                   gbcol = NA,
                   vrb=TRUE,
                   knfile = NA,
                   design.file = "HB19Kv2.design",
                   spotLabel.design.file=12,
                   control.design.file=12,
                   special.control.levels = c("EMPTY","H2O"),
                   plate.label = 14,
                   ...)
```

Arguments

aCGH	an aCGH object
subdx	an index of samples to use
cc	column to be used as column argument in read.maimages or read.imagene calls of limma. May be left NA but then need to specify fcol, gcol, fbcol, and gbcol
fcol	name of column in image analysis files that stores the red channel foreground intensities

`gcol`	name of column in the image analysis files that stores the green channgel foreground intensities
`fbcol`	name of the column in the image analysis files that stores the red background intensities
`gbcol`	name of the column in teh image analysis files that stores the green background intensities
`vrb`	logical indicating if status messages should be printed
`knfile`	numeric either 1 or 2. Indicates if one file microarray experiemtns or 2 file microarray experiments should be used
`design.file`	name of design file to use
`spotLabel.design.file`	numeric indication of which column in the given design file lists the probeNames/spot.IDs/etc
`control.design.file`	numeric indication of which column in the given design file lists the controls for the spots
`special.control.levels`	list of special labels given to control column to indicate features
`plate.label`	numeric indication of which column in the given design file lists the plate information
`...`	additional arguments to be passed into the limma package read.maimages or read.imagene calls

Details

This function will create a marrayRaw object by retrieve the original software image analysis file[s] for each sample and reloading through the limma package. The limma package is used to handle imagene output.

Additional arguments may be passed into the limma read.maimage and read.imagene calls. columns for these calls should be specified by the cc argument. If cc is left blank, "", then the function will construct the column argument through fcol, gcol, fbcol, and gbcol. If there are two image analysis files per sample, fcol is the name of the column in the raw image analysis file representing the foreground intensities to be used and fbcol is the name of the column in the raw image analysis file representing the background intensities to be used. If there is only one file per sample fcol is the red (cy5)channel foreground intensities, gcol is the green(cy3) channel foreground intensities, fbcol is the red (cy5) channel background intensities, and gbcol is the green (cy3) channel background intensities. The names given should match a column name in the image analysis flat file exactly.

knife indicates if samples with one image anlaysis file or with two image analysis files should be used during the load in. (It is our understanding that unlike our package, limma and marray can handle only one type at a time).

After an RGList object is created, it is converted to an marrayRaw object by the bioconductor package convert. Addtional fields for the marrayRaw object are specified if applicable.

The marrayRaw object has information on the chip design. Only one deisgn may be specified for the marray object which is unlike the flexibiity of our aCGH object. The function therefore will subset the aCGH object based on a given design file. If a design file is not specified the function will use the design of the first sample in the list of samples to potentially use. Subdx indicates a subset of samples to use when creating the marray object (if all designs are the same).

We mentioned the marrayRaw object stores informtation concerning the chip design. The design information that must be provided is more rigid than what we require for the aCGHplus object. The

user must have specified a design for samples during the aCGHplus load-in process. This design may have been a name describing a particular design or it may have been the name of a design file containing the specifics of the chip design. For the marrayRaw object to be made correctly, the design must have been the name of a design file. The aCGHplus design file does not need to be a 'complete' design file; by complete we mean that all spots on the chip are accounted for. If a spot was left blank as a control or as prevented measures, etc. the complete file will still include the spot location and empty fields. The aCGHplus packages allows the design file to discard this information. The marray object requires a complete file. The design file that will be used is specified by design.file. If design.file is left as NA, the design file will become the design of the first sample in the subdx. There are three other arguments to this function we have not discussed yet that all are related to the design file: spotLabel.design.file, control.design.file, and special.control.levels. The spotLabel.design.file is a numeric indication of the column in the design file that stores the names of BAC.IDs/ProbeIDs/ect. control.design.file is a numeric indication of the column in the design data that has control information. This should be a column showing which spots were probes, and which were controls such as water or empty. Teh function will create a field for the controls. It will create an array with all fields equal to probe and where ever control.design.file is equal to special.control.levels, this being any value other than probe, the control will be placed back in as the value.

Value

a marrayRaw object

Note

required packages: limma, marray, Biobase, convert

Author(s)

Lori Shepherd

References

See Also

`marrayWrapper`, `aCGHReLoadtoLimma`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex1()
Write.aCGH.ex2()
load("RData/aCGH.RData")

marrayObj = aCGHReLoadtoMarray(aCGH,
                               subdx =c(1,3,5),
                               fcol = "Signal Mean",
                               fbcol ="Background Mean",
                               design.file ="HB19Kv2.design",
```

```
                                spotLabel.design.file=12,
                                control.design.file=12,
                                special.control.levels = c("EMPTY","H2O"),
                                plate.label = 14)
```

`marrayWrapper` *CONVERTS aCGH OBJECT TO MARRAYRAW OBJECT OF MARRAY CLASS*

Description

This function will take in an aCGH object and return an object of the marray class marrayRaw. The bioconductor convert function can be used to alter the marrayRaw object to objects of the limma and biobase packages.

Usage

```
aCGHtoMarray(aCGH,
             vrb=TRUE,
             Rbackground="array.images$matrix$cy3$Background.Mean",
             Gbackground="array.images$matrix$cy5$Background.Mean",
             Rforeground="array.images$matrix$cy3$Signal.Mean",
             Gforeground="array.images$matrix$cy5$Signal.Mean",
             weights=NA,
             design=NA,
             subdx=NA,
             spotLabel.design.file=12,
             control.design.file=12,
             special.control.levels = c("EMPTY","H2O"),
             plate.names = 14)
```

Arguments

`aCGH`	an aCGH object
`vrb`	logical indicating if status messages should be printed
`Rbackground`	location of background intensities for red/cy5 channel
`Gbackground`	location of background intensities for green/cy3 channel
`Rforeground`	location of intensities for red/cy5 channel
`Gforeground`	location of intensities for green/cy3 channel
`weights`	weights for spots
`design`	name of design file to use
`subdx`	subset of samples to use
`spotLabel.design.file`	numeric indication of which column in the given design file lists the probeNames/spot.IDs/etc
`control.design.file`	numeric indication of which column in the given design file lists the controls for the spots

`special.control.levels`
: list of special labels given to control column to indicate features

`plate.names`
: numeric indication of which column in the given design file lists the names of plate from which the chip was spotted

Details

This function will convert an aCGH object into an marrayRaw object of the marray package. The user must indicate where the values for the foreground and background intensities are located. If the defaults were used during the load in process, they should be stored in each samples' array.image file as a matrix of values. The marrayRaw object has information on the chip design. Only one deisgn may be specified for the marray object which is unlike the flexibiity of our aCGH object. The function therefore will subset the aCGH object based on a given design file. If a design file is not specified the function will use the design of the first sample in the list of samples to potentially use. Subdx indicates a subset of samples to use when creating the marray object (if all designs are the same).

We mentioned the marrayRaw object stores informtation concerning the chip design. The design information that must be provided is more rigid than what we require for the aCGHplus object. The user must have specified a design for samples during the aCGHplus load-in process. This design may have been a name describing a particular design or it may have been the name of a design file containing the specifics of the chip design. For the marrayRaw object to be made correctly, the design must have been the name of a design file. The aCGHplus design file does not need to be a 'complete' design file; by complete we mean that all spots on the chip are accounted for. If a spot was left blank as a control or as prevented measures, etc. the complete file will still include the spot location and empty fields. The aCGHplus packages allows the design file to discard this information. The marray object requires a complete file. The object can still be built with an incomplete design file, however the marray functions and conversion to limma or biobase will not be possible.

There are four other arguments to this function we have not discussed yet that all are related to the design file: spotLabel.design.file, control.design.file, special.control.levels, and plate.names. The spotLabel.design.file is a numeric indication of the column in the design file that stores the names of BAC.IDs/ProbeIDs/ect. control.design.file is a numeric indication of the column in the design data that has control information. This should be a column showing which spots were probes, and which were controls such as water or empty. Teh function will create a field for the controls. It will create an array with all fields equal to probe and where ever control.design.file is equal to special.control.levels, this being any value other than probe, the control will be placed back in as the value. Plate.name is a numeric indication of the column in the design file that stores plate information for the spots.

Value

a marrayRaw object

Note

required package: marray

The bioconductor convert package allows conversion of the marray package marrayRaw object to an RGList of the limma package or exprSet of the biobase package

See details on design file.

Author(s)

Lori Shepherd

References

See Also

aCGHReLoadtoMarray

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

marrayObj = aCGHtoMarray(aCGH)
```

---

meanPlot — *INTERACTIVE PLOT OF AN aCGH OBJECT'S MEAN LOG 2 RATIO*

Description

This function loads or takes in an aCGH object, with factor information, and creates an interactive plot of the mean across samples.

Usage

```
makeMeanPlot(fileloaded,
                aCGH=NA,
                fileLoad="RData/vizMean.RData",
                rmNA=TRUE,
                maxNA=.5,
                fitVls=TRUE,
                recenter=TRUE,
                bacwin=20,
                orientation.flipped=1 )
```

Arguments

fileloaded	logical value indicating if the file containing aCGH and factor information is loaded
aCGH	an aCGH object if fileloaded is T
fileLoad	file to load containing aCGH with factor information.

`rmNA` a logical indicating if NAs should be removed when calculating the mean across samples

`maxNA` a numerical indication of the percentage of sample data that must be present for the sample to be included

`fitVls` logical value indicating if the fitted log2 ratios are used or the raw log 2 ratios are used initially in all graphs

`recenter` logical value indicating if the log2 ratios should be recentered

`bacwin` numeric value indicating the number of bacwins (+/-) that will be shown on subsequent graph of surrounding area of a selected point

`orientation.flipped` value in orientation field of aCGH inventory representing flipped/dye swaps. Any sample with this value for the orientation will have their log2 ratios multipled by a -1 for analysis

Details

`makeMeanPlot` will create an interactive plot of the aCGH object's samples' mean log2 ratio values. The user may indicate whether the fitted log2 ratios or the raw log2 ratios are used to create the mean plot by setting fitVls. fitVls default is to use the fitted values. The user may toggle between fit and raw values interactivly after the plot is drawn. If no fit values are found (CBS not run) the apllication will use the raw values automatically. The user may also specify whether the data should be recentered. The default is to recenter data. The default will recenter data based on autosomes removing NAs. If other recenter methods are desired, the aCGH object should be recentered before use in this function and the function recenter should be set to F.

There is a cutoff range for the amount of data that must be observed for a sample to be included. maxNA is initially set to .5; half the data must be observed for the sample to be included. This value may be changed interactivly.

If there are dye swaps/orientation flips, samples will have log ratios opposite than what is normal. To correct for this effect we multiple a -1 on all sample values whose orientation suggests flipped.

The program will initially open with two blank windows, a mean graph, and a legend window. The legend window provides users withthe following options: select point, select BAC/spotID, raw, fitted, see table, write table to file, change NA cutoff, or exit all.

Clicking on select point will activate the mean graph for interaction. The user may then click upon any sample point in the graph of the means to view an area more closely. The selected point will be circled and the section surrounded by blue vertical lines. The two initially blank windows will plot two different graphs. One will be a flanking graph, showing sample log2 ratios for +/- the number of bacwins (default +/- 20) surrounding the selected spot/BAC. The sample at the BAC/spotID will be in a bold color, the flanking spots in faded color. Data plotted as a + character indicates observed values. Data plotted as an open square indicates missing/estimated values. The color indicates a factor type. The second graph window shows the sample log2 ratios at the selected BAC/spotID. The names of the samples are used as plotting character. Bold colors indicate real values, faded indicate observed/estimated values. In order for a user to select a different point on the frequency plot they must first re-select Select point on the legend device. Clicking on select BAC/spotID allows the user to enter the name of a specific BAC or spot.ID in question. This must match exactly to the name of a spot.ID in the aCGH object. Capitalization and spacing matters; type carefully. It will activate the graphs similarly to the select point circling the specified spotID and creating additional graphs. Clicking on raw or fitted will alter the log 2 ratio values. All three plots will be updated if the user has activated the two subsequent graphs. Raw will update to raw log 2 ratios and fitted will graph the fitted log 2 ratios if available. See table will show a table of all the samples for the selected spot.ID, the samples' log2 ratios, the samples' fitted log 2 ratios (if available), and the

factor group they belong to. Write table will write the table viewed using see table to a file. The file will be saved in a new directory created called ValuesAtspotIDs. The NA cutoff is changed on the R command line prompt. The user will be directed to enter a numeric value corresponding to the percentage of data observed that must be present for a sample to be included in the graph. Clicking on exit all will close all windows and exit the application. The user may click on any of the legend options at anytime.

Value

An interactive plot of the mean log2 ratio

Note

`makeFactorObject` should be run before running makeMeanPlot

Author(s)

Lori Shepherd

References

See Also

`vizMeanPrep`, `makeFactorObject`,`getMean`, `recenterData`,`setCutoff`, `makeFlank`, `makeSample`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# First we will run the function to make factor objects
# This will factor by sex
aCGH = makeFactorObject(aCGHloaded = FALSE, invloaded = TRUE, factorBy = "sex")

# because this is user interactive it is  commented out

# makeMeanPlot(fileloaded = FALSE)



# This is the same as running
# makeMeanPlot(fileloaded = FALSE, aCGH=NA,
#                  fileLoad="RData/vizMean.RData", rmNA=TRUE, maxNA=.5,
#                  fitVls=FALSE, recenter=FALSE, bacwin=20)
#or
#  load("RData/vizMean.RData")
#  makeMeanPlot(fileloaded=TRUE, aCGH)
```

`mean.vs.freq` *CREATES FILE OF PLOTS SHOWING MEAN AND FREQUENCY ACROSS SAMPLES*

Description

This function will create a mean and a frequency across samples plot based on samples in an aCGH object

Usage

```
make.Mean.vs.Freq.plots(
                        aCGH,
                        smplDX=NA,
                        maxNA=.25,
                        rmNA=TRUE,
                        fitVls=TRUE,
                        ylims=c(-1,1),
                        cutoffs=c(-.15,.15),
                        chrmList=1:22,
                        fileName="mean.freq.plot",
                        recenter=TRUE,
                        recenterOpt=2,
                        orientation.flipped=1
                        )
```

Arguments

`aCGH`	an aCGH object
`smplDX`	list of samples to use in aCGH object. Default, NA, uses all samples
`maxNA`	the maximum percentage of missing data allowable
`rmNA`	logical if NA values should be removed when calculating the mean across samples. Default is T
`fitVls`	logical indicating if fitted log2 ratios should be used
`ylims`	labels for y-axis of graphs
`cutoffs`	cuttoff values for frequency aberrations
`chrmList`	list of individual chromosome graphs to view
`fileName`	name of file to save to in the plots directory
`recenter`	logical if recentering should be done on aCGH values
`recenterOpt`	numeric indicating which recentering method to use. 1 is recentering on autosomes. 2 is recenter on autosomes removing those that are NA. 3 is recentering on autosomes, removing NA, and then on most aberrant samples. The default is 2.
`orientation.flipped`	value in orientation field of aCGH inventory representing flipped/dye swaps. Any sample with this value for the orientation will have their log2 ratios multipled by a -1 for analysis

Details

This function will create a file with two graphs per page: mean across samples and frequency across samples. An index of samples to use may be specified but the default, NA, will use all samples in the aCGH object. Samples will be excluded that have above a percentage of missing data. The percentage used is maxNA. The default will exclude any samples that have greated than 25% missing data. The user may specify whether raw or fitted raws are used in fitVls. fitVls=T, means the log2.ratios.fitted values are used in all calculations. When calculating the mean across samples, it may be desirable to weight samples. This may be the case if there are duplicates, dye swaps, or samples the user wishes to exclude. It should be either of length equal to the total number of samples in the aCGH object or equal to the length of smplDX. If there are dye swaps/orientation flips, samples will have log ratios opposite than what is normal. To correct for this effect we multiple a -1 on all sample values whose orientation suggests flipped. When calculating the frequency across samples, cutoffs must be set to determine aberrations. This should be of length two giving the minimum and maximum cutoff. The y axis label limits should also be specified.

Value

A file is saved in the plots directory. The first page is a graph of the mean log2 ratios and the frequency across samples for the entire genome. Every subsequent page is the mean and frequency plots for each specified chromosome in chrmList.

Note

Author(s)

Lori Shepherd

References

See Also

`meanPlot`, `freqPlot`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

make.Mean.vs.Freq.plots(aCGH)
```

PCAfile *calculates Principle component analysis for aCGH object*

Description

calculates principle component analysis for specified samples

Usage

```
makePCAfile(aCGH,
       smpldx = NA,
       fileName = NA,
       overwrite = FALSE)
```

Arguments

aCGH	An aCGH object
smpldx	a vector indicating which samples to use in calculation. The defualt uses all samples in the aCGH object
fileName	name for the new aCGH file containing PCA and sample information. The default saves to PCAtest
overwrite	logical. if a file exists with the specified fileName, indicates if it should be overwriten or assigned a temp name

Details

makePCAfile takes in a specific aCGH object and calculates a principle component anaylsis for the samples. The user may specify which samples are used for the calculations; the defualt uses all samples in the aCGH object. Information about the samples used are saved as a sampleInfo object

see princomp for details on PCtest object

sampleInfo has 5 slots: smpldx, scores, comments, exclude, removed smpldx is the sample index specified for calculations scores is of length of smpldx. It holds scores assigned by user when using interactive PCA viewer comments is also fo length smpldx. It holds comments assigned by user when using interactive PCA viewer exclude is a vector of sample indecies of samples flagged for removal removed is a vector of sample indecies of samples removed from analysis exclude and removed are assigned when using interactive PCA viewer

Value

Saves a file as fileName storing aCGH object, and the sample and principle component information

Note

CBS must have been run as well as flank.CBS. If these are not run an error will occur

This assumes that missing values have been removed from log2 ratios. Running just flankNA.CBS may not remove all NA's; it does not remove columns where log2 ratios are all NAs. Before running this function, getFailed should be run on the aCGH object to remove any columns of NA.

Author(s)

Lori Shepherd

See Also

getFailed,CBS.Batch, flankNA.CBS,princomp, runPCAFile

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


# it is important to note here that the example aCGH object
# has already had CBS performed on it. CBS must be performed to
# use this function
# see ?CBSBatch , ?flankNA.CBS

aCGH = getFailed(aCGH)

makePCAfile(aCGH)
```

---

plotGenome *CREATES A THREE ROW GENOME PLOT FOR AN aCGH SAMPLE*

Description

This function takes in an aCGH object and a desired sample number and creates a genomic plot of log2 ratios for that sample

Usage

```
plotGenome(aCGH,
           ismpl=1,
           ylims=c(-1,1),
           flipFlag=FALSE,
           ylab="log2 T/C",
           fitFlag=FALSE,
           fitFlag2=FALSE,
           spcl=NA,
           spcl.fit=NA,
           spcl.fit2=NA,
           fit.col="red",
           fit.col2="darkgreen",
           col.pnts=c("gray57","blue"),
           fit.cex=0.5,
```

```
            fit.cex2=0.5,
            chrom.label.cex=2,
            chrom.label.col="sienna",
            cent.flag=TRUE,
            cent.col="gray87",
            row.flags=c(TRUE,TRUE,TRUE),
            ttl=NA,
            ttlB=NA,
            hlines=NA,
            largeXlabel=TRUE)
```

Arguments

aCGH	an aCGH object
ismpl	numeric indicating with aCGH sample to use.Default uses the first sample in the aCGH object
ylims	what values to scale the y axis
flipFlag	will take the -1 value of points to be plotted
ylab	The label for the y axis. The defualt is the log 2 ratios.
fitFlag	logical indicating if a second set of points should be graphed
fitFlag2	logical if a third set of points should be graphed
spcl	The vector of points to graph. The default, NA, will graph the log2 ratios of the sample
spcl.fit	a vector of points to graph if fitFlag is T. This should be the length of the number of spots/BACs used. If fitFlag is T, The default, NA, will graph the fitted log 2 ratios
spcl.fit2	a vector of points to graph if fitFlag2 is T. This should be the length of the number of spots/BACs used. If fitFlag2 is T, the default, NA, will graph the fitted log 2 ratios
fit.col	color to use for the first set of additional points
fit.col2	Color to use for the second set of additional points
col.pnts	character vector of color names to use for original set of points
fit.cex	Size of the first set of additional points
fit.cex2	Size of the second set of additional points
chrom.label.cex	size of the chromosome labels
chrom.label.col	color for the chromosome labels
cent.flag	logical indicating if the center line should be drawn through each chromosome
cent.col	Color of the center line
row.flags	specifies the number of rows to create
ttl	a character vector of titles for the graphs. The default, NA, uses the sample number and the sample.ID name
ttlB	a character vector of second titles for the graphs. The default, NA, uses the aCGH sample image name

`hlines`	a vector indicating where horizontal lines should be drawn. The Default, NA, will not drawn any horizontal lines
`largeXlabel`	logical indicating if X-axis chromosome labels should be large or small. The default will print large chromosome labels above the axis in the specified chrom.label.col color and size.If False, the labels will be black, on the bottom of the axis, and have an additional chr added to the label.

Details

This function will plot a sample's genomic plot. The defaults will plot the raw log2 ratios for the sample (and the fitted ratios if fitFlag = T), but it is possible to pass in any set of values to plot over the genome by specifing spcl.

Value

A genomic plot of a sample in the aCGH object

Note

Author(s)

Lori Shepherd

References

See Also

`create.GenomeOneRow`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# will plot the aCGH's first sample
plotGenome(aCGH, 1)



graphics.off()
```

`plotMean` *CREATES A MEAN ACROSS SAMPLES PLOT*

Description

This function takes in an aCGH object and creates a mean across samples graph.

Usage

```
plotMean(aCGH,
         fitVls=TRUE,
         rmNA=TRUE,
         orientation.flipped=1,
         ...)
```

Arguments

`aCGH`	an aCGH object
`fitVls`	logical for if smoothed log2 ratios should be used. Default is T
`rmNA`	logical if NA should be removed while calculating the mean across samples
`orientation.flipped`	value in orientation field of aCGH inventory representing flipped/dye swaps. Any sample with this value for the orientation will have their log2 ratios multipled by a -1 for analysis
`...`	any additional arguements to be passed into a points call

Details

This function will take in an aCGH object, call the getMean function, and create a mean across sample graph. If there are dye swaps/orientation flips, samples will have log ratios opposite than what is normal. To correct for this effect we multiple a -1 on all sample values whose orientation suggests flipped.

Value

a plot of the mean across samples

Note

Author(s)

Lori Shepherd

References

See Also

cmean, meanPlot, makeMeanPlot, getMean, makeMeanGraph

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

plotMean(aCGH, col="blue", pch=3)



graphics.off()
```

QC1report *GET SUMMARY MEASURES INDICATING ARRAY QUALITY*

Description

These functions calculate and store some summary measures indicating array quality.

Usage

```
QCreport1.sample.j(j,
                   aCGHroster)

QCreport1.papply(aCGHroster)
```

Arguments

j	index of a particular sample to get array quality measures
aCGHroster	aCGH or aCGHroster object

Details

QCreport1.papply is designed to work with a cluster. Therefore, cluster nodes have to be reserved and the following commands run at the linux command line: module load lam lamboot -v R Load the package (aCGHplus, Rmpi and papply libraries must be loaded). QCreport1.papply utilizes the fuction QC1Wraps. QC1Wrap takes an individual sample entry and creates a list of the sample's array quality measures by calling the function QCreport1.sample.j. QCreport1.papply uses the papply function to send out the multiple QC1Wrap calls to different nodes.

QCreprt1.sample.j takes in an aCGH or aCGHroster object and a sample index and calculates array quality measures. The measures that are calculated are the total number of array spots, the total number of non NA spots, the total number of spot flags, the median cy3 signal mean, the median

cy5 signal mean,the median absolute deviation (mad) of cy3 signal mean, the mad of cy5 signal mean, the median cy3 background signal mean, the median cy5 background signal mean, the mad cy3 signal mean, and the mad cy5 signal mean. This list of measures is returned.

Value

QCreport1.sample.j returns a list of data measures for a single sample

QCreport1.papply returns and aCGH or aCGHroster object with an updated inventory to included a QC1 data frame of array quality measures.

Note

QCreport1.papply uses the papply package.

Author(s)

Lori Shepherd

References

See Also

`reportMissingInvEntries`,`reportMissingArrays`, `diagnosticRosterCheck`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# This will create a list of data measures for sample 3
smplInfo = QCreport1.sample.j(3, aCGH)
```

---

recenter — *RECENTERS LOG2 RATIOS OF aCGH OBJECT OR REMOVES RECENTERING*

Description

These functions recenter an aCGH object's log2 ratios data. There are three recentering options. It is possible to remove recentering values.

Usage

```
recenter.aCGH(aCGH,
         maxNA=.1,
         cutoffs=c(-.225, .225),
         qcntr=0.5,
         fitVls=TRUE,
         saveACGH=TRUE,
         fileName="RData/aCGH.RData",
         useStep = 2)

undo.recenter(aCGH,
              saveACGH=TRUE,
              fileName="RData/aCGH.RData")
```

Arguments

aCGH	an aCGH object
maxNA	maximum percentage of NA's that may make up a sample for sample inclusion
cutoffs	list of lower and upper bound for determining aberrant regions
qcntr	percentage of most aberrant samples to use
fitVls	logical indicating if smoothed or raw log2.ratios should be used for apply cutoffs to find most aberrant
saveACGH	logical if new recentered aCGH object should be saved
fileName	path and name to save aCGH object if saveACGH is T
useStep	numeric indication of which recentering option to apply to the aCGH object. see details on three options. default is the second recentering option

Details

`recenterACGH` will take in an aCGH object and will recenter its log2 ratio data. There are three options for recentering data. The values of the recentering option used will be saved in the aCGH object as recenterVls. The first option, useStep=1, will recenter data based purely on autosomes. The second option, useStep=2, will recenter data based on autosomes removing missing data (NAs). This is our standard way of recentering. It takes the median of all autosomal samples that contain above a given percentage of data. Samples that are missing maxNA percentage are not included in calculating the recentering value.

The third option, useStep=3, will recenter data based on a percentage of the most aberrant spots. Aberration is determined by given cutoffs.

`undo.recenter` will take off any recentering performed. The log2 ratios will be returned to their original values by adding back the recenterVls.

The aCGH object will be returned but will only be saved if saveACGH is T.

Value

aCGH object

Note

recenter is called in meanPlot and cmean

Author(s)

Lori Shepherd

References

See Also

`median`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

aCGHcentered = recenter.aCGH(aCGH, useStep=2)

aCGHuncentered = undo.recenter(aCGHcentered)
```

ReLevelDF *RELEVELS DATA FRAME AFTER ALTERATIONS*

Description

This function will relevel a data frame

Usage

```
ReLevelDF(inidf)
```

Arguments

`inidf`	data frame to be releveled

Details

This is a helper function to relevel data.frame's after alterations have been made to the data. This function will take in a data.frame and individually level each column utilizing is.factor and factor of the base package.

Value

a data frame

Note

uses functions is.factor and factor for base package

Author(s)

Lori Shepherd

References

See Also

`factor`, `data.frame`

Examples

```
df = list()
df$a = rep(c("one", "two", "three"), 2)
df$b = rep("test", 6)
df$b[4] = "remove"
df = as.data.frame(df)

#see data frame
df

# for comparison see levels
levels(df$a)
levels(df$b)

# alter data.frame
df[4,2] = "test"

# levels are the same
levels(df$b)

#relevel
df2 = ReLevelDF(df)

# compare
levels(df2$b)
```

---

reportMissingArrays

CREATES REPORT OF MISSING IMAGE FILES BASED ON INVENTORY

Description

This function will take in an inventory file and a path to image analysis files, and report on which image files are missing based on the inventory

Usage

```
makeReportOfMissingArrays(arraypath="arrays/",
                          inventory.file = "Uber.inv",
                          inv.sep=";",
                          output.file="masterDataArrays.Report.txt",
                          twofileExt = c("_532.txt", "_635.txt"),
                          vrb=TRUE,
                          ...)
```

Arguments

`arraypath`	The path to the image analysis files
`inventory.file`	inventory file
`inv.sep`	seperation character for inventory file
`output.file`	name/path of the report that will be written
`twofileExt`	list of file extenstions for samples with two image files
`vrb`	logical if status messages should be printed
`...`	any aditional arguments to be passed into a read table call for reading the inventory (besides file,sep, and header already specified)

Details

This function will try and match all of the image.names in the inventory file with the image files in the given directory, arraypath. If a file, or two are not found, the application will note which files are missing. A report is generated with details of missing image names.

Value

status message of how many files are missing. A report, text file, is generated with details of missing files

Note

This only works for files with one or two files per sample.

This assumes there is a column in the inventory, image.name, that stores the base name for image files.

This currently assumes image.names that begin with HB and MB have two files

Author(s)

Lori Shepherd

References

See Also

`reportMissingInvEntries`, `QC1report`, `diagnosticRosterCheck`

Examples

```
library(aCGHplus)

# This function will use the example inventory given in Write.aCGH.ex1
Write.aCGH.ex1()

makeReportOfMissingArrays(arraypath="arrays/",
                          inventory.file = "aCGH.ex1.inv",
                          inv.sep=",",
                          output.file="noMissingArrays.Report.txt",
                          twofileExt = c("_532.txt", "_635.txt"),
                          vrb=TRUE)

# now create a fake example inventory
r = read.table("aCGH.ex1.inv", sep=",", header=TRUE)
s = r
s[(dim(r)[1]+1),] = NA
s[(dim(s)[1]+1),] = NA
imN = as.character(s$image.name)
imN[dim(s)[1]]= "mockImage"
imN[(dim(s)[1])-1]= "HBmockImage"
s$image.name = imN
write.table(s, file="mock.inv", sep=",", row.names=FALSE)

makeReportOfMissingArrays(arraypath="arrays/",
                          inventory.file = "mock.inv",
                          inv.sep=",",
                          output.file="MissingArrays.Report.txt",
                          twofileExt = c("_532.txt", "_635.txt"),
                          vrb=TRUE)
```

---

reportMissingInvEntries

REPORT MISSING INVENTORY ENTRIES BASED ON IMAGE FILES

Description

This function will take in an inventory file and a list of file extensions, and match image names in a directory to the image.name column in the inventory. It will keep track of missing inventory entries.

Usage

```
makeReportOfMissingInv(exts = c("_532.txt"),
                       inventory.file = "../Uber.inv",
                       inv.sep=";",
                       inv.column="image.name",
                       output.file="../masterDataInv.Report.txt",
                       vrb=TRUE,
                       ...)
```

Arguments

exts	a list of the suffixes representing unique files for samples if added to the names of the inv.column in the inventory. For samples that have two files, only list one of the file extensions.($_5 32.txt$)$.iei f the image f iles is$"$HBmytest_6 35.txt$", $the inv.column in the inve
inventory.file	
inv.sep	seperation character for inventory file
inv.column	name of the column in the inventory that stores the base name of the image files
output.file	name/path of the report that will be written
vrb	logical if status messages should be printed
...	any additiional arguments to be passed into a read.table call for reading the inventory file (besides file,sep, and header already specified)

Details

This function will try and match all of the image file names in the working directory to teh image.name column of the inventory file. If an image file exists but it is not found in the inventory file, the application will note the name of this file. A report is genreated with all image files missing an inventory entry.

Value

status message of how many inventory entries are missing. A report, text file, is generated with details of missing entries

Note

This assumes that you run this function from a working directory where all the image files are stored (in example data this would be the arrays directory)

This also assumes this working directory only contains image files

This assumes there is a column in the inventory that stores the base name for image files.

Author(s)

Lori Shepherd

References

See Also

reportMissingArrays, QC1report, diagnosticRosterCheck

Examples

`RosterBatch` *LOADS DATA IN BATCH*

Description

This function will create the array.image, design, and map files for all samples listed in an inventory file in batch mode. This is helpful for loading in large data sets.

Usage

```
RosterBatch(array.dir="arrays",
                    inventory.file="aCGH.ex1.inv",
                    image.dir="array.images",
                    inv.sep=",",
                    overwrite=FALSE,
                    BatchDX=NA,
                    nbatchjobs=2,
                    ibatchjob=1,
                    vrb=TRUE)
```

Arguments

`array.dir`	name or path to the directory holding the raw data files from image software
`inventory.file`	Name of the inventory file listing samples that will make up the aCGHroster. The default uses the inventory given as part of the example dataset. (see also example `aCGH.ex1.inv`)
`image.dir`	name of the directory to store image array files and map files. The subdirectories images, maps, and designs will be created within this directory
`inv.sep`	The separation character for the inventory file. Will be passed in as sep for a read.table call
`overwrite`	flag to overwrite existing array.image files. If this is of length one, then it will be as specified for all samples. If this is of length of number of samples, each will be as specified. Default is to not overwrite files
`BatchDX`	index of roster samples to be run in batch mode
`nbatchjobs`	number of batch jobs
`ibatchjob`	the batch job turn in this call of SmoothBatch
`vrb`	a logical flag indicating if status messages should be printed

Details

This function will load all samples specified in the inventory file and BatchDX. If BatchDX is not specified or is NA, all the samples are loaded.

This function will split the inventory file based on the number of batch jobs to be run (nbatchjobs). Every nbatchjob element is in one run of the batch. Therefore, for nbatchjobs=2 every other element is in one set, nbatchjobs=3 every third element is in one set, and so on.

ibatchjob specifies which run the batch is performing. If nbatchjobs=2 this may be either 1 or 2; If nbatchjobs=3, this may be 1,2,or 3. This is an indicator of which element to begin; If nbatchjobs=3, ibatchjob=1, every third sample beginning with the first will be in the set. If nbatchjobs=2, ibatchjob=2, every third sample beginning with the second will be in the set; and so on.

Value

creates array.image, map, and design files for sample's listed in the inventory file

Note

This function will only create array.image, map, and design files. It will not create an aCGHroster object. See example

Author(s)

Lori Shepherd

References

See Also

create.roster.R, Roster.papply

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

 # a mapping information file must have been built
 # this will build a mapping information file
 buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
          map.label="HB19Kv2.HG18",
         spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
         loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
         Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
             "chr7","chr8","chr9","chr10","chr11","chr12",
             "chr13","chr14","chr15","chr16","chr17","chr18",
             "chr19","chr20","chr21","chr22","chrX","chrY"),
         chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
         map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
         rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
         YchromNum = 24, centromere.file =  "Centromeres.txt",
         centromere.loc.lbl="location", centromere.file.sep="\t")


# this will run the first batch job
# since there is a total of two jobs (nbatchjobs=2)
# this will run every other sample in the inventory beginning with
# the first element

RosterBatch(array.dir="arrays",
            inventory.file="aCGH.ex1.inv",
            image.dir="array.images",inv.sep=",",overwrite=TRUE,
            nbatchjobs=2,ibatchjob=1)

# this will run the second batch job
# since there is a total of two jobs (nbatchjobs=2)
# this will run every other sample in the inventory beginning with
```

```
# the second element


RosterBatch(array.dir="arrays",
            inventory.file="aCGH.ex1.inv",
            image.dir="array.images",inv.sep=",",overwrite=TRUE,
            nbatchjobs=2,ibatchjob=2)

# Now all the array.image, map, and design files for the samples listed
# in the inventory file have been created without creating the roster
# object
# create.roster.R is run again with overwrite=F, to create an aCGHroster
# file/object for the full inventory (note: this is optional if the only
# objective was to load in files)

create.roster.R(array.dir="arrays",roster.file="RData/aCGHroster.RData",
         inventory.file="aCGH.ex1.inv", image.dir="array.images",
           inv.sep=",", overwrite=FALSE, returnFlag=FALSE, vrb=TRUE)
```

`Roster.papply` *CREATES aCGHroster OBJECT*

Description

This function will create and aCGHroster object utilizing papply. This function will use papply to send multiple jobs to nodes on a cluster for efficiency.

Usage

```
Roster.papply(array.dir="arrays",
              inventory.file="aCGH.ex1.inv",
              image.dir="array.images",
              inv.sep=",",
              overwrite=FALSE,
              BatchDX=NA,
              roster.file = "RData/aCGHroster.RData",
              vrb=TRUE)
```

Arguments

`array.dir`	name or path to the directory holding the raw data files from image software
`inventory.file`	Name of the inventory file listing samples that will make up the aCGHroster. The default uses the inventory given as part of the example dataset. (see also example `aCGH.ex1.inv`)
`image.dir`	name of the directory to store image array files and map files. The subdirectories images, maps, and designs will be created within this directory

`inv.sep`	The separation character for the inventory file. Will be passed in as sep for a read.table call
`overwrite`	flag to overwrite existing array.image files. If this is of length one, then it will be as specified for all samples. If this is of length of number of samples, each will be as specified. Default is to not overwrite files
`BatchDX`	Flag for running create.roster in batch mode. Dafault is NA and will run all nsmpls. If an index is specified batch mode will section off pieces of the inventory BatchDX long to run.
`roster.file`	the name or path to save the roster object. The default is to save it as aCGHroster.RData in the RData directory (i.e. RData/aCGHroster.RData)
`vrb`	a logical flag indicating if status messages should be printed

Details

This function is designed to work with a cluster. Therefore, cluster nodes have to be reserved and the following commands run at the linux command line: module load lam lamboot -v R

Load the package (aCGHplus, Rmpi and papply libraries must be loaded).

Roster.papply loads the given inventory file and then utilizes the internal fuction rosterWraps. rosterWrap takes an individual sample entry and creates the sample's array.image, mapping file, and design.file. Roster.papply uses the papply function to send out the multiple rosterWrap calls to different nodes. After all the samples are completed, the function will create an aCGHroster using the aCGHplus create.roster.R function. This aCGHroster is returned.

Value

Roster.papply will return an aCGHobject

Note

This function uses the papply package.

Author(s)

Lori Shepherd

References

See Also

`RosterBatch`,`create.roster.R`

Examples

`runPCAFile` *create PCA viewer to interogate sample sets*

Description

This application creates an interactive 3D PCA graph of samples. CBS must have been performed as well as flank.CBS

Usage

```
runPCA(aCGH,
       smpldx = NA,
       aCGHfileName = NA,
       factorList = NA,
       invName = NA,
       overwriteInv = FALSE)
```

Arguments

`aCGH`	An aCGH object
`smpldx`	a vector indicating which samples to graph. The defualt uses all samples in the aCGH object
`aCGHfileName`	name for the new aCGH file containing PCA, sample, and factor information. The default saves to PCAtest
`factorList`	a numeric or character vector indicating the columns within the aCGH inventory to cycle through for factoring
`invName`	name of the initial inventory file. The default saves to origPCA
`overwriteInv`	logical indicating whether the first inventory file should overwrite the last saved inventory of the aCGH object

Details

`runPCA` takes in a specific aCGH object. This will create or load existing PCA data for that object. Specific sample information and factor information is also created and stored. An interactive 3D PCA graph is displayed. The user may interrogate data: commenting, scoring, removing, etc. sample points. Genome plots and chip images are displayed for each selected sample. The user may save inventory files corresponding to sample data, as well as an aCGH file.

Value

Creates a 3D PCA graph. Inventory files and aCGH files created.

Note

CBS must have been run as well as flank.CBS or an error will occur

Author(s)

Lori Shepherd

See Also

CBS.Batch, makePCAfile, initFactor, makeInvDir, graphPCA, selectPt, loadPCAInv

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

# Write.aCGH.ex2()
# load("RData/aCGH.RData")


# it is important to note here that the example aCGH object
# has already had CBS performed on it. CBS must be performed to
# use this function
# see ?CBSBatch , ?flankNA.CBS


# because this is user interactive it has been commented out

# runPCA(aCGH)

###  or

# runPCA(aCGH, factorList=c("sex", "orientation", NA))
```

---

savePCAInv *saves inventory from 3D PCA Viewer*

Description

saves inventory from the 3D PCA Viewer in PCA.inventory directory.

Usage

```
savePCAInv(aCGH,
       sampleInfo,
       factorInfo,
       PCtest,
       plotInfo,
       fileName = "origPCA",
       overwrite = FALSE)
```

Arguments

aCGH	An aCGH object
sampleInfo	object containing data for samples
factorInfo	object containing factor information
PCtest	object containing principle component analysis data

plotInfo	object containing plotting information
fileName	name the file will be saved to. Default is origPCA.
overwrite	flag whether the last saved inventory in aCGH is to be replaced with the new file

Details

savePCAInv will save the principle component data, sample data, and factor data in a PCA.inventory directory. If the fileName is not specified, origPCA is used. If the fileName is passed in as NA or empty string, the date and time are used as the file name. The aCGH object is updated. The aCGH object PCAinv is updated to include the new file name. If overwrite is true, the last saved inventory is replaced.

Value

aCGH

Note

It is called from within runPCAfile and choiceAct

makePCAFile, and either initFactor or factorObj must have been run

Author(s)

Lori Shepherd

See Also

runPCAFile, choiceAct, loadPCAInv, date, graphPCA

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


# it is important to note here that the example aCGH object
# has already had CBS performed on it. CBS must be performed to
# use this function
# see ?CBSBatch , ?flankNA.CBS

#remove NA columns
aCGH = getFailed(aCGH)

# make PCA file
makePCAfile(aCGH)

# load PCA information
load("RData/PCAtest.RData")

# make Factor object
```

```
factorInfo=initFactor(aCGH,smpldx=sampleInfo$smpldx,factorList=c("sex", "orientation", NA

# make ploting object
plotInfo = graphPCA(aCGH, PCtest, factorInfo, interactive=FALSE)

# makes needed directory to store inventories
makeInvDir()

# saves inventory
aCGH = savePCAInv(aCGH,sampleInfo, factorInfo, PCtest, plotInfo,
                fileName=NA,overwrite=FALSE)

graphics.off()
rgl.close()
```

`SegmentMasking` *PERFORMS SEGMENT MASKING*

Description

These functions perform segment masking.

Usage

```
autoMask(aCGH,smplDX=NA,bacDX=NA,
         cuts=c(-0.1254537,0.1325175),
         plt=TRUE,vrb=TRUE)

armMask(aCGH,iarm=1,smplDX=NA,bacDX=NA,
        cuts=c(-0.1254537,0.1325175),
        plt=TRUE,vrb=TRUE)
```

Arguments

`aCGH`	an aCGH object (that has undergone CBS)
`smplDX`	index of samples to use
`bacDX`	index of spot.IDs to use
`cuts`	list of the lower and upper bounds for calling gains or loss. All log2 ratios of or below the first value of the list will be considered a loss. All log2 ratios of or above the second value of the list will be considered a gain.
`plt`	logical if plots should be drawn and a file saved in the plots directory
`vrb`	logical indicating if status messages should be printed
`iarm`	number indicating which arm factor level should be executed. This is compared to aCGH$mapping.info$iArm.

Details

Value

autoMask returns an aCGH object that contains an object SegMask with all segmental masking data.

armMask returns a matrix of information on the segmental masking for an arm.

Note

This function assumes CBS has already been performed.

armMask utilizes the internal function CountSegs

Author(s)

Lori Shepherd

References

See Also

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

temp = armMask(aCGH, iarm=1, smplDX=NA, bacDX=NA,vrb=TRUE,
               cuts=c(-0.1254537,0.1325175), plt=TRUE)


aCGH2 = autoMask(aCGH,smplDX=NA,bacDX=NA,vrb=TRUE,plt=TRUE,
                 cuts=c(-0.1254537,0.1325175))
```

---

selectPt *makes 3D Viewer interactive*

Description

makes 3D PCA Viewer interactive

Usage

```
selectPt(aCGH,
       PCtest,
       sampleInfo,
       factorInfo,
       plotInfo,
       heat = "correction",
       pt2 = NA,
       gen = "PCA")
```

Arguments

aCGH	An aCGH object
PCtest	object containing principle component analysis data
sampleInfo	object containing data for samples
factorInfo	object containing factor information
plotInfo	object containing information for graphical devices
heat	flag for activated heat map. options are "cy", "background" or "correction" heat maps.The defaults is cy3 and cy5 chips
pt2	flag for last point chosen. used for graphing purposes
gen	flag for genome graph. options are "PCA" or "gen". The default is "PCA"

Details

`selectPt` allows the user to interrogate principle component data and aCGH samples. A 3D PCA Viewer is made active so the user may select any sample point within the graph. A genome plot, and chip images are displayed when a sample is selected. If an empty space is selected, an interactive legend is activated with choices to manipulate sample inforamtion. see `choiceAct` for details concerning legend options. If more than one sample is selected, the first sample with regards to position in aCGH is used. A user may rotate the 3D space using the right mouse button, zoom with the center button, and select an area using the left mouse button.

Value

Note

this is called from within `runPCAFile` to make PCA graph interactive

to run seperately the following must also have been run: `CBS.Batch`, `flankNA.CBS`, `PCAfile`, initFactor or `factorObj`, and `graphPCA` with interactive set to T

Author(s)

Lori Shepherd

See Also

`runPCAFile`, `PCAfile`, `initFactor`, `factorObj`, `graphPCA`, `rgl`, `select3d`, `choiceAct`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# it is important to note here that the example aCGH object
# has already had CBS performed on it. CBS must be performed to
# use this function
# see ?CBSBatch , ?flankNA.CBS

#remove NA columns
aCGH = getFailed(aCGH)

# make PCA file
makePCAfile(aCGH)

# load PCA information
load("RData/PCAtest.RData")

# make Factor object
factorInfo=initFactor(aCGH,smpldx=sampleInfo$smpldx,factorList=c("sex", "orientation", NA

# makes graphs
plotInfo = graphPCA(aCGH, PCtest, factorInfo, interactive=FALSE)

#makes needed directory to hold inventories
makeInvDir()

# because this is user interactive it has been commented out

# interactive

# selectPt(aCGH, PCtest, sampleInfo, factorInfo, plotInfo)



graphics.off()
rgl.close()
```

---

setCutoff *SETS A NA CUTOFF FOR DATA*

Description

This function takes in an aCGH object, a vector of data, and a NA cutoff, along with a factor type created by using makeFactorObject. Based on the factor grouping and the maxNA given, the mean is calculated and the data above the missing Data threshold will be removed.

Usage

```
setCutoff(aCGH,
          yData,
          maxNA = 0.5,
          grp.lbls = "groupA")
```

Arguments

aCGH	an aCGH object
yData	vecotr of data cutoff is applied to
maxNA	numerical indicating a percentage of data that may be missing for a sample to be included. Default is .5 (half)
grp.lbls	character vector of factor objects

Details

The aCGH object must have been factored using makeFactorObject. The group labels created while factoring are passed in as an argument along with an aCGH object, a vector of data, and a percentage of NAs to use as a cutoff. A mean is calculated for the log2 ratios for each group. If the percentage of missing data is above the maxNA, the value is removed.

Value

a vector with values above cutoffs removed

Note

makeFactorObject must have been run. see vizMeanPrep. called in cmean and meanPlot

Author(s)

Lori Shepherd

References

See Also

vizMeanPrep

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


newaCGH = makeFactorObject(aCGHloaded = TRUE, invloaded = TRUE)
```

```
vec = rep(1, newaCGH$data.info$nBAC)

newvec = setCutoff(newaCGH, vec, grp.lbls="groupA")
```

`smooth2D.papply` *SMOOTH DATA ON CLUSTER*

Description

This function will smooth the log2 ratios of a sample using a cross validation-like technique utilizing papply. This function will use papply to send multiple jobs to nodes on a cluster for efficiency.

Usage

```
smooth2D.papply(aCGHroster,
                overwrite=FALSE,
                BatchDX=NA,
                map.name=NA,
                thetas=seq(0.5,7.5,length=11),
                cvLevel=10,
                LossF=c("L1","L2"),
                useResid=TRUE,
                BCoption=c("none","raw","smoothed"),
                lambdas=seq(0,1,length=2),
                DesignFlag=FALSE,
                DesignList=c("Plate","Pin","PlateRow","PlateCol","Repetition"),
                lib.loc=NA,
                cyC.label="array.images$matrix$cy3$Signal.Mean",
                cyT.label="array.images$matrix$cy5$Signal.Mean",
                cyC.BC.label="array.images$matrix$cy3$Background.Mean",
                cyT.BC.label="array.images$matrix$cy5$Background.Mean",
                cy.grid.label="array.images$matrix$Grid",
                exclude.rule="array.images$matrix$cy3$Flag!=0",
                output.label="smooth2D",
                pname="auto",
                noDesignExit=FALSE,
                weightSex.loess=1/10,
                excludeSex.smooth=FALSE,
                excludeSex.design=FALSE,
                weightNonMap.loess=1/10,
                excludeNonMap.smooth=TRUE,
                excludeNonMap.design=TRUE,
                ilambda.default=1,
                saveAll=TRUE,
                plt=TRUE,
                vrb=TRUE)
```

Arguments

aCGHroster	aCGH roster object
overwrite	Flag to overwrite existing array.images
BatchDX	index of roster samples to be run in batch
map.name	name of map file, if NA uses default in array.images
thetas	a sequence of numbers to be tested as the smoothing value, function will use whichever theta yields best result. This is passed as argument into Smooth.Image.CV. The default is a sequence of 11 values from .5 to 7.5
cvLevel	cross validation level, number indicating percentage of data points to remove before smoothing and use as a reference check. Default is 10, therefore 1/10 of the data will be removed and used as validation of method. This is passed as an argument into Smooth.Image.CV
LossF	Must be either "L1" or "L2", choice of which data from Smooth.Image.CV to use. "L1" represents using the sum of the absolute difference of the observed values and smoothed values. "L2" represents using the sum of the difference of oberved values and the smoothed values squared.
useResid	logical flag indicating if residuals are to be used
BCoption	option whether to background correct. Must be either "none", "raw", or "smoothed"
lambdas	sequence of numbers represeting the percentage of background correction to use, function will use whichever lambda yields best result. The default currently checks 0 (none) and 1(100%)
DesignFlag	logical indicating if the Design file for the array.image should be loaded
DesignList	List of the names of columns in the design file that should be corrected
lib.loc	path name to local library, Default, NA, assumes working in main R directory. DNAcopy and fields must be installed in the loc.lib library.
cyC.label	path/name to the sample's control signal mean matrix, the default is "array.images$matrix$cy3$Signal
cyT.label	path/name to the sample's tumor signal mean matrix, the default is "array.images$matrix$cy5$Signal.
cyC.BC.label	path/name to the sample's control background signal mean matrix in the array.image object, the default is "array.images$matrix$cy3$Background.Mean"
cyT.BC.label	path/name to the sample's tumor background signal mean matrix in the array.image object, the default is "array.images$matrix$cy3$Background.Mean"
cy.grid.label	path/name to the array.image's grid matrix, the default is "array.images$matrix$Grid"
exclude.rule	will indicate a rule for exclusion of points. The default is "array.images$matrix$cy3$Flag!=0" (see details).
output.label	name of the object to be added to the array.image object, Default is "smooth2D"
pname	a name to be used for a file containing the plot outputs of the function, the default, "auto" will plot in a directory plots/ with the name SmoothArray.[the root name of the array.image file]. If the user enters a blank field, "", the function will assume no plotting
noDesignExit	logical indicating if the function should abort if no design file is found
weightSex.loess	percentage of sex chromosomes to use
excludeSex.smooth	logical indicating if sex chromosome spots should be excluded from smoothing

`excludeSex.design`
: logical indicating if sex chromosome spots should be excluded from design phase

`weightNonMap.loess`
: percentage of non mapped samples to use

`excludeNonMap.smooth`
: logical indicating if non-mapped chromosome spots should be excluded from smoothing

`excludeNonMap.design`
: logical indicating if non-mapped chromosome spots should be excluded from design phase

`ilambda.default`
: which lambda should be stored as default Mxy

`saveAll`
: logical indicating if all data should be saved

`plt`
: logical indicating if plotting should occur

`vrb`
: a logical flag indicating if status messages should be printed

Details

This function is designed to work with a cluster. Therefore, cluster nodes have to be reserved and the following commands run at the linux command line: module load lam lamboot -v R

Load the package (aCGHplus, Rmpi and papply libraries must be loaded).

smooth2D.pappy uses the papply function to send out multiple smoothWrap calls to different nodes. The smoothWrap function will smooth an individual sample's log2 ratios and make any design corrections if such options are flagged.

Value

nothing is returned but updated log2 ratios are saved in appropriate objects

Note

This function uses the papply package.

Author(s)

Lori Shepherd

References

See Also

`SmoothArray`, `SmoothBatch`,`DiagPlot.Smooth2D`

Examples

SmoothArray *SMOOTH DATA*

Description

This function will smooth the log2 ratios of a sample using a cross validation-like technique.

Usage

```
SmoothArray(fname.array.images,
            map.name=NA,
            thetas=seq(0.5,7.5,length=11),
            cvLevel=10,
            LossF=c("L1","L2"),
            useResid=TRUE,
            BCoption=c("none","raw","smoothed"),
            lambdas=seq(0,1,length=2),
            DesignFlag=FALSE,
            DesignList=c("Plate","Pin","PlateRow","PlateCol","Repetition"),
            lib.loc=NA,
            cyC.label="array.images$matrix$cy3$Signal.Mean",
            cyT.label="array.images$matrix$cy5$Signal.Mean",
            cyC.BC.label="array.images$matrix$cy3$Background.Mean",
            cyT.BC.label="array.images$matrix$cy5$Background.Mean",
            cy.grid.label="array.images$matrix$Grid",
            exclude.rule="array.images$matrix$cy3$Flag!=0",
            output.label="smooth2D",
            pname="auto",
            noDesignExit=FALSE,
            weightSex.loess=1/10,
            excludeSex.smooth=TRUE,
            excludeSex.design=TRUE,
            weightNonMap.loess=1/10,
            excludeNonMap.smooth=TRUE,
            excludeNonMap.design=TRUE,
            ilambda.default=1,
            saveAll=TRUE,
            plt=FALSE,
            vrb=TRUE)
```

Arguments

`fname.array.images`	name/path to a sample's array.image file
`map.name`	name/path of map to use for residuals. The default, NA,uses default map listed in the array.image file
`thetas`	a sequence of numbers to be tested as the smoothing value, function will use whichever theta yields best result. This is passed as argument into Smooth.Image.CV. The default is a sequence of 11 values from .5 to 7.5

`cvLevel` cross validation level, number indicating percentage of data points to remove before smoothing and use as a reference check. Default is 10, therefore 1/10 of the data will be removed and used as validation of method. This is passed as an argument into Smooth.Image.CV

`LossF` Must be either "L1" or "L2", choice of which data from Smooth.Image.CV to use. "L1" represents using the sum of the absolute difference of the observed values and smoothed values. "L2" represents using the sum of the difference of oberved values and the smoothed values squared.

`useResid` logical flag indicating if residuals are to be used

`BCoption` option whether to background correct. Must be either "none", "raw", or "smoothed".

`lambdas` sequence of numbers represeting the percentage of background correction to use, function will use whichever lambda yields best result. The default currently checks 0 (none) and 1(100%)

`DesignFlag` logical indicating if the Design file for the array.image should be loaded

`DesignList` List of the names of columns in the design file that should be corrected

`lib.loc` path name to local library, Default, NA, assumes working in main R directory. DNAcopy and fields must be installed in the `loc.lib` library.

`cyC.label` path/name to the sample's control signal mean matrix, the default is "array.images$matrix$cy3$Signal

`cyT.label` path/name to the sample's tumor signal mean matrix, the default is "array.images$matrix$cy5$Signal.

`cyC.BC.label` path/name to the sample's control background signal mean matrix in the array.image object, the default is "array.images$matrix$cy3$Background.Mean"

`cyT.BC.label` path/name to the sample's tumor background signal mean matrix in the array.image object, the default is "array.images$matrix$cy3$Background.Mean"

`cy.grid.label` path/name to the array.image's grid matrix, the default is "array.images$matrix$Grid"

`exclude.rule` will indicate a rule for exclusion of points. The default is "array.images$matrix$cy3$Flag!=0" (see details).

`output.label` name of the object to be added to the array.image object, Default is "smooth2D"

`pname` a name to be used for a file containing the plot outputs of the function, the default, "auto" will plot in a directory plots/ with the name SmoothArray.[the root name of the array.image file]. If the user enters a blank field, "", the function will assume no plotting

`noDesignExit` logical indicating if the function should abort if no design file is found

`weightSex.loess` percentage of sex chromosomes to use

`excludeSex.smooth` logical indicating if sex chromosome spots should be excluded from smoothing

`excludeSex.design` logical indicating if sex chromosome spots should be excluded from design phase

`weightNonMap.loess` percentage of non mapped samples to use

`excludeNonMap.smooth` logical indicating if non-mapped chromosome spots should be excluded from smoothing

`excludeNonMap.design` logical indicating if non-mapped chromosome spots should be excluded from design phase

`ilambda.default`
: which lambda should be stored as default Mxy

`saveAll`
: logical indicating if all data should be saved

`plt`
: logical indicating if plotting should occur

`vrb`
: a logical flag indicating if status messages should be printed

Details

This function will load in the specified array.image file and associated map and design files if required. It extracts the raw signal mean and background signal mean from the array.image object to use in calculations.

The `BCoption` allows for background correction. This will specify if no correction should take place ("none"), if the raw background values should be used in correction ("raw"), or if the smoothed background values should be used in teh correction ("smoothed")

The percentage of background correction that should take place is specified as `lambda`. The function will validate which lambda yields the best result. The percentage of background correction that should take place and the validation method to prove what maxmimal lamdba should be are open ended questions in the field on bioinformatics. We are currently investigating such matters and, if background correction is selected, will implement a method to choose the best lambda for the dataset. Currently the function will default back to no background correction.

exclude.rule:

This may be any logical equation that will reference the dimension of data points equal to the dimensions of the signal mean matrix. The defualt is based on the Imagene data. It assumes that the data field flag was loaded in during the map build and/or map.array.images. It uses the following labeling: 0 - unflagged 4 - polymorphic 5 - High degree of Segmental duplication -1 - No reliable mapping information (either no data on UCSC or we excluded it because it failed fingerprinting) The defualt, "array.images$matrix$cy3$Flag$! $= 0$", $therefore will only grab the unflagged spots.$

Value

The array.image file is updated to include an object `output.label`, default `"smooth2D"`, which holds function settings and smoothed data.

If `plt` is True, a postscript file of all plots for the run is created in a plot directory. These plots include chip images (heatmaps) and log ratio profiles.

Note

This function requires DNAcopy and fields packages. This function operates on an array.image object and requires a map be built if residuals are to be used. We will be enhancing this function in the near future to run on an aCGHroster (over multiple samples.) Also, the question of determining percentage of background correction is an open ended question we are investigating. Currently the function will only use no background correction. Adaptation to use 'maximal' background correction will eventually be included.

Author(s)

Lori Shepherd

References

See Also

SmoothBatch,Smooth.Image.CV,create.array.images, map.array.images, smooth2D.papply,D

Examples

```
  library(aCGHplus)
  Write.aCGH.ex1()

# a mapping information file must have been built
# this will build a mapping information file
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
        map.label="HB19Kv2.HG18",
       spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
       loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
       Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
           "chr7","chr8","chr9","chr10","chr11","chr12",
           "chr13","chr14","chr15","chr16","chr17","chr18",
           "chr19","chr20","chr21","chr22","chrX","chrY"),
       chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
       map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
       rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
       YchromNum = 24, centromere.file =  "Centromeres.txt",
       centromere.loc.lbl="location", centromere.file.sep="\t")


# a sample's array image must have been created
# this will build an array image file for sample1
  cyinfo="HB19Kv2-sample1"
  create.array.images(cyroot=cyinfo)

# with the build file and array image file we now can map sample 1
  map.array.images(
      fname.array.images="array.images/images/HB19Kv2-sample1.RData",
      mapping.data="RData/HB19Kv2.HG18.RData",
      vrb=TRUE)

 # with the array image file we now can create a design file for
 # sample1's chip design
  design.array.images(
      fname.array.images="array.images/images/HB19Kv2-sample1.RData",
      design.data = "HB19Kv2.design")



# now we can smooth data
SmoothArray(fname.array.images="array.images/images/HB19Kv2-sample1.RData",
            map.name=NA,thetas=seq(0.5,7.5,length=11), cvLevel=10,
            LossF="L1",useResid=TRUE,BCoption="none",lambdas=seq(0,1,length=2),
            DesignFlag=FALSE, lib.loc=NA,
            cyC.label="array.images$matrix$cy3$Signal.Mean",
            cyT.label="array.images$matrix$cy5$Signal.Mean",
            cyC.BC.label="array.images$matrix$cy3$Background.Mean",
            cyT.BC.label="array.images$matrix$cy5$Background.Mean",
            cy.grid.label="array.images$matrix$Grid",
            exclude.rule="array.images$matrix$cy3$Flag!=0",
```

```
            output.label="smooth2D",pname="auto",plt=FALSE,vrb=TRUE)

#this could have been run using the defaults with the call:
#SmoothArray(fname.array.images="array.images/images/HB19Kv2-sample1.RData",
#            LossF="L1", BCoption="none")
```

SmoothBatch *SMOOTH DATA IN BATCH*

Description

This function will smooth the log2 ratios of a sample using a cross validation-like technique in batch

Usage

```
SmoothBatch(aCGHroster,
                      overwrite=FALSE,
                      BatchDX=NA,
                      nbatchjobs=2,
                      ibatchjob=1,
                      time.order=FALSE,
                      map.name=NA,
                      thetas=seq(0.5,7.5,length=11),
                      cvLevel=10,
                      LossF=c("L1","L2"),
                      useResid=TRUE,
                      BCoption=c("none","raw","smoothed"),
                      lambdas=seq(0,1,length=2),
                      DesignFlag=FALSE,
                      DesignList=c("Plate","Pin","PlateRow","PlateCol","Repetitio
                      lib.loc=NA,
                      cyC.label="array.images$matrix$cy3$Signal.Mean",
                      cyT.label="array.images$matrix$cy5$Signal.Mean",
                      cyC.BC.label="array.images$matrix$cy3$Background.Mean",
                      cyT.BC.label="array.images$matrix$cy5$Background.Mean",
                      cy.grid.label="array.images$matrix$Grid",
                      exclude.rule="array.images$matrix$cy3$Flag!=0",
                      output.label="smooth2D",
                      pname="auto",
                      noDesignExit=FALSE,
                      weightSex.loess=1/10,
                      excludeSex.smooth=FALSE,
                      excludeSex.design=FALSE,
                      weightNonMap.loess=1/10,
                      excludeNonMap.smooth=TRUE,
                      excludeNonMap.design=TRUE,
                      ilambda.default=1,
                      saveAll=TRUE,
                      plt=TRUE,
                      vrb=TRUE)
```

Arguments

aCGHroster	aCGH roster object
overwrite	Flag to overwrite existing array.images
BatchDX	index of roster samples to be run in batch
nbatchjobs	number of batch jobs
ibatchjob	the batch job tun in this call of SmoothBatch
time.order	logical indicating if an ordering of samples by time stamp should occur
map.name	name of map file, if NA uses default in array.images
thetas	a sequence of numbers to be tested as the smoothing value, function will use whichever theta yields best result. This is passed as argument into Smooth.Image.CV. The default is a sequence of 11 values from .5 to 7.5
cvLevel	cross validation level, number indicating percentage of data points to remove before smoothing and use as a reference check. Default is 10, therefore 1/10 of the data will be removed and used as validation of method. This is passed as an argument into Smooth.Image.CV
LossF	Must be either "L1" or "L2", choice of which data from Smooth.Image.CV to use. "L1" represents using the sum of the absolute difference of the observed values and smoothed values. "L2" represents using the sum of the difference of oberved values and the smoothed values squared.
useResid	logical flag indicating if residuals are to be used
BCoption	option whether to background correct. Must be either "none", "raw", or "smoothed"
lambdas	sequence of numbers represeting the percentage of background correction to use, function will use whichever lambda yields best result. The default currently checks 0 (none) and 1(100%)
DesignFlag	logical indicating if the Design file for the array.image should be loaded
DesignList	List of the names of columns in the design file that should be corrected
lib.loc	path name to local library, Default, NA, assumes working in main R directory. DNAcopy and fields must be installed in the loc.lib library.
cyC.label	path/name to the sample's control signal mean matrix, the default is "array.images$matrix$cy3$Signal
cyT.label	path/name to the sample's tumor signal mean matrix, the default is "array.images$matrix$cy5$Signal.
cyC.BC.label	path/name to the sample's control background signal mean matrix in the array.image object, the default is "array.images$matrix$cy3$Background.Mean"
cyT.BC.label	path/name to the sample's tumor background signal mean matrix in the array.image object, the default is "array.images$matrix$cy3$Background.Mean"
cy.grid.label	path/name to the array.image's grid matrix, the default is "array.images$matrix$Grid"
exclude.rule	will indicate a rule for exclusion of points. The default is "array.images$matrix$cy3$Flag!=0" (see details).
output.label	name of the object to be added to the array.image object, Default is "smooth2D"
pname	a name to be used for a file containing the plot outputs of the function, the default, "auto" will plot in a directory plots/ with the name SmoothArray.[the root name of the array.image file]. If the user enters a blank field, "", the function will assume no plotting
noDesignExit	logical indicating if the function should abort if no design file is found

`weightSex.loess`
: percentage of sex chromosomes to use

`excludeSex.smooth`
: logical indicating if sex chromosome spots should be excluded from smoothing

`excludeSex.design`
: logical indicating if sex chromosome spots should be excluded from design phase

`weightNonMap.loess`
: percentage of non mapped samples to use

`excludeNonMap.smooth`
: logical indicating if non-mapped chromosome spots should be excluded from smoothing

`excludeNonMap.design`
: logical indicating if non-mapped chromosome spots should be excluded from design phase

`ilambda.default`
: which lambda should be stored as default Mxy

`saveAll`
: logical indicating if all data should be saved

`plt`
: logical indicating if plotting should occur

`vrb`
: a logical flag indicating if status messages should be printed

Details

This function will load all samples specified in the aCGHroster object and BatchDX. If BatchDX is not specified or is NA, all the samples are loaded.

This function will split the inventory file based on the number of batch jobs to be run (nbatchjobs). Every nbatchjob element is in one run of the batch. Therefore, for nbatchjobs=2 every other element is in one set, nbatchjobs=3 every third element is in one set, and so on.

ibatchjob specifies which run the batch is performing. If nbatchjobs=2 this may be either 1 or 2; If nbatchjobs=3, this may be 1,2,or 3. This is an indicator of which element to begin; If nbatchjobs=3, ibatchjob=1, every third sample beginning with the first will be in the set. If nbatchjobs=2, ibatchjob=2, every third sample beginning with the second will be in the set; and so on.

If `time.order` is true, sample's are ordered by their run time before they are divided into batch groups.

There is a call to SmoothArray:

This function will load in the specified array.image file and associated map and design files if required. It extracts the raw signal mean and background signal mean from the array.image object to use in calculations.

The `BCoption` allows for background correction. This will specify if no correction should take place ("none"), if the raw background values should be used in correction ("raw"), or if the smoothed background values should be used in teh correction ("smoothed")

The percentage of background correction that should take place is specified as `lambda`. The function will validate which lambda yields the best result. The percentage of background correction that should take place and the validation method to prove what maxmimal lamdba should be are open ended questions in the field on bioinformatics. We are currently investigating such matters and, if background correction is selected, will implement a method to choose the best lambda for the dataset. Currently the function will default back to no background correction.

exclude.rule:

This may be any logical equation that will reference the dimension of data points equal to the dimensions of the signal mean matrix. The defualt is based on the Imagene data. It assumes that the data field flag was loaded in during the map build and/or map.array.images. It uses the following labeling: 0 - unflagged 4 - polymorphic 5 - High degree of Segmental duplication -1 - No reliable mapping information (either no data on UCSC or we excluded it because it failed fingerprinting) The defualt, "array.images$matrix$cy3$Flag! = 0$", $therefore will only grab the unflagged spots.$

Value

The array.image file is updated to include an object `output.label`, default `"smooth2D"`, which holds function settings and smoothed data. This occurs with the call to SmoothArray.

If `plt` is True, a postscript file of all plots for the run is created in a plot directory. These plots include chip images (heatmaps) and log ratio profiles.

Note

It is possible to run a single job on the aCGHroster object. This is done by setting nbatchjobs and ibatchjob to 1 (see example below)

SmoothArray: This function requires DNAcopy and fields packages. This function operates on an array.image object and requires a map be built if residuals are to be used. We will be enhancing this function in the near future to run on an aCGHroster (over multiple samples.) Also, the question of determining percentage of background correction is an open ended question we are investigating. Currently the function will only use no background correction. Adaptation to use 'maximal' background correction will eventually be included.

Author(s)

Lori Shepherd

References

See Also

`SmoothArray`, `smooth2D.papply`, `DiagPlot.Smooth2D`

Examples

```
# please note these examples will take several minutes to run

library(aCGHplus)

# creates directories and files needed for running example
Write.aCGH.ex1()

# a build file for each of the chip designs used in analyzing the
# images must be built. For our example this is the HB19K_v2_HG18
# this builds that mapping information file
 buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
          map.label="HB19Kv2.HG18",
         spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
         loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
```

```
          Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
              "chr7","chr8","chr9","chr10","chr11","chr12",
              "chr13","chr14","chr15","chr16","chr17","chr18",
              "chr19","chr20","chr21","chr22","chrX","chrY"),
          chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
          map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
          rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
          YchromNum = 24, centromere.file =  "Centromeres.txt",
          centromere.loc.lbl="location", centromere.file.sep="\t")


# this will create an aCGHroster object off the example inventory file
# aCGH.ex1.inv
aCGHroster=create.roster.R(array.dir="arrays",
             roster.file="RData/aCGHroster.RData",
             inventory.file="aCGH.ex1.inv", image.dir="array.images",
             inv.sep=",", overwrite=FALSE, returnFlag=TRUE, BatchDX=NA, vrb=TRUE)

# this will run a single batch job
SmoothBatch(aCGHroster=aCGHroster,
            overwrite=FALSE, BatchDX=NA, nbatchjobs=1,
            ibatchjob=1, map.name=NA, thetas=seq(0.5,7.5,length=11),
            cvLevel=10, LossF="L1", useResid=TRUE, BCoption="none",
            lambdas=seq(0,1,length=2), DesignFlag=FALSE, lib.loc=NA,
            cyC.label="array.images$matrix$cy3$Signal.Mean",
            cyT.label="array.images$matrix$cy5$Signal.Mean",
            cyC.BC.label="array.images$matrix$cy3$Background.Mean",
            cyT.BC.label="array.images$matrix$cy5$Background.Mean",
            cy.grid.label="array.images$matrix$Grid",
            exclude.rule="array.images$matrix$cy3$Flag!=0",
            output.label="smooth2D", pname="auto", plt=TRUE, vrb=TRUE)




# or run in batch



library(aCGHplus)

# This loads pre-existing aCGHroster object for convience
Write.aCGH.ex2()
load("RData/aCGHroster.RData")

# In One R Session:
# this section will smooth half of the samples,
# since this is the first of two jobs,(nbatchjobs=2,ibatchjob=1)
# it will smooth the 1,3, and 5 samples

SmoothBatch(aCGHroster,overwrite=FALSE,BatchDX=NA, nbatchjobs=2,
                     ibatchjob=1,map.name=NA,thetas=seq(0.5,7.5,length=11),
                     cvLevel=10,LossF="L1",useResid=TRUE,BCoption="none",
                     lambdas=seq(0,1,length=2),DesignFlag=FALSE,lib.loc=NA,
                     cyC.label="array.images$matrix$cy3$Signal.Mean",
                     cyT.label="array.images$matrix$cy5$Signal.Mean",
```

```
                      cyC.BC.label="array.images$matrix$cy3$Background.Mean",
                      cyT.BC.label="array.images$matrix$cy5$Background.Mean",
                      cy.grid.label="array.images$matrix$Grid",
                      exclude.rule="array.images$matrix$cy3$Flag!=0",
                      output.label="smooth2D", pname="auto",plt=FALSE,vrb=TRUE)

# In Second R Session:
# in another session or at a later date, the other half may be run
# this is the second of two jobs, (nbatchjobs=2,ibatchjob=2)
# it will take the 2,4, and 6 samples
SmoothBatch(aCGHroster,overwrite=FALSE,BatchDX=NA, nbatchjobs=2,
                      ibatchjob=2,map.name=NA,thetas=seq(0.5,7.5,length=11),
                      cvLevel=10,LossF="L1",useResid=TRUE,BCoption="none",
                      lambdas=seq(0,1,length=2),DesignFlag=F,lib.loc=NA,
                      cyC.label="array.images$matrix$cy3$Signal.Mean",
                      cyT.label="array.images$matrix$cy5$Signal.Mean",
                      cyC.BC.label="array.images$matrix$cy3$Background.Mean",
                      cyT.BC.label="array.images$matrix$cy5$Background.Mean",
                      cy.grid.label="array.images$matrix$Grid",
                      exclude.rule="array.images$matrix$cy3$Flag!=0",
                      output.label="smooth2D", pname="auto",plt=FALSE,vrb=TRUE)
```

`Smooth.Image.CV` *SMOOTH DATA BY CROSS-VALIDATION*

Description

This function uses a cross validation technique to smooth values.

Usage

```
Smooth.Image.CV(rawmat,
                      cvLevel=10,
                      thetas=seq(0.5,2.5,length=21),
                      flagmat=NA,
                      plt=FALSE,
                      vrb=TRUE)
```

Arguments

`rawmat`	matrix of values to perform cross validation on
`cvLevel`	cross validation level, number indicating percentage of data points to remove before smoothing and use as a reference check. Default is 10, therefore 1/10 of the data will be removed and used as validation of method.
`thetas`	a sequence of numbers to be tested as the smoothing value, function will use whichever theta yields best result. The default is a sequence of 21 values from .5 to 2.5
`flagmat`	indicates values that should be excluded from calculations. This must be of equal length to the rawmat passed in as argument in order for the exclusion of values to take place.
`plt`	logical indicating if plotting should occur
`vrb`	a logical flag indicating if status messages should be printed

Details

This function utilizes the `smooth.2d` function of the fields library. It will pass the list of thetas to the smooth.2d function and decided which theta is best for two alternatives: 1. minimizing the the sum of the absolute difference of the observed values and smoothed values 2. minimizing the sum of the difference of oberved values and the smoothed values squared This data and associated thetas are returned in a list.

Value

returns an object of type list containing the values and data for minimzing the sum of the absolute difference of the observed values and smoothed values, and minimizing the sum of the difference of oberved values and the smoothed values squared.

Note

required package: fields

Author(s)

Lori Shepherd

References

See Also

`SmoothArray`

Examples

```
## this assumes create.array.images has already been called
## see/run SmoothArray example if unsure

# To ensure data is present we will load example data
Write.aCGH.ex2()

load("array.images/images/HB19Kv2-sample1.RData")
BCmat=array.images$matrix$cy5$Background.Mean

cyT.BCsmoothed = Smooth.Image.CV(rawmat=BCmat,cvLevel=10,
                                 thetas=seq(0.5,2.5,length=21),
                                 flagmat=NA, plt=FALSE, vrb=TRUE)

 # this could have also been run using the defaults as
 # Smooth.Image.CV(BCmat)
```

`SNR.interactive` *INTERACTIVE VIEWER UTILIZING SIGNAL TO NOISE CALCULATIONS*

Description

This function plots signal to noise and allows the user to click on any sample to view genomic plot, and cy3 and cy5 heatmap chip images.

Usage

```
SNR.interactive(aCGH,
                heatmaps=TRUE,
                recenter=FALSE
                )
```

Arguments

`aCGH`	an aCGH object
`heatmaps`	logical indicating if heatmap images of cy3 and cy5 chip should be displayed
`recenter`	logical indicating if aCGH object should be recentered

Details

This function will first calculate the signal to noise ratio for each sample if it has not already been calculated. Two to four different windows will appear. If heatmaps is true, two blank windows will initially appear. There will also be one window with an empty genomic plot and a window with a plot of the signal to noise ratios vs. noise of the sample. Each sample may be selected to view the genomic profile. If heatmaps is ture the cy3 and cy5 chip images will also appear for the sample selected.

To exit close out all (two to four) windows.

Value

user interactive plot

Note

This function assumes circulary binary segmentation has been performed and that log2.ratios were not overwritten with fitted values.

Author(s)

Lori Shepherd

References

See Also

GetSNR

Examples

```
# because this is a user interactive function the following code has
# been commented out

#    load("RData/aCGH.RData")
#    SNR.interactive(aCGH)
```

SpotsAcrossSamples *CREATE STATIC PLOTS TO TRACK SPOT/BACs ACROSS SAMPLES*

Description

These functions will create static plots to track spotIDs/BACs across a given list of samples. The graphs will show raw and fitted log2 ratios and circle the specified spot[s]/BAC[s]

Usage

```
spotsAcrossSamples(aCGH,
                   spotIDX=NA,
                   sampleDX=NA,
                   output.file="spotsAcrossSamples",
                   highlight.color="blue",
                   raw.color="gray",
                   fit.color="red",
                   vrb=TRUE,
                   cex.pts=.5,
                   ...)

aSpotAcrossSamples(aCGH,
                   spotIDX,
                   sampleDX=NA,
                   output.file= NA,
                   highlight.color="blue",
                   raw.color="gray",
                   fit.color="red",
                   vrb=TRUE,
                   cex.pts=.5,
                   ...)
```

Arguments

aCGH	an aCGH object
spotIDX	indcates which spot.ID/BAC to highlight. may be a vector in spotsAcrossSamples and a single value in aSpotAcrossSamples. This may be numeric corresponding to the order in the aCGH object or a character specific spot.ID/BAC name
sampleDX	a numeric vector indicating which samples should be used
output.file	base name of the output file. This file will be created inside the plots directory. The default for spotsAcrossSamples is spotsAcrossSamples.ps/pdf and the default for aSpotAcrossSamples is spot[spot.ID/BAC name].ps/.pdf
highlight.color	color for highlighting selected spot.IDs/BACs
raw.color	color for the raw log2 ratios
fit.color	color for the fit log2 ratios
vrb	a logical flag indicating if status messages should be printed
cex.pts	The size of the plotted points
...	additional arguments to be passed into GenomeOneRow

Details

These functions will output static plots to a file in the plot directory.

spotsAcrossSamples will take an aCGH object, a sample index, and a spot/BAC index. A genomic plot of each specified sample will be graphed with the specified spots from the spotDX highlight. This allows the user to track spots through different samples. The spotIDX may be either numeric or character. Numeric would be associated with its position in the aCGH object; Character would be the exact names of spots/BACs in question. The user may change different attributes of the graphs such as colors, point size, ect.

aSpotAcrossSamples is similar to spotsAcrossSample. A single spotIDX is specified and may be numeric or character. Again, numeric would be associated with its position in the aCGH object and character would be the exact names of spots/BACs in question. If more than one spotIDX is given, only the first entry is used. A single spot is hightlighted over the specified sample index of an aCGH object. The user may change different attributes of the graphs such as colors, point size, ect. Three graphs are drawn per sample: a genomic plot, a plot of the entire arm location of the specified spot, and a plot of the fine.band location of the specified spot.

Value

creates static plots in the plot directory

Note

uses GenomeOneRow

Author(s)

Lori Shepherd

References

See Also

create.GenomeOneRow,addTo.GenomeOneRow, make.aCGH

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")


# This will create the file spotsAcrossSamples.ps/.pdf which contains
# a graph of the first three samples with the 200th, 4000th, 10000th,
# 12000th and 15000th spots highlighted

spotsAcrossSamples(aCGH, sampleDX = c(1:3), spotIDX = c(200,4000, 10000, 12000, 15000))

# This will create the file spotRP11-91A17.ps/.pdf which contains graphs
# of the genome, arm and fine.band for the first three samples in the
# aCGH obect highlighting the 4000 spot "RP11-91A17"

aSpotAcrossSamples(aCGH, spotIDX= "RP11-91A17", sampleDX = c(1:5))
# same as running aSpotAcrossSamples(aCGH, spotIDX=4000, sampleDX = c(1:5))
```

---

subsetACGH *SUBSET AN aCGH OBJECT*

Description

This function will subset an aCGH object based on a given set of samples

Usage

```
subsetACGH(aCGH,
           smpls,
           vrb=TRUE,
           saveFile=TRUE,
           fileName="RData/subsetACGH.RData")
```

Arguments

aCGH	an aCGH object
smpls	a numeric list of which samples should be kept for subsetting purposes
vrb	logical if status messages should be printed
saveFile	logical indicating if the new subset object should be saved
fileName	if saveFile, file path/name to save to

Details

This function will take in an aCGH object and a sample index, and subsets the aCGH object. The resulting object will be a smaller aCGH object consisting only of the selected samples. Only objects created during the package's load-in, smoothing, or CBS process will be subset. If vrb, a list of subset objects and non subset objects will be given.

Value

a new (smaller) aCGH object consisting of selected samples

Note

This function only subsets default objects of the aCGH (those created during loadin, smoothing, and CBS if applicable). If additional columns or functions were run that added data to the object, this data will have to be subset and added manually by the user. If vrb the functio will list any objects not taken as a subset

Author(s)

Lori Shepherd

References

See Also

`make.aCGH`, `getFailed`, `subsetByRule`

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# creates a smaller aCGH object consisting of the first and third samples
aCGH = subsetACGH(aCGH, smpls=c(1,3))
```

`subsetByRule` *SUBSET AN aCGH OBJECT*

Description

This function will subset an aCGH object based on rule for inverntory data

Usage

```
subsetByRule(aCGH,
             rule="(aCGH$inventory$PCA.score >=0) & (aCGH$inventory$PCA.excluded=
             fileName="RData/PCA.subset.ACGH.RData")
```

Arguments

aCGH	An aCGH object
rule	Rule for subseting inventory
fileName	fiel to save new subset aCGH object

Details

This function will take in an aCGH object and a rule for subsetting the aCGH inventory. All samples that are retrieved by the rule will be kept as a new aCGH object.

This was designed with the PCA interactive viewer in mind. When using the PCA interactive viewer it is possible to add scoring information, remove samples, or flag samples for removal to the existing aCGH inventory. We though it would be useful to be able to subset samples that for instance were given a low score, perhaps for quality control purposes.

Value

aCGH

Note

This function was made for use after the interactive PCA viewer. see runPCA It can however be used to subset on any rule given for an aCGH object inventory This is a wrapper on the subsetACGH function

Author(s)

Lori Shepherd

See Also

subsetACGH, makePCAfile, runPCA

Examples

```
# We make sure an aCGH object has already been created and is in memory
# by loading example data and loading the aCGH object
#   see make.aCGH

Write.aCGH.ex2()
load("RData/aCGH.RData")

# subsets based on orientaiton. all dye flipes are used
aCGH = subsetByRule(aCGH, rule="aCGH$inventory$orientation == 1",
fileName="RData/DyeFlips.subset.RData")
```

`vec2image` *CONVERT A VECTOR TO A MATRIX*

Description

This function takes a vector and an array image map file, and converts the vector into a matrix using the mappings from the map file.

Usage

```
vec2image(vec.val,
                  map.images,
                  flag.vec=NA)
```

Arguments

`vec.val`	vector to be mapped to image
`map.images`	associated array image map to use for mapping
`flag.vec`	logical indicating values to exclude, T means to exclude

Details

Value

out a vector of values

Note

Author(s)

Lori Shepherd

References

See Also

`map.array.images`

Examples

```
 library(aCGHplus)
 Write.aCGH.ex1()

 # this builds a map build
buildMap(chrom.band.file = "HB19Kv2.HG18.map.csv",
         map.label="HB19Kv2.HG18",
        spot.ID.lbl="Clone",loc.start.lbl="start", loc.center.lbl="Center",
        loc.stop.lbl="Stop", Chrom.lbl="Chromosome", Fine.Band.lbl="Band",
        Chrom.labels= c("chr1","chr2","chr3","chr4","chr5","chr6",
            "chr7","chr8","chr9","chr10","chr11","chr12",
            "chr13","chr14","chr15","chr16","chr17","chr18",
            "chr19","chr20","chr21","chr22","chrX","chrY"),
        chrom.band.file.sep=",", mapped.by.lbl="Mapped.by",
        map.flag.lbl="Flag", random.lbl=NA,genome.loc.lbl="g\_loc",
        rm.spotID.lbls=c("EMPTY","H2O"), maxnChrom = 1:24, XchromNum = 23,
        YchromNum = 24, centromere.file =  "Centromeres.txt",
        centromere.loc.lbl="location", centromere.file.sep="\t")

 # this will build an array image file for sample1
      cyinfo="HB19Kv2-sample1"
      create.array.images(cyroot=cyinfo, load.specs="Imagene.load.specs.csv",
                  array.dir="arrays", image.dir="array.images",
                  design.name=NA, image.soft=NA, vrb=TRUE)

  # with the build file and array image file we now can map sample 1
   map.array.images(
       fname.array.images="array.images/images/HB19Kv2-sample1.RData",
       mapping.data="RData/HB19Kv2.HG18.RData",
       vrb=TRUE)

  # loads sample1's array image
  load("array.images/images/HB19Kv2-sample1.RData")

  # load in associated array image map
  mapName = array.images$info$full.map.name
  load(mapName)

  # takes the vector chrom with all chromosomes
  # maps to a matrix representing chip position based on map file

  chrom = map.images$mapping.info$iChrom
  xyChrom = vec2image(vec.val= chrom, map.images=map.images, flag.vec=NA)
```

`vizMeanPrep` *STORES SPECIFIED FACTOR INFORMATION FOR AN aCGH OBJECT'S INVENTORY*

Description

This function will create a new aCGH object file which stores the aCGH object and additional factor information for use in the mean across samples plots

Usage

```
makeFactorObject(aCGHloaded,
                 invloaded,
                 invFile = NA,
                 aCGHfile = "RData/aCGH.RData" ,
                 sep="\t",
                 factorBy=NA,
                 factors=NA,
                 autoSave = "RData/vizMean.RData")
```

Arguments

`invloaded`	logical indicating whether the inventory file for aCGH object has already been loaded
`aCGHloaded`	logical indicating whether the file containing the aCGH object is loaded and the aCGH object is in memory
`invFile`	full path to the inventroy file that will be loaded to aCGH object if invloaded is F
`aCGHfile`	path name of file to load aCGH object if aCGHloaded is F
`sep`	seperation character used to parse inventory file when loadInv is called, passed into a read.table call
`factorBy`	name of column in the aCGH inventory to factor by
`factors`	character vector of the names of specfic levels to factor by with respect to the column factorBy
`autoSave`	path name of the file to save containing aCGH and factor information

Details

`makeFactorObject` will load factor information to an aCGH object. The file will load an existing aCGH file or use an aCGH object already loaded. If the inventory file is not loaded, the file will load the inventory file specified. FactorBy indicates which inventory column to factor by. If this is left unspecified, the aCGH object will be factored using a unique factor "GroupA" that will include all samples. The user may also indicate specific levels to be included when factoring. A new file is storing the aCGH object and factor information.

Value

A file is saved as autoSave containing aCGH and factor information, and aCGH object returned

Note

Author(s)

Lori Shepherd

References

See Also

meanPlot, factor, initFactor

Examples

```
# We make sure an aCGH object has been created by loading example data
# see make.aCGH
Write.aCGH.ex2()


aCGH = makeFactorObject(aCGHloaded = FALSE, invloaded = TRUE, factorBy = "sex")

# This is the same as running:
# makeFactorObject(aCGHloaded = FALSE,invloaded = TRUE,invFile = NA,
#                  aCGHfile = "RData/aCGH.RData" , sep="\t",
#                  factorBy="sex",factors=NA,
#                  autoSave = "RData/vizMean.RData")
# or
#  load("RData/aCGH.RData")
#  makeFactorObject(aCGHloaded=TRUE, invloaded = TRUE, factorBy="sex")
```

---

writeExample — *DATA AND DIRECTORIES NEEDED FOR RUNNING LOAD-IN AND PROCESSING EXAMPLES*

Description

This function will create needed directories and files for running all examples in help files and documentation for the load-in and processing functions

Usage

```
Write.aCGH.ex1(target.dir="")

Write.aCGH.ex2(target.dir="", vrb=TRUE)
```

Arguments

target.dir	name of directory to write files
vrb	logical indicating if status messages should be printed

Details

aCGHplus requires some files to be present for certain functions to run, as well as some specified directories.

`Write.aCGH.ex1()` loads a needed data file, creates the needed directories arrays and RData, provides 3 samples, and creates the needed files Centromeres.txt, HB19Kv2.HG18.map.csv, Imagene.load.specs.csv, aCGH.ex1.inv, and HB19Kv2.design. These files will be used to demonstrate load-in functions for the package.

`Write.aCGH.ex2()` loads in .RData files. These represent files that would have been created during the load in process. This set provides 6 samples' array.image files, map files, and design files. It also provides an aCGHroster file for the 6 samples and the aCGH file.

Note

Author(s)

Lori Shepherd

References

See Also

Examples

```
library(aCGHplus)
Write.aCGH.ex1()

library(aCGHplus)
Write.aCGH.ex2(target.dir="seeEx2")
```

Index