ADaCGH2

March 24, 2012

`inputDataToADaCGHData`
Convert CGH data to ff data frames

Description

An input data frame with CGH data is converted to several ff files and data checked for potential errors and location duplications.

Usage

```
inputDataToADaCGHData(ffpattern = paste(getwd(), "/", sep = ""),
                      MAList = NULL,
                      cloneinfo = NULL,
                      filename = NULL,
                      sep = "\t",
                      quote = "\"",
                      na.omit = FALSE,
                      minNumPerChrom = 10)
```

Arguments

`ffpattern`	See argument `pattern` in `ff`. The default is to create the "ff" files in the current working directory.
`MAList`	The name of an object of class `MAList` (`as.MAList`) or `SegList` (e.g., `dim.SegList`). See vignnettes for these packages for details about these objects.
`cloneinfo`	A character vector with the full path to a file that conforms to the characteristis of `file` in function `read.clonesinfo` (see details in the vignette) or the name of a data frame with at least a column named "Chr" (with chromosomal informtaion) and "Position".
`filename`	Name of data RData file that contains the data frame with original, non-ff, data. Note: this is the name of the RData file (possibly including path), NOT the name of the data frame. The first three columns of the data frame are the IDs of the probes, the chromosome number, and the position, and all remaining columns contain the data

for the arrays, one column per array. The names of the first three column do not matter, but the order does. Names of the remaining columns will be used if existing; otherwise, fake array names will be created.

`sep` Argument to `read.table` if reading a `cloneinfo` file.

`quote` Argument to `read.table` if reading a `cloneinfo` file.

`na.omit` Omit NAs? If there are NAs and na.omit is set to FALSE, the function will stop with an error.

`minNumPerChrom`
If any chromosome has fewer observations than minNumPerChrom the function will fail. This can help detect upstream pre-processing errors.

Details

If there are identical positions (in the same chromosome) a small random uniform variate is added to get unique locations.

Commented examples of reading objects from **limma** and **snapCGH** are provided in the vignnette.

Value

This function is used mainly for its side effects: writing several ff files to the current working directory (the actual names are printed out).

In addition, and since we need to manipulate the complete set of original data, the return value is a data frame that is could be used later to speed up certain calculations. Right now, however, this is not used for anything, except for information purposes. This table is similar to a dictionary or hash table. This data frame has (number of arrays * number of chromosomes) rows. The columns are

`Index` The integer index of the entry, 1:number of arrays * number of chromosomes

`ArrayNum` The array number

`Arrayname` The name of the array

`ChromNum` The chrosome number

`ChromName` The chromosome name. Yes, chromosome must be numeric, but the values of ChromNum form a set of integers starting at one and going up to the total number of different chromosomes. E.g., if you only have two chromosomes, say 3 and 22, ChromNum contains values 1 and 2, whereas ChromName contains values 3 and 22.

`posInit` The first position (in a vector ordered from 1 to total number of probes, with probes ordered by chromosome and position within chromosome) of a probe of this chromosome.

`posEnd` The last position of a probe of this chromosome.

Note

Converting a very large data set into a set of ff files can be memory consuming. Since this function is mainly used for its side effects (leaving the ff files in the disk), it can be run in a separate process that will then be killed. See an example below using **multicore**. (For the example you must install **multicore**).

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

Examples

```
## Create a temp dir for storing output.
## (Not needed, but cleaner).
dir.create("ADaCGH2_example_input_dir")
originalDir <- getwd()
setwd("ADaCGH2_example_input_dir")
Sys.sleep(1)

## Get location (and full filename) of example data file
fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx1")

tableChromArray <- inputDataToADaCGHData(filename = fname)



### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")


### Running in a separate process
### This example only does anything if you have multicore installed.
  if(require(multicore)) {
  parallel(inputDataToADaCGHData(filename = fname), silent = FALSE)
  tableChromArray <- collect()[[1]]
  if(inherits(tableChromArray, "try-error")) {
    stop("ERROR in input data conversion")
  }
### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")
}

### Try to prevent problems in R CMD check
Sys.sleep(2)
```

```
### Delete temp dir
setwd(originalDir)
Sys.sleep(2)
unlink("ADaCGH2_example_input_dir", recursive = TRUE)
Sys.sleep(2)
```

`inputEx1` *A fictitious aCGH data set*

Description

A fictitious aCGH data set.

Usage

```
inputEx1
```

Format

A data frame with 4323 rows and 6 columns; the last three correspond to the aCGH data for three samples.

Source

Simulated data

`inputEx2` *A fictitious aCGH data set*

Description

A fictitious aCGH data set.

Usage

```
inputEx2
```

Format

A data frame with 452 rows and 6 columns; the last three correspond to the aCGH data for three samples.

Source

Simulated data

`outputToCGHregions` *ADaCGH2 output as input to CGHregions*

Description

Convert ADaCGH2 output to a data frame that can be used as input for `CGHregions`.

Usage

```
outputToCGHregions(output, directory = getwd())
```

Arguments

`output`	The name of the output from a call to a `pSegment` function.
`directory`	The directory where the initial data transformation and the analysis have been carried out. It is a lot better if you just work on a single directory for a set of files. Otherwise, unless you keep very carefull track of where you do what, you will run into trouble.

Value

A data frame of 4 + k columns that can be used as input to the `CGHregions` function. The first four columns are the probe name, the chromosome, the position and the position. The last k columns are the calls for the k samples.

Note

This function does NOT check if the calls are meaningfull. In particular, you probably do NOT want to use this function when `pSegment` has been called using '`merging = "none"`'.

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

See Also

`pSegment`

Examples

```
## Create a temp dir for storing output.
## (Not needed, but cleaner).
dir.create("ADaCGH2_cghreg_example_tmp_dir")
originalDir <- getwd()
setwd("ADaCGH2_cghreg_example_tmp_dir")
Sys.sleep(1)

snowfallInit(universeSize = 2, typecluster = "SOCK")

## To speed up R CMD check, we do not use inputEx1, but a much smaller
## data set. When you try the examples, you might one to use
## inputEx1 instead.
```

```
## Not run:

fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx1")

## End(Not run)

fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx2")


tableChromArray <- inputDataToADaCGHData(filename = fname)

hs_mad.out <- pSegmentHaarSeg("cghData.RData",
                           "chromData.RData", merging = "MAD")

forcghr <- outputToCGHregions(hs_mad.out)
if(require(CGHregions)) {
  regions1 <- CGHregions(forcghr)
  regions1
}

### Explicitly stop cluster
sfStop()


### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")

lapply(hs_mad.out, delete)
rm(hs_mad.out)

### Try to prevent problems in R CMD check
### (As a regular user, most likely you do not need this)
Sys.sleep(2)
detach("package:rlecuyer", unload = TRUE)


### Delete all files and temp dir
setwd(originalDir)
Sys.sleep(2)
unlink("ADaCGH2_cghreg_example_tmp_dir", recursive = TRUE)
Sys.sleep(2)
```

pChromPlot *Segment plots for aCGH as PNG*

Description

Produce PNG figures of egment plots (by chromosome) for aCGH segmentation results. Internal calls are parallelized for increased speed and we use ff objets to allow the handling of very large objects. The output can include files for creating HTML with imagemaps.

Usage

```
pChromPlot(outRDataName, cghRDataName, chromRDataName,
           probenamesRDataName,
           posRDataName = NULL,
           imgheight = 500,
           pixels.point = 3,
           pch = 20,
           colors = c("orange", "red", "green", "blue", "black"),
           imagemap = FALSE,
           ...)
```

Arguments

outRDataName	The Rdata file name that contains the ffdf with the results from the segmentation as carried out by any of the pSegment functions.
cghRDataName	The Rdata file name that contains the ffdf with the aCGH data. This file can be created using as.ffdf with a data frame with genes (probes) in rows and subjects or arrays in columns. Function inputDataToADaCGHData produces these type of files.
chromRDataName	The RData file name with the ff (short integer) vector with the chromosome indicator. Function inputDataToADaCGHData produces these type of files.
probenamesRDataName	The RData file name with the vector with the probe names. Function inputDataToADaCGHData produces these type of files. Note that this is not an ff file.
posRDataName	The RData file name with the ff double vector with the location (e.g., position in kbases) of each probe in the chromosome. Function inputDataToADaCGHData produces these type of files. Used for the spacing in the plots. If NULL, the x-axis goes from 1:number of probes in that chromosome.
imgheight	Height of png image. See png.
pixels.point	Approximate number of pixels that each point takes; this determines also final figure size. With many probes per chromosome, you will want to make this a small value.
pch	The type of plotting symbol. See par.
colors	A five-element character vector with the colors for: probes without change, probes that have a "gained" status, probes that have a "lost" status, the line that connects (smoothed values of) probes, the horizontal line at the 0 level.

imagemap	If FALSE only the png figure is produced. If TRUE, for each array * chromosome, to additional files are produced: "pngCoord_ChrNN@MM" and "geneNames_ChrNN@MM", where "NN" is the chromosome number and "MM" is the array name. The first file contains the coordinates of the png and radius and the second the gene or probe names, so that you can easily produce an HTML imagemap. (Former versions of ADaCGH did this automatically with Python. In this version we include the Python files under "imagemap-example".)
...	Additional arguments; not used.

Value

Used only for its side effects of producing PNG plots, stored in the current working directory (`getwd()`.)

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

See Also

pSegment

Examples

```
## Create a temp dir for storing output
dir.create("ADaCGH2_plot_tmp_dir")
originalDir <- getwd()
setwd("ADaCGH2_plot_tmp_dir")


## Start cluster
snowfallInit(universeSize = 2, typecluster = "SOCK")

## To speed up R CMD check, we do not use inputEx1, but a much smaller
## data set. When you try the examples, you might one to use
## inputEx1 instead.
## Not run:

fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx1")

## End(Not run)

fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx2")


tableChromArray <- inputDataToADaCGHData(filename = fname)

hs_mad.out <- pSegmentHaarSeg("cghData.RData",
                              "chromData.RData", merging = "MAD")

save(hs_mad.out, file = "hs_mad.out.RData", compress = FALSE)

pChromPlot(outRDataName = "hs_mad.out.RData",
```

```
                                cghRDataName = "cghData.RData",
                                chromRDataName = "chromData.RData",
                                posRDataName = "posData.RData",
                                probenamesRDataName = "probeNames.RData",
                                imgheight = 350)


## Not run:
## Produce the coordinate and probe names files.
pChromPlot(outRDataName = "hs_mad.out.RData",
                                cghRDataName = "cghData.RData",
                                chromRDataName = "chromData.RData",
                                posRDataName = "posData.RData",
                                probenamesRDataName = "probeNames.RData",
                                imgheight = 350,
                                imagemap = TRUE)

## End(Not run)

### PNGs are in this directory
getwd()

### Explicitly stop cluster
sfStop()

### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")

lapply(hs_mad.out, delete)
rm(hs_mad.out)
unlink("hs_mad.out.RData")

### Try to prevent problems in R CMD check
Sys.sleep(2)
## To prevent CMD check from crashing after cleanEx
detach("package:rlecuyer", unload = TRUE)


### Delete all png files and temp dir
setwd(originalDir)
Sys.sleep(2)
unlink("ADaCGH2_plot_tmp_dir", recursive = TRUE)
Sys.sleep(2)
```

`pSegment` *Parallelized/"unified" versions of several aCGH segementation algorithms/methods*

Description

These functions parallelize several segmentation algorithms or (for HaarSeg) make their calling use the same conventions as for other methods.

Usage

```
pSegmentDNAcopy(cghRDataName, chromRDataName, merging = "mergeLevels",
                mad.threshold = 3, smooth = TRUE,
                alpha=0.01, nperm=10000,
                p.method = "hybrid",
                min.width = 2,
                kmax=25, nmin=200,
                eta = 0.05,trim = 0.025,
                undo.splits = "none",
                undo.prune=0.05, undo.SD=3,
                ...)

pSegmentHaarSeg(cghRDataName, chromRDataName,
                merging = "MAD", mad.threshold = 3,
                W = vector(),
                rawI = vector(),
                breaksFdrQ = 0.001,
                haarStartLevel = 1,
                haarEndLevel = 5, ...)

pSegmentHMM(cghRDataName, chromRDataName,
            merging = "mergeLevels", mad.threshold = 3,
            aic.or.bic = "AIC", ...)



pSegmentBioHMM(cghRDataName, chromRDataName, posRDataName,
               merging = "mergeLevels", mad.threshold = 3,
               aic.or.bic = "AIC",
               ...)

pSegmentCGHseg(cghRDataName, chromRDataName, CGHseg.thres = -0.05,
               merging = "MAD", mad.threshold = 3, ...)


pSegmentGLAD(cghRDataName, chromRDataName,
             deltaN = 0.10,
             forceGL = c(-0.15, 0.15),
             deletion = -5,
             amplicon = 1,
```

```
                ...)

pSegmentWavelets(cghRDataName, chromRDataName, merging = "MAD",
                 mad.threshold = 3,
                 minDiff = 0.25,
                 minMergeDiff = 0.05,
                 thrLvl = 3, initClusterLevels = 10, ...)
```

Arguments

cghRDataName	The Rdata file name that contains the ffdf with the aCGH data. This file can be created using as.ffdf with a data frame with genes (probes) in rows and subjects or arrays in columns. Function inputDataToADaCGHData produces these type of files.
chromRDataName	The RData file name with the ff (short integer) vector with the chromosome indicator. Function inputDataToADaCGHData produces these type of files.
posRDataName	The RData file name with the ff double vector with the location (e.g., position in kbases) of each probe in the chromosome. Function inputDataToADaCGHData produces these type of files.
merging	Merging method; for most methods one of "MAD" or "mergeLevels". For CBS (pSegmentDNAcopy), GGHseg (pSegmentCGHseg), and Wavelets (as in Hsu et al. —pSegmentWavelets) also "none". This option does not apply to GLAD (which has its own merging-like approach). See details.
mad.threshold	If using merging = "MAD" the value such that all segments where abs(smoothed value) > m*MAD will be declared aberrant —see p. i141 of Ben-Yaacov and Eldar. No effect if merging = "mergeLevels" (or "none").
smooth	For DNAcopy only. If TRUE (default) carry out smoothing as explained in smooth.CNA.
alpha	For DNAcopy only. See segment.
nperm	For DNAcopy only. See segment.
p.method	For DNAcopy only. See segment.
min.width	For DNAcopy only. See segment.
kmax	For DNAcopy only. See segment.
nmin	For DNAcopy only. See segment.
eta	For DNAcopy only. See segment.
trim	For DNAcopy only. See segment.
undo.splits	For DNAcopy only. See segment.
undo.prune	For DNAcopy only. See segment.
undo.SD	For DNAcopy only. See segment.
W	For HaarSeg: Weight matrix, corresponding to quality of measurment. Insert 1/(sigma**2) as weights if your platform output sigma as the quality of measurment. W must have the same size as I.

`rawI` For HaarSeg. Mininum of the raw red and raw green measurment, before the log. rawI is used for the non-stationary variance compensation. rawI must have the same size as I.

`breaksFdrQ` For HaarSeg. The FDR q parameter. Common used values are 0.05, 0.01, 0.001. Default value is 0.001.

`haarStartLevel` For HaarSeg. The detail subband from which we start to detect peaks. The higher this value is, the less sensitive we are to short segments. The default is value is 1, corresponding to segments of 2 probes.

`haarEndLevel` For HaarSeg. The detail subband until which we use to detect peaks. The higher this value is, the more sensitive we re to large trends in the data. This value DOES NOT indicate the largest possible segment that can be detected. The default is value is 5, corresponding to step of 32 probes in each direction.

`aic.or.bic` For HMM and BioHMM. One of "AIC" or "BIC". See '`criteria`' in `runBioHMM`.

`CGHseg.thres` The threshold for the adaptive penalization in Picard et al.'s CGHseg. See p. 13 of the original paper. Must be a negative number. The default value used in the original reference is -0.5. However, our experience with the simulated data in Willenbrock and Fridlyand (2005) indicates that for those data values around -0.005 are more appropriate. We use here -0.05 as default.

`deltaN` Only for GLAD. See '`deltaN`' in `daglad`.

`forceGL` Only for GLAD. See '`forceGL`' in `daglad`.

`deletion` Only for GLAD. See '`deletion`' in `daglad`.

`amplicon` Only for GLAD. See '`amplicon`' in `daglad`.

`minMergeDiff` Used only when doing merging in the wavelet method of Hsu et al.. The finall call as to which segments go together is done by a `mergeLevels` approach, but an initial collapsing of very close values is performed (otherwise, we could end up passing to mergeLevels as many initial levels as there are points).

`minDiff` For Wavelets (Hsu et al.). Minimum (absolute) difference between the medians of two adjacent clusters for them to be considered truly different. Clusters "closer" together than this are collapsed together to form a single cluster.

`thrLvl` The level used for the wavelet thresholding in Hsu et al.

`initClusterLevels` For Wavelets (Hsu et al.). The initial number of clusters to form.

`...` Additional arguments; not used.

Details

In most cases, these are wrappers to the original code, with modifications for parallelization and for using `ff` objects. The functions will not work if you try to use them with the regular R data frames, matrices, and vectors.

We have parallelized all computations by array (in contrast to former versions of ADaCGH, where some computations, depending on number of samples, could be parallelized over array*chromosome).

CGHseg has been implemented here following the original authors description. Note that several publications incorrectly claim that they use the CGHseg approach when, actually, they are only using the "segment" function in the "tilingArray" package, but they are missing the key step of choosing the optimal number of segments (see p. 13 in Picard et al, 2005). We implement the author's method in our (internal, so use "ADaCGH2:::piccardsKO") function "piccardsKO".

For DNAcopy, BioHMM and HMM the smoothed results are merged, by default by the mergeLevels algorithm, as recommended in *Willenbrock and Fridlyand, 2005*. Merging is also done in GLAD (with GLAD's own merging algorithm). For HaarSeg, calling/merging is carried out using MAD, following page i141 of Ben-Yaacov and Eldar, section 2.3, "Determining aberrant intervals": a MAD (per their definition) is computed and any segment with absolute value larger than mad.threshold * MAD is considered aberrant. Merging is also performed for CGHseg (the default, however, is MAD, not mergeLevels). Merging (using either of "mergeLevels" or "MAD") can also be used with the wavelet-based method of Hsu et al.; please note that the later is an experimental feature implemented by us, and there is no study of its performance.

In summary, for all segmentation methods (except GLAD) merging is available as either "mergeLevels" or "MAD". For DNAcopy, CGHseg, and wavelets as in Hsu et al., you can also choose no merging, though this will rarely be what you want (we offer this option to allow using the original authors' choices in their first descriptions of methods).

When using mergeLevels, we map the results to states of "Alteration", so that we categorize each probe as taking one, and only one, of three possible values, -1 (loss of genomic DNA), 0 (no change in DNA content), +1 (gain of genomic DNA). We have made the assumption, in this mapping, that the "no change" class is the one that has the absolute value closest to zero, and any other classes are either gains or losses. When the data are normalized, the "no change" class should be the most common one. When using MAD this step is implicit in the procedure (any segment with absolute value larger than mad.threshold * MAD is considered aberrant).

Note that "mergeLevels", in addition to being used for calling gains and losses, results in a decrease in the number of distinct smoothed values, since it can merge two or more adjacent smoothed levels. "MAD", in contrast, performs no merging as such, but only calling.

Value

A list of two components:

`outSmoothed` An `ffdf` object with smoothed values. Each column is an array or sample, and each row a probe.

`outState` An `ffdf` object with calls for probes. Each column is an array or sample, and each row a probe. For methods that accept "none" as an argument to '`merging`', the states cannot be interpreted directly as gain or loss; they are simply discrete codes for distinct segments.

Rows and columns of each element can be accessed in the usual way for `ffdf` objects, but accept also most of the usual R operations for data frames.

Author(s)

The code for DNAcopy, HMM, BioHMM, and GLAD are basically wrappers around the original functions by their corresponding authors, with some modiffications for parallelization and usage of ff objects. The original packages are: `DNAcopy`, `aCGH`, `snapCGH`, `cgh`, `GLAD`, respectively. The CGHseg method uses package `tilingArray`.

HaarSeg has been turned into an R package, available from `https://r-forge.r-project.org/projects/haarseg/`. That package uses, at its core, the same R and C code as we do, from Ben-Yaacov and Eldar. We have not used the available R package for historical reasons (we used Eldar and Ben-Yaacov's C and R code in the former ADaCGH package, before a proper R package was available).

For the wavelet-based method we have only wrapped the code that was kindly provided by L. Hsu and D. Grove, and parallelized a few calls. Their original code is included in the sources of the package.

Parallelization and modifications for using ff and additions are by Ramon Diaz-Uriarte <rdiaz02@gmail.com>

References

Carro A, Rico D, Rueda O M, Diaz-Uriarte R, and Pisano DG. (2010). waviCGH: a web application for the analysis and visualization of genomic copy number alterations. *Nucleic Acids Research*, **38 Suppl**:W182–187.

Fridlyand, Jane and Snijders, Antoine M. and Pinkel, Dan and Albertson, Donna G. (2004). Hidden Markov models approach to the analysis of array CGH data. *Journal of Multivariate Analysis*, **90**: 132–153.

Hsu L, Self SG, Grove D, Randolph T, Wang K, Delrow JJ, Loo L, Porter P. (2005) Denoising array-based comparative genomic hybridization data using wavelets. *Biostatistics*, **6**:211-26.

Hupe, P. and Stransky, N. and Thiery, J. P. and Radvanyi, F. and Barillot, E. (2004). Analysis of array CGH data: from signal ratio to gain and loss of DNA regions. *Bioinformatics*, **20**: 3413–3422.

Lingjaerde OC, Baumbusch LO, Liestol K, Glad I, Borresen-Dale AL. (2005). CGH-Explorer: a program for analysis of CGH-data. *Bioinformatics*, **21**: 821–822.

Marioni, J. C. and Thorne, N. P. and Tavare, S. (2006). BioHMM: a heterogeneous hidden Markov model for segmenting array CGH data. *Bioinformatics*, **22**: 1144–1146.

Olshen, A. B. and Venkatraman, E. S. and Lucito, R. and Wigler, M. (2004) Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics*, **4**, 557–572. http://www.mskcc.org/biostat/~olshena/research.

Picard, F. and Robin, S. and Lavielle, M. and Vaisse, C. and Daudin, J. J. (2005). A statistical approach for array CGH data analysis. *BMC Bioinformatics*, **6**, 27. http://dx.doi.org/10.1186/1471-2105-6-27.

Price TS, Regan R, Mott R, Hedman A, Honey B, Daniels RJ, Smith L, Greenfield A, Tiganescu A, Buckle V, Ventress N, Ayyub H, Salhan A, Pedraza-Diaz S, Broxholme J, Ragoussis J, Higgs DR, Flint J, Knight SJ. (2005) SW-ARRAY: a dynamic programming solution for the identification of copy-number changes in genomic DNA using array comparative genome hybridization data. *Nucleic Acids Res.*, **33**:3455-64.

Willenbrock, H. and Fridlyand, J. (2005). A comparison study: applying segmentation to array CGH data for downstream analyses. *Bioinformatics*, **21**, 4084–4091.

Diaz-Uriarte, R. and Rueda, O.M. (2007). ADaCGH: A parallelized web-based application and R package for the analysis of aCGH data, *PLoS ONE*, **2**: e737.

Ben-Yaacov, E. and Eldar, Y.C. (2008). A Fast and Flexible Method for the Segmentation of aCGH Data, *Bioinformatics*, **24**: i139-i145.

See Also

pChromPlot, inputDataToADaCGHData

Examples

```
## Create a temp dir for storing output.
## (Not needed, but cleaner).

dir.create("ADaCGH2_example_tmp_dir")
originalDir <- getwd()
setwd("ADaCGH2_example_tmp_dir")
```

```
## Start a socket cluster. Change the appropriate number of CPUs
## for your hardware

snowfallInit(universeSize = 2, typecluster = "SOCK")

## Get input data in ff format

## To speed up R CMD check, we do not use inputEx1, but a much smaller
## data set. When you try the examples, you might one to use
## inputEx1 instead.

## Not run:

fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx1")

## End(Not run)

fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx2")

tableChromArray <- inputDataToADaCGHData(filename = fname)



### Run all segmentation methods

cbs.out <- pSegmentDNAcopy("cghData.RData",
                           "chromData.RData")
cbs_mad.out <- pSegmentDNAcopy("cghData.RData",
                           "chromData.RData", merging = "MAD")
cbs_none.out <- pSegmentDNAcopy("cghData.RData",
                           "chromData.RData", merging = "none")

names(cbs.out)
cbs.out$outState ## not the best way
open(cbs.out$outSmoothed) ## better
cbs.out$outSmoothed
rle(cbs.out$outSmoothed[, 1])

open(cbs_mad.out$outSmoothed)
rle(cbs_mad.out$outSmoothed[, 1])



hs_ml.out <- pSegmentHaarSeg("cghData.RData",
                           "chromData.RData", merging = "mergeLevels")
hs_mad.out <- pSegmentHaarSeg("cghData.RData",
                           "chromData.RData", merging = "MAD")

open(hs_ml.out[[2]])
open(hs_mad.out[[2]])
summary(hs_ml.out[[2]][,])
summary(hs_mad.out[[2]][,])
```

```
hmm_ml.out <- pSegmentHMM("cghData.RData",
                          "chromData.RData", merging = "mergeLevels")
hmm_mad.out <- pSegmentHMM("cghData.RData",
                          "chromData.RData", merging = "MAD")
hmm_mad_bic.out <- pSegmentHMM("cghData.RData",
                          "chromData.RData", merging = "MAD",
                           aic.or.bic = "BIC")

## we can open the two ffdfs in the list with lapply
lapply(hmm_ml.out, open)
lapply(hmm_mad.out, open)
lapply(hmm_mad_bic.out, open)


rle(hmm_ml.out[[2]][, 3])$lengths
rle(hmm_mad.out[[2]][, 3])$lengths

## MAD and mergeLevels seem to make similar calls in second array
rle(hmm_ml.out[[2]][, 2])$lengths
rle(hmm_mad.out[[2]][, 2])$lengths

## but smoothed values are grouped differently
rle(hmm_ml.out[[1]][, 2])$lengths
rle(hmm_mad.out[[1]][, 2])$lengths

## And BIC leads to differences compared to AIC
open(hmm_mad_bic.out[[2]])
rle(hmm_mad_bic.out[[1]][, 2])$lengths
rle(hmm_mad_bic.out[[2]][, 2])$lengths

### BioHMM is very slow and can crash
## Not run:
biohmm_ml.out <- pSegmentBioHMM("cghData.RData",
                          "chromData.RData",
                          "posData.RData",
                           merging = "mergeLevels")
biohmm_mad.out <- pSegmentBioHMM("cghData.RData",
                          "chromData.RData",
                          "posData.RData",
                           merging = "MAD")
biohmm_mad_bic.out <- pSegmentBioHMM("cghData.RData",
                          "chromData.RData",
                          "posData.RData",
                           merging = "MAD",
                           aic.or.bic = "BIC")

lapply(biohmm_ml.out, open)
lapply(biohmm_mad.out, open)
lapply(biohmm_mad_bic.out, open)

summary(biohmm_ml.out[[2]][,])
summary(biohmm_mad.out[[2]][,])
summary(biohmm_mad_bic.out[[2]][,])

summary(biohmm_ml.out[[1]][,])
summary(biohmm_mad.out[[1]][,])
```

```
summary(biohmm_mad_bic.out[[1]][,])

## End(Not run)


cghseg_ml.out <- pSegmentCGHseg("cghData.RData",
                          "chromData.RData", merging = "mergeLevels")
cghseg_mad.out <- pSegmentCGHseg("cghData.RData",
                          "chromData.RData", merging = "MAD")
cghseg_none.out <- pSegmentCGHseg("cghData.RData",
                          "chromData.RData", merging = "none")

lapply(cghseg_ml.out, open)
lapply(cghseg_mad.out, open)
lapply(cghseg_none.out, open)

summary(cghseg_ml.out[[1]][,])
summary(cghseg_mad.out[[1]][,])
summary(cghseg_none.out[[1]][,])

summary(cghseg_ml.out[[2]][,])
summary(cghseg_mad.out[[2]][,])
summary(cghseg_none.out[[2]][,])


glad.out <- pSegmentGLAD("cghData.RData",
                         "chromData.RData")


waves_ml.out <- pSegmentWavelets("cghData.RData",
                          "chromData.RData", merging = "mergeLevels")
waves_mad.out <- pSegmentWavelets("cghData.RData",
                          "chromData.RData", merging = "MAD")
waves_none.out <- pSegmentWavelets("cghData.RData",
                          "chromData.RData", merging = "none")
lapply(waves_ml.out, open)
lapply(waves_mad.out, open)
lapply(waves_none.out, open)


summary(waves_ml.out[[1]][,])
summary(waves_mad.out[[1]][,])
summary(waves_none.out[[1]][,])

summary(waves_ml.out[[2]][,])
summary(waves_mad.out[[2]][,])
summary(waves_none.out[[2]][,])


############## Clean up actions
####  (These are not needed. They are convenient here, to prevent
####   leaving garbage in your hard drive. In "real life" you will
####   have to decide what to delete and what to store).



### Explicitly stop cluster
```

```
sfStop()


### All objects (RData and ff) are left in this directory
getwd()

### We will clean it up, and do it step-by-step
### BEWARE: DO NOT do this with objects you want to keep!!!

## Remove ff and RData for the data

load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")


## Remove ff and R objects with segmentation results

lapply(cbs.out, delete)
rm(cbs.out)

lapply(cbs_mad.out, delete)
rm(cbs_mad.out)

lapply(cbs_none.out, delete)
rm(cbs_none.out)

lapply(hs_ml.out, delete)
rm(hs_ml.out)

lapply(hs_mad.out, delete)
rm(hs_mad.out)

lapply(hmm_ml.out, delete)
rm(hmm_ml.out)

lapply(hmm_mad.out, delete)
rm(hmm_mad.out)

lapply(hmm_mad_bic.out, delete)
rm(hmm_mad_bic.out)

lapply(cghseg_ml.out, delete)
rm(cghseg_ml.out)

lapply(cghseg_mad.out, delete)
rm(cghseg_mad.out)

lapply(cghseg_none.out, delete)
```

```
rm(cghseg_none.out)

lapply(glad.out, delete)
rm(glad.out)

lapply(waves_mad.out, delete)
rm(waves_mad.out)

lapply(waves_ml.out, delete)
rm(waves_ml.out)

lapply(waves_none.out, delete)
rm(waves_none.out)


## Not run:
## Execute only if you run the BioHMM examples
lapply(biohmm_ml.out, delete)
rm(biohmm_ml.out)
lapply(biohmm_mad.out, delete)
rm(biohmm_mad.out)
lapply(biohmm_mad_bic.out, delete)
rm(biohmm_mad_bic.out)

## End(Not run)

### Try to prevent problems in R CMD check
Sys.sleep(2)
### To prevent CMD check from crashing after cleanEx
detach("package:rlecuyer", unload = TRUE)


### Delete temp dir
setwd(originalDir)
Sys.sleep(2)
unlink("ADaCGH2_example_tmp_dir", recursive = TRUE)
Sys.sleep(2)
```

`snowfallInit` *Initialize a cluster of workstations using snowfall*

Description

With either MPI or sockets, use snowfall to initialize a cluster to have ADaCGH2 run in parallel. Check possible errors during initialization.

Usage

```
snowfallInit(universeSize = NULL, wdir = getwd(),
             minUniverseSize = 2, exit_on_fail = FALSE,
             maxnumcpus = 500, typecluster = "SOCK",
             socketHosts = NULL,
             RNG = "RNGstream")
```

Arguments

universeSize	Desired size of cluster (number of CPUs). Can be set to NULL. See details.
wdir	The common —e.g., NFS mounted resource, a directory in your machine if running on only one computer, etc — directory. We need a common directory for the graphics and ff files so that they are all found in the same location.
minUniverseSize	The minimal LAM/MPI universe for the function to return successfully. If the function determines that the available number of slaves is smaller than minUniverseSize it will fail (if exit_on_fail = TRUE) or give a warning.
exit_on_fail	If TRUE, kills R session if it cannot run succesfully. Setting it to TRUE is something you probably only want to do when running as an unattended service.
maxnumcpus	Passed directly to snowfall. This is the new value of sfSetMaxCPUs set to a very large number to allow us to use large clusters.
typecluster	Either "MPI" or "SOCK". To use MPI, BEFORE calling this function you must configure your MPI environment properly and then load the R package **Rmpi**.
socketHosts	Passed to snowfall-init.
RNG	The type of random number generator. One of "RNGstream" (to use **rlecuyer**) or "SPRNG" (to use the **rsprng** package). If the generator requested is not available, the function tries to use the other one (givin a warning). To use either of these you need to have the appropriate package installed.

Details

This function is designed to be used mainly with MPI, but clusters with sockets might be easier to create in any operating system without additional software. Moreover, installing Rmpi in Windows and Mac is not easy. Thus, by default, the cluster is one of sockets, and Rmpi is listed in "Enhances" (not "Depends" nor "Suggests"). But this function will fail if you try to use an MPI cluster and do not have **Rmpi** loaded. Moreover, even if you successfully install and load **Rmpi**, note that the cluster that gets created by default might not be the want you want (e.g., you might end up with a universe size of one), so it is up to you to configure and, if appropriate, start/boot your MPI environment before loading **Rmpi**.

When using MPI, the recommended usage is to set only minUniverseSize, leaving "universeSize" as NULL. Then, the cluster will use as many nodes as available to MPI (found from "mpi.universe.size()"), or fail if the available number of nodes is less than minUniverseSize. This usage makes sense in many clusters where the actual number of nodes available can vary, but you definitely do not want to run a job unless a minimal number of nodes can be used. (Moreover, mpi.universe.size returning a very small number can be an indication of a configuration file problem).

If "universeSize" is set, this will be the number of nodes of the cluster (unless you are using MPI and mpi.universe.size is smaller, in which case the function will fail).

Value

This function is used to create a cluster.

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

Examples

```
snowfallInit(universeSize = 2, typecluster = "SOCK")

## Not run:
## If you are using MPI, a better approach would be
snowfallInit(minUniverseSize = 4, typecluster = "MPI")
## where minUniverseSize is set to whatever
## you regard as an unacceptable minimum

## End(Not run)


## Better to explicitly stop cluster after you are done
sfStop()

## Not needed. To prevent Windoze from crashing in CMD check.
Sys.sleep(2)

## To prevent CMD check from crashing after cleanEx
detach("package:rlecuyer", unload = TRUE)
Sys.sleep(2)
```

Index