

Package ‘ADaCGH’

May 26, 2009

Version 1.3-13

Date 2009-05-26

Title Analysis of data from aCGH experiments

Author Ramon Diaz-Uriarte <rdiaz02@gamil.com>, Oscar M. Rueda <rueda.om@gamil.com>. Wavelet-based aCGH smoothing code from Li Hsu <lih@fhcrc.org> and Douglas Grove <dgrove@fhcrc.org>. Imagemap code from Barry Rowlingson <B.Rowlingson@lancaster.ac.uk>. HaarSeg code from Erez Ben-Yaacov; downloaded from <<http://www.ee.technion.ac.il/people/YoninaEldar/Info/software/HaarSeg.htm>>

Maintainer Ramon Diaz-Uriarte <rdiaz02@gmail.com>

Depends R (>= 2.2.1), cgh, tilingArray, aCGH, cghMCR, papply, GDD, waveslim, cluster, snapCGH, Hmisc

Suggests Rmpi, GLAD, DNACopy, rsprng

Description Analysis and plotting of array CGH data. Allows usage of Circular Binary Segmentation, wavelet-based smoothing (both as in Liu et al., and HaarSeg as in Ben-Yaacov and Eldar), ACE method (CGH Explorer), HMM, BioHMM, GLAD, CGHseg, and Price’s modification of Smith & Waterman’s algorithm. Most computations are parallelized. Figures are imagemaps with links to IDClight (<http://idclight.bioinfo.cnio.es>).

LazyLoad Yes

License GPL (>= 3)

URL <http://adacgh2.bioinfo.cnio.es>, <http://launchpad.net/adacgh>

Repository CRAN

Date/Publication 2009-05-26 17:55:56

R topics documented:

| | |
|----------------------------|-----------|
| cghE1 | 2 |
| cghMCRre | 3 |
| doMCR | 3 |
| mpiInit | 5 |
| pSegment | 6 |
| segmentPlot | 14 |
| SegmentPlotWrite | 15 |
| summary.ACE | 18 |
| tempdir2 | 20 |
| writeResults | 20 |
| Index | 23 |

| | |
|-------|-----------------------------------|
| cghE1 | <i>A fictitious aCGH data set</i> |
|-------|-----------------------------------|

Description

A fictitious aCGH data set that contains UniGene identifiers to show how linking to IDClight <http://idclight.bioinfo.cnio.es> works.

Usage

```
cghE1
```

Format

A data frame with 4323 rows and 9 columns, five of which correspond to the aCGH data for 5 samples.

Source

Simulated data

References

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.

`cghMCRe`*A fictitious aCGH data set*

Description

A fictitious aCGH data set that can be used to show some Minimal common regions.

Usage`cghMCRe`**Format**

A data frame with 4706 rows and 7 columns, three of which correspond to the aCGH data for 3 samples.

Source

Simulated data

References

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.

`doMCR`*Find minimal common regions*

Description

A wrapper for the functionality implemented in the `cghMCR` package suitable for objects such as those returned by our functions. Output is returned as HTML and text files.

Usage

```
doMCR(x, chrom.numeric, data, MCR.gapAllowed = 500,  
      MCR.alteredLow = 0.03, MCR.alteredHigh = 0.97,  
      MCR.recurrence = 75, fsink = "results.txt",  
      hsink = "mcr.results.html", Pos, ...)
```

Arguments

| | |
|------------------------------|---|
| <code>x</code> | The "segm" component of an object returned by our <code>pSegment*</code> functions, such as <code>pSegmentDNACopy</code> . For instance <code>whatevername\$semg</code> , where <code>whatevername</code> is the result of a previous invocation of <code>pSegment</code> . |
| <code>chrom.numeric</code> | See <code>pSegmentDNACopy</code> |
| <code>data</code> | The original aCGH data; genes in rows, subjects or arrays in columns. Data should be ordered by chromosome and position within chromosome. |
| <code>MCR.gapAllowed</code> | See <code>MCR</code> |
| <code>MCR.alteredLow</code> | See <code>MCR</code> |
| <code>MCR.alteredHigh</code> | See <code>MCR</code> |
| <code>MCR.recurrence</code> | See <code>MCR</code> |
| <code>fsink</code> | Name of file where text results are written to. |
| <code>hsink</code> | Name of file where HTML results are written to. |
| <code>Pos</code> | The location (e.g., position in kbases) of each probe in the chromosome. A vector of the same length as the number of rows in <code>data</code> |
| <code>...</code> | Additional arguments (not used now). |

Value

Used only for its side effects of writing out files.

Author(s)

Ramon Diaz-Uriarte (rdiaz02@gmail.com)

References

Aguirre, Brennan, Bailey, Sinha, Feng, Leo, Zhang, Zhang, Gans, Bardeesy, Cauwels, Cordon-Cardo, Redston, Depinho, Chin. (2004). High-resolution characterization of the pancreatic adenocarcinoma genome. *PNAS*, 101: 9067–9072.

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh2.bioinfo.cnio.es>.

See Also

`MCR`

| | |
|---------|--|
| mpiInit | <i>Initialized a cluster of workstations</i> |
|---------|--|

Description

Use Rmpi and papply, and set a cluster of workstations ready to run ADaCGH.

Usage

```
mpiInit(wdir = getwd(), minUniverseSize = 15, universeSize = NULL,  
        exit_on_fail = FALSE)
```

Arguments

`wdir` The common —e.g., NFS mounted— directory. We need a common directory for the graphics, 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.

`universeSize` Alternatively, you can specify the actual size of the LAM/MPI universe. If this many slaves cannot be started, the function will fail.

`exit_on_fail` If TRUE, kills R session if it cannot run successfully.

Details

This function calls Rmpi and related functions to start a cluster of workstations, initializes the random number generator, and calls the required libraries on the slave nodes.

Value

This function is used to create a cluster.

Note

We have a somewhat similar function in `basicClusterInit.Rd`, but that is more sophisticated.

Author(s)

Ramon Diaz-Uriarte (rdiaz02@gmail.com)

References

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.

Examples

```
## Not run:
## all slaves need a common dir to read and write.
dir.to.create <- tempdir2()
setwd(dir.to.create)
mpiInit()
## End(Not run)
```

pSegment

Parallelized/"unified" versions of several aCGH segmentation 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(x, chrom.numeric, parall = "arr", ...)
pSegmentACE(x, chrom.numeric, parall = "auto", ...)
pSegmentHMM(x, chrom.numeric, parall = "auto", ...)
pSegmentGLAD(x, chrom.numeric, ...)
pSegmentBioHMM(x, chrom.numeric, Pos, parall = "auto", ...)
pSegmentCGHseg(x, chrom.numeric, CGHseg.thres = -0.05, ...)
pSegmentPSW(x, chrom.numeric, common.data,
             sign = -1,
             nIter = 1000, prec = 100, p.crit = 0.10,
             name = NULL, ...)
pSegmentWavelets(x, chrom.numeric, mergeSegs = TRUE,
                 minDiff = 0.25,
                 minMergeDiff = 0.05,
                 thrLvl = 3, initClusterLevels = 10, ...)
pSegmentHaarSeg(x, chrom.numeric, HaarSeg.m = 3,
                W = vector(),
                rawI = vector(),
                breaksFdrQ = 0.001,
                haarStartLevel = 1,
```

```
haarEndLevel = 5, ...)
```

Arguments

| | |
|--------------------------------|--|
| <code>x</code> | The aCGH data; genes in rows, subjects or arrays in columns. Data should be ordered by chromosome and position within chromosome. |
| <code>chrom.numeric</code> | The (numeric) vector with the chromosome indicator |
| <code>Pos</code> | The location (e.g., position in kbases) of each probe in the chromosome. A vector of the same length as the number of rows in <code>x</code> |
| <code>parall</code> | Where available whether to parallelize over arrays (all except ACE) or chromosomes (ACE) or array by chromosome (ACE, DNACopy, HMM, BioHMM). |
| <code>CGHseg.thres</code> | 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. |
| <code>mergeSegs</code> | Merge (genome-wide) the resulting segments using <code>mergeLevels?</code> . See details. |
| <code>common.data</code> | A data frame with columns that contain the common information about the clones/genes. For now, this is limited to five columns of data. We do nothing with this, except it is part of the output. It is common practice that this data frame contains columns ID, Chromosome (need not be numeric, can be a factor), Start (the initial position of a clone or gene), End (the final position of a clone or gene) and MidPoint (often, the mid point or whatever you regard as appropriate representative of the "center" of the clone). But you can use something else. |
| <code>sign</code> | See <code>sw.threshold</code> |
| <code>nIter</code> | See <code>sw.perm.test</code> |
| <code>prec</code> | See <code>sw.rob</code> |
| <code>p.crit</code> | The largest p-value for which we want a region to be shown, in red, in the plot. |
| <code>name</code> | Sample name or ID |
| <code>minDiff</code> | 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. |
| <code>minMergeDiff</code> | Used only when doing merging in the wavelet method. The final call as to which segments go together is done by a <code>mergeLevels</code> approach, but an initial collapsing of very close values is performed (otherwise, we could end up passing to <code>mergeLevels</code> as many initial levels as there are points). |
| <code>thrLvl</code> | The level used for the wavelet thresholding in Hsu et al. |
| <code>initClusterLevels</code> | The initial number of clusters to form. |
| <code>HaarSeg.m</code> | For HaarSeg the value such that all segments where $\text{abs}(\text{smoothed value}) > m * \text{MAD}$ will be declared aberrant —see p. 1141 of Ben-Yaacov and Eldar |

| | |
|-----------------------------|--|
| <code>W</code> | For HaarSeg: Weight matrix, corresponding to quality of measurement. Insert $1/(\sigma^{**2})$ as weights if your platform output sigma as the quality of measurement. <code>W</code> must have the same size as <code>I</code> . |
| <code>rawI</code> | For HaarSeg. Minimum of the raw red and raw green measurement, before the log. <code>rawI</code> is used for the non-stationary variance compensation. <code>rawI</code> must have the same size as <code>I</code> . |
| <code>breaksFdrQ</code> | For HaarSeg. The FDR <code>q</code> parameter. Common used values are 0.05, 0.01, 0.001. Default value is 0.001. |
| <code>haarStartLevel</code> | 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 value is 1, corresponding to segments of 2 probes. |
| <code>haarEndLevel</code> | For HaarSeg. The detail subband until which we use to detect peaks. The higher this value is, the more sensitive we are to large trends in the data. This value DOES NOT indicate the largest possible segment that can be detected. The default value is 5, corresponding to step of 32 probes in each direction. |
| <code>...</code> | Additional arguments; not used. |

Details

In most cases, these are wrappers to the original code, with modifications for parallelization. For CGHseg, BioHMM and HMM the results are merged, by the `mergeLevels` algorithm, as recommended in *Willenbrock and Fridlyand, 2005*. Merging is also done in GLAD (with GLAD's own merging algorithm). Merging is the default for DNACopy (but this can be set by the user). Merging 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 evidence of its performance.

Using the output of `mergeLevels`, we then 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.

For HaarSeg, we use the approach in page 1141 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 $m * MAD$ is considered aberrant.

For PSW, the method implemented here is a small departure from the one described originally in Price et al.: here we use all the data for a subject/array for the thresholding, but subsequent analyses (finding islands and robustness analysis and permutation tests) are carried out chromosome by chromosome. This should allow to detect cases where a complete chromosome is gained or lost, as well as cases where smaller regions are gained or lost. Note that this approach is probably not advisable for the X chromosome, at least for males. You probably want to analyze the X separately, since you only expect one copy of the genomic DNA in males. Finally, the function for calculating the threshold (the default used by T. Price originally, which we also use —median + 0.2 * MAD—) was validated by Price and collaborators on a 75-probe dataset that spanned the telomeric 2MB of chromosome 16p, using subjects with well-characterized deletions in the region and healthy controls. But it has not been validated with whole-genome chips or other DNA sources. (If you want to play around with the code, the sources are available from <http://www.well.ox.ac.uk/~tprice/cgh/>

and as the R package `cgh`). (I thank Tom Price for his discussion on these issues; some sentences here are copied almost verbatim from his emails.)

Beware that, with DNACopy, parallelization over arrays by chromosome can lead to slightly different results than parallelization over arrays (and, thus, to the original function). This is because of the way the smoothing is carried out, which affects the `timmed.SD` used.

Value

All the functions return an object of class "adacgh.generic.out". In addition, depending on the function, the returned object can also be of class "CGH.wave.merged", "CGH.wave", "mergedHMM", "adacghGLAD", "mergedBioHMM", "CGHseg", "CGH.PSW".

For methods DNACopy, HMM, BioHMM, GLAD, CGHseg, and wavelet smoothing the output object is a list with two components:

```

segm          The output from the segmentation
chrom.numeric The same value as chrom.numeric in the function call.
```

The `segm` component is a list, with as many components as subjects or arrays. Each of these lists is a matrix with three columns. The first are the observed data, the second the predicted/smoothed/merged data, and the third the state/alteration. When the method performs a merging step, we call the third column "Alteration", and it can only take three values, -1 (loss of genomic DNA), 0, +1 (gain of genomic DNA).

Method PSW's output contains two components. The second, `plotData` with data that is used for plotting (and the details are not documented, since you are not supposed to use them, as this is subject to change). The first component, `Data`, is a list with as many components as arrays or subjects. Each of this is an array with columns with original data, the sign, the value from the robustness test, and the p-value from the permutation test. See details in [sw.threshold](#), [sw.perm.test](#), [sw.rob](#).

Method ACE's output is a list. None of the elements of this list are to be used directly by a user, and are later used by the function [summary.ACE](#).

Author(s)

The code for DNACopy, HMM, BioHMM, PSW, and GLAD are basically wrappers around the original functions by their corresponding authors, with some modifications for parallelization (packages DNACopy, aCGH, snapCGH, cgh, GLAD, respectively). The CGHseg method uses package `tilingArray`.

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.

The code for ACE is based on the original Java code for the CGHExplorer package. It was turned into C and R code by Oscar M. Rueda (omrueda@cniio.es).

Parallelization and other modifications and additions are by Ramon Diaz-Uriarte (rdiaz02@gmail.com)

References

- 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. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.
- 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

[segmentPlot](#)

Examples

```
## Not run:
## all slaves need a common dir to read and write.
dir.to.create <- tempdir2()
setwd(dir.to.create)
dir.to.create
mpiInit() ## this is used in SegmentPlotWrite, which is run first.
## End(Not run)

## library(Rmpi)
```

```

## try(mpi.spawn.Rslaves(nslaves = 1), silent = TRUE)

data(cghE1)
tmpchr <- sub("chr", "", cghE1$Chromosome)
chrom.numeric <- as.numeric(as.character(tmpchr))
chrom.numeric[tmpchr == "X"] <- 23
chrom.numeric[tmpchr == "Y"] <- 24
rm(tmpchr)
### we need the data ordered
reorder <- order(chrom.numeric,
                 cghE1$UG.Start,
                 cghE1$UG.End,
                 cghE1$Name)
cghE1 <- cghE1[reorder, ]
chrom.numeric <- chrom.numeric[reorder]

## run all methods
cbs.out <- pSegmentDNACopy(cghE1[, 5:7], chrom.numeric)
cbs.ns.out <- pSegmentDNACopy(cghE1[, 5:7], chrom.numeric, smooth = FALSE)
glad.out <- pSegmentGLAD(cghE1[, 5:7], chrom.numeric)
cghseg.out <- pSegmentCGHseg(cghE1[, 5:7], chrom.numeric, CGHseg.thres = -0.05)
ace.out <- pSegmentACE(cghE1[, 5:7], chrom.numeric)

hmm.out <- pSegmentHMM(cghE1[, 5:7], chrom.numeric)
wave.out <- pSegmentWavelets(cghE1[, 5:7], chrom.numeric)
wave.nm.out <- pSegmentWavelets(cghE1[, 5:7], chrom.numeric, mergeSegs = FALSE)

haar.out <- pSegmentHaarSeg(cghE1[, 5:7], chrom.numeric)

## Next one we try, as it could fail when it should not.
try(psw.pos.out <- pSegmentPSW(cghE1[, 5:7], chrom.numeric, sign = 1))
try(psw.neg.out <- pSegmentPSW(cghE1[, 5:7], chrom.numeric, sign = -1))

## BioHMM is the only one that uses distances
## it is the slowest, so do only two
## We use try, as snapCGH sometimes fails and does not load its compiled C code
try(biohmm.out <- pSegmentBioHMM(cghE1[, 5:6], chrom.numeric, cghE1$UG.Start))

## With ACE we need to choose fdr
ace.out.sum <- summary(ace.out)

#### Writing
common <- cghE1[, 1:4]
writeResults(hmm.out, cghE1[, 5:7], common)
writeResults(glad.out, cghE1[, 5:7], common)
writeResults(cghseg.out, cghE1[, 5:7], common)
writeResults(ace.out.sum, cghE1[, 5:7], common)
writeResults(wave.out, cghE1[, 5:7], common)
writeResults(wave.nm.out, cghE1[, 5:7], common)
writeResults(cbs.out, cghE1[, 5:7], common)
writeResults(haar.out, cghE1[, 5:7], common)

try(writeResults(psw.pos.out, common))

```

```
try(writeResults(psw.neg.out, common))
try(writeResults(biohmm.out, cghE1[, 5:6], common))

### plot all

## Not run:

## This will not work unless you have a working Python installation

segmentPlot(hmm.out,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")

segmentPlot(glad.out,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")

segmentPlot(cghseg.out,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")

segmentPlot(wave.out,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")

segmentPlot(wave.out,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")

segmentPlot(haar.out,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")
```

```
segmentPlot(cbs.out,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")

try(segmentPlot(psw.pos.out,
              geneNames = cghE1[, 1],
              chrom.numeric = chrom.numeric,
              cghdata = cghE1[, 5:7],
              idtype = "ug",
              organism = "Hs"))

try(segmentPlot(psw.neg.out,
              geneNames = cghE1[, 1],
              chrom.numeric = chrom.numeric,
              cghdata = cghE1[, 5:7],
              idtype = "ug",
              organism = "Hs"))

try(segmentPlot(biohmm.out,
              geneNames = cghE1[, 1],
              chrom.numeric = chrom.numeric,
              cghdata = cghE1[, 5:6],
              idtype = "ug",
              organism = "Hs"))

segmentPlot(ace.out.sum,
            geneNames = cghE1[, 1],
            chrom.numeric = chrom.numeric,
            cghdata = cghE1[, 5:7],
            idtype = "ug",
            organism = "Hs")

## End(Not run)

## To verify we are doing the right thing with Picard et al.'s method,
## compare with their figure 1. Need to get the txt files with the
## original data.
## We got the data from the GLAD BioC package, and then exported the txt.
## For convenience, it is in the "/inst/txt-data" directory of our package.

## This verifies the output (look at ADaCGH.R for further details and other
## comparisons):

## coriel.data <- read.table("gm03563.txt", header = TRUE)
## cd3 <- coriel.data[coriel.data$Chromosome == 3, 3]

## out.our <- CGHsegWrapper(cd3, rep(1, length(cd3)), s = -0.5) ## k = 2
## out.our

## so our implementation seems correct. Look at where the
```

```
## breakpoint is located, etc, and it is like figure 1 of
## Picard's paper.
```

segmentPlot

Segment plots for aCGH

Description

A (more or less) uniform interface and uniform output to segment plots for the available aCGH methods in this package. By the default, it produces html files (and associated png) with an image map with links to additional information. You need Python for all this to work.

Usage

```
segmentPlot(x, geneNames, chrom.numeric = NULL, cghdata = NULL,
            arraynames = NULL, idtype = "ug", organism = "Hs",
            yminmax = NULL, numarrays = NULL,
            colors = c("orange", "red", "green", "blue", "black"),
            html_js = TRUE,
            superimp = TRUE,
            imgheight = 500,
            ...)
```

Arguments

| | |
|---------------|---|
| x | The fitted object of the appropriate class. |
| geneNames | A vector of gene names. |
| chrom.numeric | A (numeric) vector of chromosome indicators. |
| cghdata | The aCGH data. |
| arraynames | A vector of array names. |
| idtype | The type of id of the gene name; one of ('None', 'cnio', 'affy', 'clone', 'acc', 'ensembl', 'entrez', 'ug') corresponding, respectively to None, CNIO ID, Affymetrix, Clone ID, Accession Number, Ensembl ID, Entrez ID, UniGene. |
| organism | The organism; one of ('None', 'Hs', 'Mm', 'Rn') corresponding, respectively, to None, Homo sapiens, Mus musculus (mouse) and Rattus norvegicus (rat). |
| yminmax | A vector of the form (min, max) for the minimum and maximum used in the figures. |
| numarrays | The number of arrays to plot. |
| colors | Colors used for plotting: a five element vector denoting colors for non-altered points, gained, lost, the smoothed line and the horizontal line at 0. |

| | |
|-----------|--|
| html_js | Produce the html figures including lots of Javascript? Default is TRUE. |
| superimp | Produce the figures that superimpose all arrays into a single plot? Default is TRUE. |
| imgheight | The image height when generating png. Default is 500. |
| ... | Other arguments, passed to the underlying plotting functions. |

Value

Used only for its side effects of producing plots. Beware that you will get from a few to a lot of HTML and png files written to your local file system.

Note

Most of the arguments are not really needed in most cases. See examples below.

You NEED a working Python for this function to work and produce the imagemaps.

Author(s)

Ramon Diaz-Uriarte (rdiaz02@gmail.com)

References

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.

See Also

[pSegmentWavelets](#), [pSegmentACE](#), [pSegmentDNACopy](#), [pSegmentPSW](#),

Examples

```
## See examples under pSegmentDNACopy
```

SegmentPlotWrite *Segment aCGH data, plot, and write out results*

Description

Wrapper to do in a single call the segmentation, plotting, and writing out of results from segmentation.

Usage

```
SegmentPlotWrite(data, chrom, mergeSegs, Pos, idtype,
                 organism, method, geneNames, comdata,
                 colors = c("orange", "red", "green", "blue", "black"),
                 html_js = TRUE,
                 superimp = TRUE,
                 imgheight = 500,
                 ...)
```

Arguments

| | |
|------------------------|--|
| <code>data</code> | The aCGH data; genes in rows, subjects or arrays in columns. Data should be ordered by chromosome and position within chromosome. |
| <code>chrom</code> | The (numeric) vector with the chromosome indicator |
| <code>mergeSegs</code> | Merge (genome-wide) the resulting segments using <code>mergeLevels?</code> |
| <code>Pos</code> | The location (e.g., position in kbases) of each probe in the chromosome. A vector of the same length as the number of rows in <code>data</code> |
| <code>idtype</code> | The type of id of the gene name; one of ('None', 'cnio', 'affy', 'clone', 'acc', 'ensembl', 'entrez', 'ug') corresponding, respectively to None, CNIO ID, Affymetrix, Clone ID, Accession Number, Ensembl ID, Entrez ID, UniGene. |
| <code>organism</code> | The organism; one of ('None', 'Hs', 'Mm', 'Rn') corresponding, respectively, to None, Homo sapiens, Mus musculus (mouse) and Rattus norvegicus (rat). |
| <code>method</code> | The segmentation method. One of "ACE", "HMM", "BioHMM", "GLAD", "DNAcopy", "CGHseg", "PSW", "Wavelets", "HaarSeg". |
| <code>geneNames</code> | A vector with the names of the genes or probes, as you want to use them to label figures and output. |
| <code>comdata</code> | A data frame with columns that contain the common information about the clones/genes. We do nothing with this, except it is part of the output. It is common practice that this data frame contains columns ID, Chromosome (need not be numeric, can be a factor), Start (the initial position of a clone or gene), End (the final position of a clone or gene) and MidPoint (often, the mid point or whatever you regard as appropriate representative of the "center" of the clone). But you can use something else. |
| <code>colors</code> | Colors used for plotting: a five element vector denoting colors for non-altered points, gained, lost, the smoothed line and the horizontal line at 0. |
| <code>html_js</code> | Produce the html figures including lots of Javascript? Default is TRUE. |
| <code>superimp</code> | Produce the figures that superimpose all arrays into a single plot? Default is TRUE. |
| <code>imgheight</code> | The image height when generating png. Default is 500. |
| <code>...</code> | Additional arguments passed to the underlying <code>pSegment*</code> and <code>segmentPlot</code> functions. |

Details

Just a convenience wrapper.

Value

This function is used for its side effects. Produce plots and write out results to a file. See [writeResults](#) and [segmentPlot](#).

Author(s)

Ramon Diaz-Uriarte (rdiaz02@gmail.com)

References

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.biocinfo.cnio.es>.

See Also

[writeResults](#), [segmentPlot](#), [pSegment](#)

Examples

```
## Not run:
## This will not work unless you have a working Python installation

##setwd("/tmp/o3") ## all slaves need a common dir to read and write.
## all slaves need a common dir to read and write.
dir.to.create <- tempdir2()
setwd(dir.to.create)
dir.to.create
mpiInit()

data(cghE1)
tmpchr <- sub("chr", "", cghE1$Chromosome)
chrom.numeric <- as.numeric(as.character(tmpchr))
chrom.numeric[tmpchr == "X"] <- 23
chrom.numeric[tmpchr == "Y"] <- 24
rm(tmpchr)
reorder <- order(chrom.numeric,
                 cghE1$UG.Start,
                 cghE1$UG.End,
                 cghE1$Name)
cghE1 <- cghE1[reorder, ]
chrom.numeric <- chrom.numeric[reorder]

## to make it run fuster, use just the first two chromosomes
si <- cumsum(table(chrom.numeric))[2]
cghE1 <- cghE1[1:si, ]
chrom.numeric <- chrom.numeric[1:si]

## one example
SegmentPlotWrite(cghE1[, 5:7], chrom.numeric,
                 merge = FALSE, Pos = cghE1$UG.Start,
```

```

        idtype = "ug", organism = "Hs",
        method = "Wavelets",
        geneNames = cghE1[, 1],
        commondata = cghE1[, 1:4])

## all other methods except PSW and ACE and BioHMM (this fails because
## of a problem in the library)
for(mm in c("DNACopy", "GLAD", "HMM", "CGHseg", "HaarSeg")) {

    cat("\n\n mm is ", mm, "\n\n")
    SegmentPlotWrite(cghE1[, 5:6], chrom.numeric,
                     merge = TRUE, Pos = cghE1$UG.Start,
                     idtype = "ug", organism = "Hs",
                     method = mm,
                     geneNames = cghE1[, 1],
                     commondata = cghE1[, 1:4])
}

## Now try BioHMM

try(SegmentPlotWrite(cghE1[, 5:6], chrom.numeric,
                    merge = TRUE, Pos = cghE1$UG.Start,
                    idtype = "ug", organism = "Hs",
                    method = "BioHMM",
                    geneNames = cghE1[, 1],
                    commondata = cghE1[, 1:4]))

## End(Not run)

```

summary.ACE

Gains/loss and FDR from an ACE object

Description

Shows the number of genes detected as gains/loss at different levels of FDR, allows setting the desired FDR, and returns the assigned status of each genes for the selected FDR.

Usage

```

## S3 method for class 'ACE':
summary(object, fdr = NULL, html = TRUE,
        outhtml = "ace.fdrtable.html", ...)

```

Arguments

| | |
|--------|--|
| object | An object of class ACE or ACE.array, as returned from pSegmentACE |
| fdr | The desired FDR. The actual value used is the closest one the desired. If NULL, the FDR selected is the closest one to 0.15. |

| | |
|---------|---|
| html | If TRUE, produce also an HTML table |
| outhtml | The name of the file with the HTML output |
| ... | Not used. |

Value

Calling this function always returns (prints) a table with the number of Genes called gains/losses at each of the available FDRs and the index (row) corresponding to the selected FDR (or the closest FDR to 0.15). Additionally, a list with as many components as samples. Each list component is a data frame with genes/clones as rows and three columns: the chromosome identifier, the original log-ratio, and the status (0 for no change, -1 for loss, +1 for gain).

Author(s)

Oscar M. Rueda (omrueda@cnio.es)

References

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.

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.

See Also

[pSegmentACE](#), [segmentPlot](#)

Examples

```
## Not run:
data(cghMCRE)
chrom.numeric <- as.numeric(as.character(cghMCRE$Chromosome))
chrom.numeric[cghMCRE$Chromosome == "X"] <- 23
chrom.numeric[cghMCRE$Chromosome == "Y"] <- 24

## Recall: we must reorder the data by chromosome and
## by position within chromosome
reorder <- order(chrom.numeric,
                 cghMCRE$Start,
                 cghMCRE$End,
                 cghMCRE$Name)
cghMCRE <- cghMCRE[reorder, ]
chrom.numeric <- chrom.numeric[reorder]

ace.out <- pSegmentACE(cghMCRE[, 5:7], chrom.numeric)
summary(ace.out, fdr = 0.01)
## End(Not run)
```

`tempdir2`*Create a directory for output*

Description

Creates a (unique) temporary directory for storing all output. A simple utility function.

Usage

```
tempdir2()
```

Details

Just a convenience wrapper.

Value

This function is used for its side effects. Produce a temporary directory. It returns the name of the temporary directory, so you can then move all further work to that directory.

Author(s)

Ramon Diaz-Uriarte (rdiaz02@gmail.com)

References

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.

Examples

```
## See examples under pSegmentDNACopy
```

`writeResults`*Write results to an external file*

Description

Write segmentation results to an external file, including original information (name, chromosome, position, log-ratio) and assigned state and/or smoothed mean (depending on what is available from the method). (In addition, a set of files with links to PaLS <http://pals.bioinfo.cnio.es> will be generated when there is state information.)

Usage

```

writeResults(obj, ...)
## S3 method for class 'CGH.PSW':
writeResults(obj, comdata, file = "PSW.output.txt", ...)
## S3 method for class 'summaryACE':
writeResults(obj, acghdata, comdata, file = NULL, ...)
## S3 method for class 'CGH.wave':
writeResults(obj, acghdata, comdata,
             file = "wavelet.output.txt", ...)
## S3 method for class 'DNACopy':
writeResults(obj, acghdata, comdata,
             file = "CBS.output.txt", ...)
## S3 method for class 'CGHseg':
writeResults(obj, acghdata, comdata,
             file = "CGHseg.output.txt", ...)
## S3 method for class 'mergedHMM':
writeResults(obj, acghdata, comdata,
             file = "HMM.output.txt", ...)
## S3 method for class 'adacghGLAD':
writeResults(obj, acghdata, comdata,
             file = "GLAD.output.txt", ...)
## S3 method for class 'mergedBioHMM':
writeResults(obj, acghdata, comdata,
             file = "BioHMM.output.txt", ...)

```

Arguments

| | |
|----------|---|
| obj | The output object from previous analysis; in all cases, the output from pSegment except for ACE, where it is the output from summary. |
| file | The name of the output file. |
| acghdata | The aCGH data set; see pSegmentWavelets . |
| comdata | The common data; see <code>common.data</code> in pSegmentPSW . |
| ... | Other arguments passed to the underlying functions. |

Value

A file is written in your system with columns corresponding to the original data (including ID, Chromosome, etc) and the results of the segmentations.

Author(s)

Ramon Diaz-Uriarte (rdiaz02@gmail.com)

References

Diaz-Uriarte, R. and Rueda, O.M. (2006). ADaCGH: an R package and web-based application for the analysis of aCGH data. Tech. report. <http://www.ligarto.org/rdiaz/Papers/adacgh.pdf>, <http://adacgh.bioinfo.cnio.es>.

See Also

[pSegmentACE](#), [pSegmentWavelets](#), [pSegmentPSW](#), [pSegmentDNACopy](#), [summary.ACE](#),

Examples

```
## See examples under pSegmentDNACopy
```

Index

*Topic **IO**

doMCR, 3
segmentPlot, 13
SegmentPlotWrite, 15
tempdir2, 19
writeResults, 20

*Topic **datasets**

cghE1, 2
cghMCR, 2

*Topic **file**

writeResults, 20

*Topic **hplot**

segmentPlot, 13
SegmentPlotWrite, 15

*Topic **misc**

summary.ACE, 17
writeResults, 20

*Topic **nonparametric**

pSegment, 5
summary.ACE, 17

*Topic **programming**

mpiInit, 4

basicClusterInit.Rd, 5

cghE1, 2
cghMCR, 2

doMCR, 3

MCR, 3, 4
mergeLevels, 6, 15
mpiInit, 4

pSegment, 5, 16
pSegmentACE, 14, 18, 21
pSegmentACE (*pSegment*), 5
pSegmentBioHMM (*pSegment*), 5
pSegmentCGHseg (*pSegment*), 5
pSegmentDNACopy, 3, 14, 21
pSegmentDNACopy (*pSegment*), 5

pSegmentGLAD (*pSegment*), 5
pSegmentHaarSeg (*pSegment*), 5
pSegmentHMM (*pSegment*), 5
pSegmentPSW, 14, 20, 21
pSegmentPSW (*pSegment*), 5
pSegmentWavelets, 14, 20, 21
pSegmentWavelets (*pSegment*), 5

segmentPlot, 10, 13, 16, 18
SegmentPlotWrite, 15
summary.ACE, 8, 17, 21
sw.perm.test, 6, 8
sw.rob, 7, 8
sw.threshold, 6, 8

tempdir2, 19

writeResults, 16, 20
writeResults.CGH.ACE.summary
(*writeResults*), 20
writeResults.CGH.PSW
(*writeResults*), 20
writeResults.CGH.wave
(*writeResults*), 20
writeResults.CGHseg
(*writeResults*), 20
writeResults.DNACopy
(*writeResults*), 20
writeResults.mergedBioHMM
(*writeResults*), 20
writeResults.mergedHMM
(*writeResults*), 20
writeResults.summaryACE
(*writeResults*), 20