

Package ‘RJaCGH’

April 17, 2009

Version 2.0.0

Date 2009-03-04

Title Reversible Jump MCMC for the analysis of CGH arrays.

Author Oscar Rueda <rueda.om@gmail.com> and Ramon Diaz-Uriarte <rdiaz02@gmail.com>. zlib from Jean-loup Gailly and Mark Adler; see README. Function “getHostname.System” from package R.utils by Henrik Bengtsson.

Maintainer Oscar Rueda <rueda.om@gmail.com>

Depends R (>= 2.7.0)

Description Bayesian analysis of CGH microarrays fitting Hidden Markov Chain models. The selection of the number of states is made via their posterior probability computed by Reversible Jump Markov Chain Monte Carlo Methods. Also returns probabilistic common regions for gains/losses.

LazyLoad Yes

License GPL-3

URL <http://www.r-project.org>

biocViews Microarray, DNACopyNumber

Repository CRAN

Date/Publication 2009-03-04 17:53:07

R topics documented:

genomePlot	2
modelAveraging	3
normal.HMM.likelihood.NH.C	5
plot.pREC_S	6
plot.Q.NH	8
plot.RJaCGH	9

pREC_A	11
pREC_S	13
print.pREC_A	16
print.pREC_S	17
print.summary.RJaCGH	18
Q.NH	19
relabelStates	21
RJaCGH	22
simulateRJaCGH	27
smoothMeans	29
snijders	30
states	31
summary.RJaCGH	32
trace.plot	33

Index	35
--------------	-----------

genomePlot	<i>Plot of the genome with probabilities of alteration.</i>
------------	---

Description

Plot of the genome showing, with a color key, the marginal probability of every gene of alteration.

Usage

```
genomePlot(obj, array=NULL, weights=NULL,
           col = NULL, breakpoints = NULL, legend.pos=NULL, ...)
```

Arguments

obj	An object of class RJaCGH.Chrom, RJaCGH.Genome or RJaCGH.array.
array	Name of the array to be plotted. If NULL, the weighed average of all is computed.
weights	vector of weights for each array. Must have the length of the number of arrays. If NULL, the weights are uniform.
col	A vector of length k for the color of every range of probabilities of alteration, starting from loss to gain.
breakpoints	A vector of length $k-1$ for the breakpoints of the color key. The corresponding to losses must be negative. See example for details.
legend.pos	Position of the legend. Must be a vector with two elements; the position of the x and y coordinates. If NULL, the legend is placed at the right.
...	Additional parameters passed to plot.

Details

If `col` and `breakpoints` are NULL, a default color key is drawn.

Value

A plot is drawn.

Note

The positions of the genes should be relative to the chromosome for the plot to make sense.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

Examples

```
data(snijders)
y <- gm13330$LogRatio[!is.na(gm13330$LogRatio)]
Pos <- gm13330$PosBase[!is.na(gm13330$LogRatio)]
Chrom <- gm13330$Chromosome[!is.na(gm13330$LogRatio)]

## Sort positions

for (i in unique(Chrom)) {
  if(any(diff(Pos[Chrom==i]) < 0)) {
    id <- order(Pos[Chrom==i])
    y[Chrom==i] <- y[Chrom==i][id]
    Pos[Chrom==i] <- Pos[Chrom==i][id]
  }
}

jpb <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
           sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)
fit.genome <- RJACGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
                    burnin=1000, TOT=1000, jump.parameters=jpb, k.max = 4)
genomePlot(fit.genome)
genomePlot(fit.genome, col=c(3, 1, 2), breakpoints=c(-0.5, 0.5))
```

modelAveraging

Method for model averaging for RJACGH objects.

Description

Bayesian model averaging for the estimation of hidden state sequence.

Usage

```
modelAveraging(obj, array=NULL, Chrom=NULL)
## S3 method for class 'RJaCGH':
modelAveraging(obj, array=NULL, Chrom=NULL)
```

Arguments

obj	An object of corresponding class
array	Array to be used. If NULL, all of them are used.
Chrom	Vector of chromosomes to be used. If NULL, all of them are used.

Details

With the posterior distribution of the number of hidden states, bayesian model averaging is performed on every model using `states` method.

As the other methods, it may return a list with sublists according to the hierarchy of RJaCGH objects.

Value

states	Factor with the hidden state sequence
prob.states	Matrix with the probabilities associated to every states for every observation.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Urriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [summary.RJaCGH](#), [states](#), [plot.RJaCGH](#), [trace.plot](#)

Examples

```
## Not run: y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
  rnorm(100, 0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.5, 5), sigma.tau.sigma.2=rep(0.3, 5),
  sigma.tau.beta=rep(0.7, 5), tau.split.mu=0.5, tau.split.beta=0.5)
fit.genome <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
  burnin=1000, TOT=10000, jump.parameters=jp, max.k=5)
```

```
mo <- modelAveraging(fit.genome)
print(mo) ## End(Not run)
```

```
normal.HMM.likelihood.NH.C
```

Likelihood for non-homogeneous hidden Markov model

Description

This function returns the log-likelihood for RJaCGH model, a hidden Markov model with normal distributed emissions and a non-homogeneous transition matrix as computed by Q.NH.

Usage

```
normal.HMM.likelihood.NH.C(y, x, mu, sigma.2, beta, stat = NULL)
```

Arguments

y	Log Ratios observed
x	Vector of distances between genes
mu	Vector of means for the hidden states
sigma.2	Vector of variances for the hidden states
beta	<i>beta</i> in transition matrix
stat	Vector of initial probabilities. If NULL, a uniforma distribution is assumed.

Details

This function is just an interface for the C routine to compute log-likelihood in RJaCGH model.

Value

It returns a list with the same components passed plus:

loglik	Log-likelihood
--------	----------------

Author(s)

Oscar M. Rueda and Ramon Diaz-Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

Examples

```
## create data
y <- c(rnorm(100, 0, 1), rnorm(50, 3, 1), rnorm(20, -3, 1),
       rnorm(60, 0, 1))
x <- sample(1:1000, 229, replace=FALSE)
x <- x/max(x)
Chrom <- rep(1:23, rep(10, 23))
## same model for all genome
loglik <- 0
for (i in 1:23) {
  loglik <- loglik + normal.HMM.likelihood.NH.C(y=y, x =x, mu=c(-3, 0, 3),
        sigma.2=c(1,1,1), beta=matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3))$loglik
}
loglik
```

plot.pREC_S *Plot number of probes shared by pairs of arrays*

Description

An image plot showing the results of [pREC_S](#) on a group of arrays.

Usage

```
## S3 method for class 'pREC_S':
plot(x, array.labels = NULL, stats = TRUE, col = NULL,
     breaks = NULL, dend = TRUE, method = "single",
     Chrom = NULL, ...)
```

Arguments

x	An object of class pREC_S
array.labels	A vector for alternative labels for the arrays.
Chrom	Chromosome to plot. If NULL, all chromosomes are plotted.
stats	Logical. If TRUE, over every cell the number of common probes and the mean length is printed.
col	A vector of color codes for the image plot.
breaks	Breakpoints for the code color. Must be a vector of length $length(col) + 1$
dend	Logical. If TRUE, a clustering of arrays is performed with hclust and arrays reordered.
method	Clustering method to apply. See hclust . Default is 'single'.
...	Additional arguments passed to image


```
burnin=1000, TOT=1000, jump.parameters=jp, k.max = 4)
Reg1 <- pREC_S(fit.array.genome, p=0.4, freq.array=2,
alteration="Gain")
plot(Reg1)
```

plot.Q.NH

Plot transition probabilities

Description

Plot the probabilities of staying in the same state for a non-homogenous hidden Markov model.

Usage

```
plot.Q.NH(x, beta, q=-beta, col = NULL, ...)
```

Arguments

x	Vector of distances between observations
beta	<i>beta</i> parameter of the transition matrix. Must be a square matrix with the same size as the number of hidden states
q	<i>q</i> parameter of the transition matrix. Must be a square matrix with the same size as the number of hidden states
col	vector of colors for each state. Must be of the same size as the number of hidden states
...	additional arguments passed to plot

Details

Please note that RJaCGH model imposes that q is $-beta$, and distances are normalized to lay between 0 and 1.

Value

A plot is produced showing the probability of staying in the same hidden state versus distance between adjacent genes, for every state.

Author(s)

Oscar M. Rueda and Ramon Diaz-Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

Examples

```
## Model with two hidden states
## Note that RJaCGH normalizes distances to be between 0 and 1
x <- rexp(99)
x <- x/ max(x)
beta <- matrix(c(0, 1, 3, 0), 2, 2)
plot.Q.NH(x=x, beta=beta, q=-beta, col=c(1,2))
```

plot.RJaCGH *'plot' method for RJaCGH objects*

Description

A plot is drawn with information from the fit of a RJaCGH object.

Usage

```
## S3 method for class 'RJaCGH':
plot(x, array=NULL, k = NULL,
     Chrom=NULL, show="average", weights=NULL,
     model.averaging = TRUE, cex=1,
     smoother=FALSE, ...)
```

Arguments

x	any of RJaCGH, RJaCGH.Chrom, RJaCGH.Genome, RJaCGH.array objects
array	Name of the array to be plotted. If NULL, the weighed average of the arrays is plotted.
k	Model to plot (i.e., number of hidden states). If NULL, the most visited is taken (only if a single array is plotted and a single model (genome/chromosome).
Chrom	Chromosome to be plotted (only if a single array is plotted and a different model for each chromosome has been fitted).
show	one of "average" or "frequency". See details.
weights	vector of weights for each array. Must have the length of the number of arrays. If NULL, the weights are uniform.
model.averaging	if TRUE, <code>modelAveraging</code> is performed. If FALSE, a call to <code>states</code> is made to get hidden state sequence.
cex	A numerical value giving the amount by which plotting text and symbols should be scaled relative to the default.
smoother	Logical. Smoothed means by model averaging.
...	additional arguments passed to plot.

Details

Depending on the object and the parameters passed, a different plot is drawn: If `array` in the case of a single model to all genome or `array` and `Chrom` in the case of a different model to each chromosome are passed, a panel with 5 subplots is returned. The first one is a barplot with the posterior distribution of the number of hidden states. The second and third are a density plot of the posterior distribution of means and variances. The fourth one is the probability of staying in the same hidden state, as returned by `plot.Q.NH`, and the last one shows the original observations colored by their hidden state and the probability of being in that hidden state.

On every plot, the 'Normal' state is coloured black. The 'Gain' states are red and the 'Loss' ones green.

If `array` is `NULL` and `show` is 'average', the last one of the plots is drawn, but the hidden state sequence and its probability is computed averaging on all the arrays with weights according to `weights` vector. If `show` is 'frequency', again the last plot is drawn, but the percentage of arrays in which every gene is Gain/Lost is shown, weighted by the `weights` vector.

If `smoother` is `TRUE`, the smoothed mean is drawn. See `smoothMeans`, except when `show` is 'frequency'.

Value

A plot.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [smoothMeans](#), [summary.RJaCGH](#), [modelAveraging](#), [states](#), [trace.plot](#)

Examples

```
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1), rnorm(100,
                                                                    0, 1))
Pos <- round(runif(230))
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))
jp <- list(sigma.tau.mu=rep(0.5, 4), sigma.tau.sigma.2=rep(0.3, 4),
           sigma.tau.beta=rep(0.7, 4), tau.split.mu=0.5, tau.split.beta=0.5)
fit.Chrom <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Chrom",
                   burnin=100, TOT=1000, jump.parameters=jp, k.max=4)
fit.Genom <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome", burnin=100, TOT=1000, jump.pa
fit.none <- RJaCGH(y=y, Pos=Pos, Chrom=NULL, model="None",
                  burnin=100, TOT=1000, jump.parameters=jp, k.max=4)
```

```

plot(fit.Chrom)
plot(fit.Chrom, array="array1")
plot(fit.Genom)
plot(fit.none)

y2 <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
        rnorm(100, 0, 1))

ya <- cbind(y, y2)

fit.Chrom.array <- RJACGH(y=ya, Pos=Pos, Chrom=Chrom, model="Chrom",
                          burnin=100, TOT=1000, jump.parameters=jp, k.max=4)
fit.Genom.array <- RJACGH(y=ya, Pos=Pos, Chrom=Chrom, model="Genome", burnin=100, TOT=1000,
                          jump.parameters=jp, k.max=4)
fit.none.array <- RJACGH(y=ya, Pos=Pos, Chrom=NULL, model="None",
                          burnin=100, TOT=1000, jump.parameters=jp, k.max=4)

plot(fit.Chrom.array)
plot(fit.Genom.array)
plot(fit.none.array)

```

pREC_A

Probabilistic Common Regions for copy number alteration.

Description

This method compute regions of gain/lost copy number with a joint probability of alteration greater than a given threshold.

Usage

```

pREC_A(obj, p, alteration = "Gain", array.weights = NULL,
       verbose = FALSE)

```

Arguments

obj	An object of class 'RJACGH', 'RJACGH.Chrom', 'RJACGH.Genome' or 'RJACGH.array'.
p	Threshold for the minimum joint probability of alteration of the region.
alteration	Either 'Gain' or 'Lost'
array.weights	When 'obj' contains several arrays, the user can give a weight to each of them according to their reliability or precision.
verbose	If TRUE provide more details on what is being done, including intermediate output from the C functions themselves. Only helpful for debugging or if you are bored; this will often write more output than you want.

Details

RJaCGH can compute common regions taking into account the probability of every probe to have an altered copy number. The result is a set of probes whose joint probability (not the product of their marginal probabilities, as returned by `states` or `modelAveraging`) is at least as p or greater.

Please note that if the method returns several sets or regions, the probability of alteration of all of them doesn't have to be over the probability threshold; in other words p is computed for every region, not for all the sequence of regions.

Writing the files with the Vitterbi sequence to disk will be done by default if RJaCGH was run with the default "delete_gzipped = TRUE" (which will happen if the global flag "`__DELETE_GZIPPED`" is TRUE). To preserve disk space, as soon as the gzipped files are read into R (and incorporated into the RJaCGH object), by default they are deleted from disk. Sometimes, however, you might want not to delete them from disk; for instance, if you will continue working from this directory, and you want to save some CPU time. If the files exist in the directory when you call pREC there is no need to write them from R to disk, which allows you to save the time in the "writeBin" calls inside pREC. In this case, you would run RJaCGH with "delete_gzipped = FALSE". Now, if for some reason those files are no longer available (you move directories, you delete them, etc), you should set "force.write.files = TRUE" (pREC will let you know if you need to do so).

`delete.rewritten` helps prevent cluttering the disk. Files with the Viterbi sequence will be written to disk, read by C, and then deleted. Again, no information is lost, since the sequences are stored as part of the RJaCGH object. Note, however, that if you run RJaCGH with "`__DELETE_GZIPPED <- FALSE`" this option has no effect, because it is implicit that you wanted, from the start, to preserve the files. In other words, "delete.rewritten" only has any effect if you either used "force.write.files = TRUE" or if you originally run RJaCGH without preserving the files in disk.

Value

An object of class `pREC_A.none` or `pREC_A.Chromosomes`, depending on whether or not the original RJaCGH had, or not, a Chromosomes component (the later only when the original object was of neither "RJaCGH.Chrom" or "RJaCGH.Genome").

They are lists with a sublist for every region encountered and elements:

<code>start</code>	Start position of the region.
<code>indexStart</code>	index position of the start of the region.
<code>indexEnd</code>	index position of the end of the region.
<code>end</code>	End position of the region.
<code>genes</code>	Number of genes in the region.
<code>prob</code>	Joint probability of gain/loss of the region.

If there are chromosome information (that is, the object inputed is of class `RJaCGH.Chrom`, `RJaCGH.Genome` or `RJaCGH.array` with each array of any of these classes), then this information will be enclosed in a list for each chromosome.

Note

There have been major changes in how pREC is implemented. For details, see "Implementing_pREC_in_C.pdf".

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [states](#), [modelAveraging](#), [print.pREC_A](#) [pREC_S](#)

Examples

```
## MCR for a single array:
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
       rnorm(100,0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
          sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)

fit.genome <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
burnin=1000, TOT=1000, jump.parameters=jp, k.max = 4)
pREC_A(fit.genome, p=0.8, alteration="Gain")
pREC_A(fit.genome, p=0.8, alteration="Loss")

##MCR for two arrays:
z <- c(rnorm(110, 0, 1), rnorm(20, 3, 1),
       rnorm(100,0, 1))
fit.array.genome <- RJaCGH(y=cbind(y,z), Pos=Pos, Chrom=Chrom, model="Genome",
burnin=1000, TOT=1000, jump.parameters=jp, k.max = 4)
pREC_A(fit.array.genome, p=0.4, alteration="Gain")
pREC_A(fit.array.genome, p=0.4, alteration="Loss")
```

pREC_S

Subgroups of arrays that share common alterations

Description

An algorithm to find regions of gain/lost copy number shared by a given proportion of arrays over a probability threshold.

Usage

```
pREC_S(obj, p, freq.array, alteration = "Gain",
       verbose = FALSE)
```

Arguments

obj	An object of class 'RJaCGH.array'.
p	Threshold for the minimum joint probability of the region on every array.
freq.array	Minimum number of arrays that share every region.
alteration	Either 'Gain' or 'Loss'.
verbose	If TRUE provide more details on what is being done, including intermediate output from the C functions themselves. Only helpful for debugging or if you are bored; this will often write more output than you want.

Details

This algorithm, as `pREC_A` computes probabilistic common regions but instead of finding regions that have a joint probability of alteration over all arrays, `pREC_S` searches for regions that have a probability of alteration higher than a threshold in at least a minimum number of arrays. So, `pREC_S` finds subsets of arrays that share subsets of alterations. Please note that if the method returns several sets or regions, the probability of alteration of all of them doesn't have to be over the probability threshold; in other words `p` is computed for every region, not for all the sequence of regions.

Writing the files with the Vitterbi sequence to disk will be done by default if RJaCGH was run with the default "delete_gzipped = TRUE" (which will happen if the global flag ".__DELETE_GZIPPED" is TRUE). To preserve disk space, as soon as the gzipped files are read into R (and incorporated into the RJaCGH object), by default they are deleted from disk. Sometimes, however, you might want not to delete them from disk; for instance, if you will continue working from this directory, and you want to save some CPU time. If the files exist in the directory when you call `pREC` there is no need to write them from R to disk, which allows you to save the time in the "writeBin" calls inside `pREC`. In this case, you would run RJaCGH with "delete_gzipped = FALSE". Now, if for some reason those files are no longer available (you move directories, you delete them, etc), you should set "force.write.files = TRUE" (`pREC` will let you know if you need to do so).

`delete.rewritten` helps prevent cluttering the disk. Files with the Viterbi sequence will be written to disk, read by C, and then deleted. Again, no information is lost, since the sequences are stored as part of the RJaCGH object. Note, however, that if you run RJaCGH with ".__DELETE_GZIPPED <- FALSE" this option has no effect, because it is implicit that you wanted, from the start, to preserve the files. In other words, "delete.rewritten" only has any effect if you either used "force.write.files = TRUE" or if you originally run RJaCGH without preserving the files in disk.

Value

An object of class `pREC_S.none`, or `pREC_S.Chromosomes` depending on whether or not the original RJaCGH had, or not, a Chromosomes component (the later only when the original object was of neither "RJaCGH.Chrom" or "RJaCGH.Genome").

They are lists with a sublist for every region encountered and elements:

start	Start position of the region.
indexStart	index position of the start of the region.
indexEnd	index position of the end of the region.
end	End position of the region.

members Arrays that share the region.

If there are chromosome information, this information will be enclosed in a list for each chromosome.

Note

There have been major changes in how pREC is implemented. For details, see "Implementing_pREC_in_C.pdf".

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [states](#), [modelAveraging](#), [print.pREC_S](#) [plot.pREC_S](#) [pREC_A](#)

Examples

```
## MCR for a single array:
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
       rnorm(100,0, 1))
z <- c(rnorm(110, 0, 1), rnorm(20, 3, 1),
       rnorm(100,0, 1))
zz <- c(rnorm(90, 0, 1), rnorm(40, 3, 1),
        rnorm(100,0, 1))

Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
           sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)

fit.array.genome <- RJaCGH(y=cbind(y,z,zz),
                          Pos=Pos, Chrom=Chrom, model="Genome",
                          burnin=1000, TOT=1000, jump.parameters=jp, k.max = 4)
pREC_S(fit.array.genome, p=0.4, freq.array=2,
        alteration="Gain")
pREC_S(fit.array.genome, p=0.4, freq.array=2, alteration="Loss")
```

print.pREC_A *Method for printing probabilistic common region.*

Description

A print method for `pREC_A` objects

Usage

```
## S3 method for class 'pREC_A':  
print(x, ...)  
## S3 method for class 'pREC_A.none':  
print(x, ...)  
## S3 method for class 'pREC_A.Chromosomes':  
print(x, ...)
```

Arguments

`x` An object of class `pREC_A`, `pREC_A.Chromosomes`, `pREC_A.RJaCGH.none`.
`...` Additional arguments passed to `print`. Currently ignored.

Value

A data.frame is printed with as many rows as regions found and with columns containing chromosome where the region is, position of start and end of the region, number of genes in it and joint probability.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [states](#), [modelAveraging](#), [pREC_A](#)

Examples

```
## MCR for a single array:  
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),  
      rnorm(100, 0, 1))  
Pos <- sample(x=1:500, size=230, replace=TRUE)
```

```

Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
           sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)

fit.genome <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
burnin=100, TOT=1000, jump.parameters=jp, k.max = 4)
pREC_A(fit.genome, p=0.8, alteration="Gain")
pREC_A(fit.genome, p=0.8, alteration="Loss")

##MCR for two arrays:
z <- c(rnorm(110, 0, 1), rnorm(20, 3, 1),
       rnorm(100,0, 1))
fit.array.genome <- RJaCGH(y=cbind(y,z), Pos=Pos, Chrom=Chrom, model="Genome",
burnin=100, TOT=1000, jump.parameters=jp, k.max = 4)
pREC_A(fit.array.genome, p=0.4, alteration="Gain")
pREC_A(fit.array.genome, p=0.4, alteration="Loss")

```

print.pREC_S *Method for printing probabilistic common regions*

Description

A print method for `pREC_S` objects.

Usage

```

## S3 method for class 'pREC_S':
print(x, ...)
## S3 method for class 'pREC_S.Chromosomes':
print(x, ...)
## S3 method for class 'pREC_S.none':
print(x, ...)

```

Arguments

`x` An object of class `pREC_S`, `pREC_S.Chromosomes` or `pREC_S.none`.
`...` Additional arguments passed to `print`. Currently ignored.

Value

A data.frame is printed with as many rows as regions found and with columns containing chromosome where the region is, position of start and end of the region and the arrays that belong to the region.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[pREC_S](#)

Examples

```

y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
       rnorm(100,0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
          sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)

z <- c(rnorm(110, 0, 1), rnorm(20, 3, 1),
       rnorm(100,0, 1))
zz <- c(rnorm(90, 0, 1), rnorm(40, 3, 1),
        rnorm(100,0, 1))

fit.array.genome <- RJaCGH(y=cbind(y,z,zz),
                          Pos=Pos, Chrom=Chrom, model="genome",
                          burnin=1000, TOT=1000, jump.parameters=jp, k.max = 4)
pREC_S(fit.array.genome, p=0.4, freq.array=2,
        alteration="Gain")
pREC_S(fit.array.genome, p=0.4, freq.array=2, alteration="Loss")

```

```
print.summary.RJaCGH
```

print summary of RJaCGH fit

Description

The list stored in [summary.RJaCGH](#) is printed.

Usage

```

## S3 method for class 'summary.RJaCGH':
print(x, ...)
## S3 method for class 'summary.RJaCGH.Chrom':
print(x, ...)
## S3 method for class 'summary.RJaCGH.Genome':
print(x, ...)

```

Arguments

`x` a `summary.RJaCGH` object
`...` Additional arguments passed to `print`

Value

The summary is printed

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. *PLoS Comput Biol.* 2007;3(6):e122

See Also

`RJaCGH`, `states`, `modelAveraging`, `plot.RJaCGH`, `trace.plot`

Examples

```
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1), rnorm(100, 0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))
jpb <- list(sigma.tau.mu=rep(0.5, 5), sigma.tau.sigma.2=rep(0.3, 5),
sigma.tau.beta=rep(0.7, 5), tau.split.mu=0.5, tau.split.beta=0.5)
fit.chrom <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Chrom",
burnin=10, TOT=100, jump.parameters=jpb, k.max = 5)
summary(fit.chrom)
```

Description

This function returns the transition matrix for a given distance between genes.

Usage

```
Q.NH(beta, x, q=-beta)
```

Arguments

beta	beta parameter of transition matrix. Must be a square matrix with dimension equal to the number of hidden states.
x	Distance between genes to compute the transition matrix. Must be a scalar.
q	q parameter of transition matrix. Note that in RJaCGH q is always -beta (details below). Must be a square matrix with dimension equal to the number of hidden states.

Details

RJaCGH assumes a non-homogeneous transition matrix with this form: $Q[i,j] = \exp(-\text{beta}[i,j] + \text{beta}[i,j]*x) / \sum(i,.) \exp(-\text{beta}[i,.] + \text{beta}[i,.]*x)$

All $\text{beta}[i,i]$ are constrained to be zero, for the model to be identifiable.

All $\text{beta}[i,j]$ are positive. This model is chosen for its simplicity and because it agrees with biological assumptions, for it makes the probabilities of staying in the same state decreasing with the distance to the next gene, and reaches a probability of 1/number of hidden state when the distance to the next gene is the maximum.

To avoid overflow errors, RJaCGH normalizes distances to be between 0 and 1.

Value

~Describe the value returned A matrix with the transition probabilities for that distance.

Author(s)

Oscar M. Rueda and Ramon Diaz-Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

plot.Q.NH

Examples

```
## Model with two hidden states
## Note that RJaCGH normalizes distances to be between 0 and 1
beta <- matrix(c(0, 1, 3, 0), 2, 2)
Q.NH(beta=beta, x=0.4)
```

relabelStates *Relabelling of hidden states to biological states of alteration.*

Description

For every model, each hidden state is assigned to a state of copy number alteration ('normal', 'loss1', 'loss2', 'gain1', 'gain2'...)

Usage

```
relabelStates(obj, normal.reference = 0,
              window = NULL, singleState = FALSE,
              array=NULL, Chrom=NULL)
## S3 method for class 'RJaCGH':
relabelStates(obj, normal.reference = 0,
              window = NULL, singleState = FALSE,
              array=NULL, Chrom=NULL)
```

Arguments

obj	An object returned from RJaCGH() of class 'RJaCGH', 'RJaCGH.Chrom', 'RJaCGH.Genome'.
normal.reference	The value considered as the mean of the normal state. See details. By default is 0.
window	Multiplier of the standard deviation of the data to determine the width of the normal state. See details. Default (window = NULL) is 1.
singleState	If TRUE, each state is assigned a probability of 1 of being Gained or Lost or Normal, and 0 of being anything else. This mimics the behavior of previous versions of RJaCGH. See details.
array	Vector of arrays to be relabeled. If NULL, all of them.
Chrom	Vector of chromosome to be relabeled (only if a different model has been fitted to each of them). If NULL, all of them are labeled.

Details

A relabelling of hidden states is performed to match biological states. We first define an upper and lower limit as: normal.reference +/- window. Next, we compute as the probability that a given state is gained the area beyond the upper limit, and the area up to the lower limit as the probability that a given state is lost. The areas are obtained from a Normal with posterior estimates of mean and standard deviation for each hidden state. The probability of being in a normal state is defined as 1 - prob.gained - prob.loss.

Labels are assigned based on the state with highest probability. If we set singleState = TRUE, all the probability is assigned to a single condition, the one with largest probability.

Value

An object of the same class as `obj` with hidden states relabelled.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJACGH](#)

Examples

```

y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
      rnorm(100, 0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
          sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)

fit.chrom <- RJACGH(y=y, Pos=Pos, Chrom=Chrom, model="Chrom",
                  burnin=10, TOT=1000, k.max = 4,
                  jump.parameters=jp)

plot(fit.chrom)
fit.chrom.2 <- relabelStates(fit.chrom, normal.reference=3)
plot(fit.chrom.2)

```

Description

This function fits a non-homogeneous hidden Markov model to CGH data through bayesian methods and Reversible Jump Markov chain Montecarlo.

Usage

```
RJaCGH(y, Chrom = NULL, Start=NULL, End=NULL, Pos = NULL,
        Dist=NULL, probe.names=NULL, maxVar=NULL, model = "Genome",
        var.equal=TRUE, max.dist=NULL, normal.reference=0,
        window = NULL, burnin = 10000, TOT =10000,
        k.max = 6, stat = NULL, mu.alfa = NULL, mu.beta = NULL,
        s1=NULL, s2=NULL, init.mu=NULL, init.sigma.2=NULL, init.beta=NULL,
        prob.k = NULL, jump.parameters=list(),
        start.k = NULL, RJ=TRUE, NC = 1, deltaT = 2, delete_gzipped =
        .__DELETE_GZIPPED, singleState = FALSE)
```

Arguments

<code>y</code>	Vector with Log Ratio observations.
<code>Chrom</code>	Vector with Chromosome indicator.
<code>Start</code>	Vector with start positions of the probes.
<code>End</code>	Vector with end positions of the probes.
<code>Pos</code>	Vector with Positions of every gene. They can be absolute to the genome or relative to the chromosome. They should be ordered within every chromosome. This is, the arrays must be ordered by their positions in the genome. They must be integers. Positions can be specified with <code>Start</code> , <code>End</code> or <code>Pos</code> , but only in one of the two ways.
<code>Dist</code>	Optional vector of distances between genes. It should be a vector of length $length(y)-1$. Note that when <code>Chrom</code> is not <code>NULL</code> , every last value of every Chromosome is not used.
<code>probe.names</code>	Character vector with the number of the probes.
<code>maxVar</code>	Maximum value for the variance of the states. If <code>NULL</code> , the range of the data is chosen.
<code>model</code>	if <code>model="Genome"</code> , the same model is fitted for the whole genome. If <code>model="Chrom"</code> , a different model is fitted for each chromosome.
<code>var.equal</code>	Logical. If <code>TRUE</code> the variances of the hidden states are restricted to be the same.
<code>max.dist</code>	maximal distance between spots. When two spots have a distance between them as far or further than <code>max.dist</code> , they are considered independent. That is, the state of that spot does not affect the state of the other. If <code>NULL</code> (the default) the maximum <code>Dist</code> or maximum difference in <code>Pos</code> is taken.
<code>normal.reference</code>	The value considered as the mean of the normal state. See details. By default is 0.
<code>window</code>	Multiplier of the standard deviation of the data to determine the width of the normal state. See details. Default (<code>window = NULL</code>) is 1.
<code>burnin</code>	Number of burn-in iterations in the Markov Chain
<code>TOT</code>	Number of iterations after the burn-in
<code>k.max</code>	Maximum number of hidden states to fit.

<code>stat</code>	Initial Distribution for the hidden states. Must be a vector of size $1 + 2 + \dots + k_{max}$. If NULL, it is assumed a uniform distribution for every model.
<code>mu.alfa</code>	Hyperparameter. See details
<code>mu.beta</code>	Hyperparameter. See details
<code>prob.k</code>	Hyperparameter. See details
<code>jump.parameters</code>	List with the parameters for the MCMC jumps. See details.
<code>start.k</code>	Initial number of states. if NULL, a random draw from <code>prob.k</code> is chosen.
<code>RJ</code>	Logical. If TRUE, Reversible Jump is performed. If not, MCMC over a fixed number of hidden states. Note that if FALSE, most of the methods for extracting information won't work.
<code>s1</code>	Standard deviation for the creation of a new mean in the birth move (first attempt). If NULL, it is the prior <code>mu.beta</code> .
<code>s2</code>	Standard deviation for the creation of a new mean in the birth move (second attempt). If NULL, it is the prior <code>mu.beta</code> .
<code>init.mu</code>	Starting values for <code>mu</code> for the chain. See details
<code>init.sigma.2</code>	Starting values for <code>sigma.2</code> for the chain. See details
<code>init.beta</code>	Starting values for <code>beta</code> for the chain. See details
<code>NC</code>	Number of coupled parallel chains. See details
<code>deltaT</code>	Heat parameter for tempering the parallel chains. See details
<code>delete_gzipped</code>	Should temporal files with viterbi paths be deleted?
<code>singleState</code>	Logical. If TRUE, each hidden state is classified into a state of copy number alteration. If FALSE, a probability is assigned to each hidden state of being gained and lost. See relabelStates

Details

RJCGH fits the following bayesian model: There is a priori distribution for the number of hidden states (different copy numbers) as stated by `prob.k`. If NULL, a uniform distribution between 1 and `k.max` is used.

The hidden states follow a normal distribution which mean (`mu`) follows itself a normal distribution with mean `mu.alfa` and stdev `mu.beta`. If NULL, these are the median of the data and the range. The square root of the variance (`sigma.2`) of the hidden states follows a uniform distribution between 0 and `maxVar`.

The model for the transition matrix is based on a random matrix *beta* whose diagonal is zero. The transition matrix, *Q*, has the form: $Q[i,j] = \exp(-\beta[i,j] + \beta[i,j]*x) / \sum(i,.) \exp(-\beta[i,.] + \beta[i,]*x)$

The prior distribution for *beta* is gamma with parameters 1, 1. The *x* are the distances between positions, normalized to lay between zero and 1 ($x = \text{diff}(Pos) / \max(\text{diff}(Pos))$)

RJCGH performs Markov Chain MonteCarlo with Reversible Jump to sample for the posterior distribution. `NC` sets the number of chains from we will sample in parallel. Each of them is tempered

in order to escape from local maximum. The temper parameter is `deltaT`, and it can be a value greater than 0. each chain is tempered according to:

$$1/(1 + \text{deltaT} * \text{NC})$$

Every sweep is performed for all chains and has 3 steps plus another one common for all of them:

1.- A Metropolis-Hastings move is used to update, for a fixed number of hidden states, `mu`, `sigma.2` and `beta`. A symmetric proposal with a normal distribution and standard deviation `sigma.tau.mu`, `sigma.tau.sigma.2` and `sigma.tau.beta` is sampled.

2.- A transdimensional move is chosen, between birth (a new hidden state is sampled from the prior) or death (an existing hidden state is erased). Both moves are tried using delayed rejection. That is, if the move is rejected, is given another try. The means for the new state are drawn for the priors, but the standard deviation can be set for the two stages with parameters `s1` and `s2`.

3.- Another transdimensional move is performed; an split move (divide an existing state in two) or a combine move (join two adjacent states). The length of the split is sampled from a normal distribution with standard deviation `tau.split.mu` for the `mu` and `tau.split.beta` for `beta`.

4.- If `NC` is greater than 1, a swap move is tried to exchange information from two of the coupled parallel chains.

`jump.parameters` must be a list with the parameters for the moves. It must have components `sigma.tau.mu`, `sigma.tau.sigma.2`, `sigma.tau.beta` These are vectors of length `k.max`. `tau.split.mu`, `tau.split.beta` are vectors of length 1. If any of them is NULL, a call to the internal function `get.jump()` is made to find 'good' values.

A relabelling of hidden states is performed to match biological states. See details in [relabelStates](#).

The initial values of the chain are drawn from an overdispersed distribution. One can start the chain in a given point with the parameters `start.k` (model to start from `1, \dots, \text{max.k}`) and the initial values `init.mu`, `init.sigma.2` (vectors of dimension `start.k`) and `init.beta` (matrix of positive values with `start.k` rows and `start.k` columns). The diagonal must be zero.

Value

The object returned follows a hierarchy:

If `y` is a matrix or data.frame (i.e., several arrays), an object of class `RJaCGH.array` is returned, with components:

<code>[[[]]</code>	A list with an object of corresponding class (see below) for every array.
<code>array.names</code>	Vector with the names of the arrays.
<code>[[[]]</code>	a list with as many objects as <code>k.max</code> , with the fits.
<code>k</code>	sequence of number of hidden states sampled.
<code>prob.b</code>	Number of birth moves performed in first and second proposal.(Includes burn-in.
<code>prob.d</code>	Number of death moves performed in first and second proposal.(Includes burn-in.
<code>prob.s</code>	Number of split moves performed (Includes burn-in.
<code>prob.c</code>	Number of combine moves performed (Includes burn-in.
<code>prob.e</code>	Number of exchange (swap) moves performed between tempered chains and with cool (main) chain.

<code>y</code>	<code>y</code> vector.
<code>Pos</code>	<code>Pos</code> vector.
<code>model</code>	model.
<code>Chrom</code>	Chromosome vector.
<code>x</code>	<code>x</code> vector of distances between genes.
<code>viterbi</code>	A list with as many components as chromosomes. For each chromosome, a list, with at least three components: <code>gzipped_sequence</code> : the compacted and gzipped sequence of states; <code>num_sequences</code> : the number of sequences in that compacted set of sequences; <code>sum_mcmc_iter</code> : the number of times a viterbi sequence was obtained. All these are only of use for calls to the <code>pREC</code> functions (<code>pREC_A</code> and <code>pREC_S</code>).
<code>[[[]]</code>	a list with as many components as chromosomes, of class <code>RJaCGH</code> (See below).
<code>Pos</code>	<code>Pos</code> vector.
<code>Start</code>	Start positions.
<code>End</code>	End positions.
<code>probe.names</code>	Names of the probes.
<code>model</code>	model.
<code>Chrom</code>	Chromosome vector.
<code>viterbi</code>	Identical structure as above.
<code>mu</code>	a matrix with the means sampled
<code>sigma.2</code>	a matrix with the variances sampled
<code>beta</code>	an array of dimension 3 with beta values sampled
<code>stat</code>	vector of initial distribution
<code>loglik</code>	log likelihoods of every MCMC iteration
<code>prob.mu</code>	probability of acceptance of <code>mu</code> in the Metropolis-Hastings step.
<code>prob.sigma.2</code>	probability of acceptance of <code>sigma.2</code> in the Metropolis-Hastings step.
<code>prob.beta</code>	probability of acceptance of <code>beta</code> in the Metropolis-Hastings step.
<code>state.labels</code>	Labels of the biological states.
<code>prob.states</code>	Marginal posterior probabilities of belonging to every hidden state.

The number of rows of components `mu`, `sigma.2` and `beta` is random, because it depends on the number of times a particular model is visited and on the number of moves between models, because when we visit a new model we also explore the space of its means, variances and parameters of its transition functions.

Note

The data must be ordered by chromosome and within chromosome by position.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

- Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122
- Cappe, Moulines and Ryden, 2005. Inference in Hidden Markov Models. Springer.
- Green, P.J. (1995) Reversible Jump Markov Chain Monte Carlo computation and Bayesian model determination. Biometrika, 82, 711-732.
- Green, P.J. and Antonietta, M. (2001) Delayed Rejection in Reversible Jump Metropolis Hastings. Biometrika, 88 (4), 1035-1053.
- Geyer, C. J. (1991). Markov Chain Monte Carlo Maximum Likelihood. Proceedings of the 23th Symposium on the Interface, 156-163.

See Also

[summary.RJaCGH](#), [states](#), [modelAveraging](#), [plot.RJaCGH](#), [trace.plot](#), [relabelStates](#), [pREC_A](#), [pREC_S](#)

Examples

```
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
      rnorm(100,0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
          sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)

fit.chrom <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Chrom",
                  burnin=10, TOT=1000, k.max = 4,
                  jump.parameters=jp)
##RJ results for chromosome 5
table(fit.chrom[["array1"]][[5]]$k)
fit.genome <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
                  burnin=100, TOT=1000, jump.parameters=jp, k.max = 4)
## Results for the model with 3 states:
summary(fit.genome)
```

simulateRJaCGH

Simulate observations form a hidden Markov model with non-homogeneous transition probabilities.

Description

This function simulates observations from a hidden Markov model with normal distributed observations and non-homogeneous transition matrix.

Usage

```
simulateRJaCGH(n, x = NULL, mu, sigma.2, beta, start)
```

Arguments

n	Number of observations to simulate
x	Distance to the next observation. Must be a vector of size n-1 and normalized between zero and one. If NULL, a vector of zeros is taken
mu	Vector of means for the hidden states
sigma.2	Vector of variances for the hidden states
beta	beta parameter of the transition matrix. Must be a square matrix with the same size as the number of hidden states.
start	Starting states of the sequence. Must be an integer from 1 to the number of hidden states.

Details

Please note that in RJaCGH model, parameter q is taken as $-\text{beta}$

Value

A list with components

states	Sequence of hidden states
y	Observations

Author(s)

Oscar M. Rueda and Ramon Diaz-Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

Q.NH, RJaCGH

Examples

```
beta <- matrix(c(0, 5, 1, 1, 0, 1, 3, 5, 0), 3)
obs <- simulateRJaCGH(n=200, x=rexp(199), mu=c(-3, 0, 3), sigma.2=c(1,1,1),
beta=beta, start=2)
plot(obs$y, col=obs$states)
```

`smoothMeans`*Smoothed posterior mean*

Description

Smoothed posterior mean for every probe after fitting a RJaCGH model.

Usage

```
smoothMeans(obj, array=NULL, Chrom = NULL, k = NULL)
## S3 method for class 'RJaCGH':
smoothMeans(obj, array=NULL, Chrom = NULL, k=NULL)
```

Arguments

<code>obj</code>	An RJaCGH object, of class 'RJaCGH'
<code>array</code>	Vector of names of the array to get smoothed means. If NULL, all of them.
<code>Chrom</code>	Vector of the chromosomes. If NULL, all of them.
<code>k</code>	Number of states (or model) to get the smoothed means from. If NULL, Bayesian Model Averaging is used.

Details

For a model with k hidden states, the mean from the MCMC samples from μ is computed for every hidden state. Then, for every probe these means are averaged by its posterior probability of belonging to every hidden state. If k is NULL, then these smoothed means are computed for every model and averaged by the posterior probability of each model.

Value

A matrix. Columns of the matrix are arrays (i.e., for an RJaCGH object with a single array, the value is a one column matrix). Each column contains the smoothed means for every probe.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [plot.RJaCGH](#)

Examples

```
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1),
      rnorm(100, 0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))

jp <- list(sigma.tau.mu=rep(0.5, 4), sigma.tau.sigma.2=rep(0.3, 4),
          sigma.tau.beta=rep(0.7, 4), tau.split.mu=0.5, tau.split.beta=0.5)

fit.genome <- RJacGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
                   burnin=10, TOT=1000, k.max = 4,
                   jump.parameters=jp)

plot(y~Pos)
lines(smoothMeans(fit.genome) ~ Pos)
```

snijders

Public CGH data of Snijders

Description

15 human cell strains with known karyotype (<http://locus.umdnj.edu/nigms>). Object taken from GLAD package 1.6.0. (Huppé and Barillot).

Usage

```
data(snijders)
```

Source

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

References

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

Philippe Huppé (2005). GLAD: Gain and Loss Analysis of DNA. R package version 1.6.0. <http://bioinfo.curie.fr>

Examples

```
data(snijders)
```

states	<i>'states' method for RJaCGH objects</i>
--------	---

Description

Methods for estimating the hidden state sequence of a RJaCGH model.

Usage

```
states(obj, array=NULL, Chrom=NULL, k=NULL)
## S3 method for class 'RJaCGH':
states(obj, array=NULL, Chrom=NULL, k=NULL)
```

Arguments

obj	any of RJaCGH, RJaCGH.Chrom, RJaCGH.Genome objects
array	vector of arrays to get the states from.
Chrom	vector of chromosomes to get the states from.
k	Model to summarize (i.e., number of hidden states). If NULL, the most visited is taken.

Details

The posterior probability of the hidden state sequence is computed via viterbi.

The state with more observatios is called 'Normal'. Those with bigger means than it are called 'Gain', 'Gain1'... and those with lesser means are called 'Loss', 'Loss1',...

Depending on the hierarchy of the object, it can return lists with sublists, as in [RJaCGH](#).

Value

states	Factor with the hidden state sequence
prob.states	Matrix with the probabilities associated to every states for every observation.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [summary.RJaCGH](#), [modelAveraging](#), [plot.RJaCGH](#), [trace.plot](#),

Examples

```

y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1), rnorm(100,
0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))
jp <- list(sigma.tau.mu=rep(0.05, 4), sigma.tau.sigma.2=rep(0.03, 4),
          sigma.tau.beta=rep(0.07, 4), tau.split.mu=0.1, tau.split.beta=0.1)
fit.genome <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
burnin=100, TOT=1000, jump.parameters=jp, k.max = 4)
states(fit.genome)

```

summary.RJaCGH

*Summarizing RJaCGH models***Description**

'summary' method for objects of class 'RJaCGH'.

Usage

```

## S3 method for class 'RJaCGH':
summary(object, array=NULL, Chrom=NULL,
k = NULL, point.estimator = "median",
quantiles=NULL, ...)

```

Arguments

object	RJaCGH objects
array	vector of names of arrays to summarize. If NULL, all of them.
Chrom	vector of chromosomes to summarize. If NULL, all of them.
k	Model to summarize (i.e., number of hidden states). If NULL, the most visited is taken.
point.estimator	Type of point estimator for <i>mu</i> , <i>sigma.2</i> and <i>beta</i> . It can be "mean", "median" or "mode".
quantiles	A vector of probabilities for the quantiles of the posterior distribution of means and variances.
...	Additional arguments passed to summary.

Details

Depending of the arguments passed, a list with contains sublists can be returned, similarly to RJaCGH and similar objects of the family. The point estimator "mode" is simply the max value obtained in a kernel density estimation through the function [density](#)

Value

k	Frequencies of the hidden states visited by the sampler.
mu	Quantiles of the posterior distribution of <i>mu</i>
sigma.2	Quantiles of the posterior distribution of <i>sigma.2</i>
beta	Point estimator of <i>beta</i>
stat	Initial distribution of the hidden states.

Author(s)

Oscar M. Rueda and Ramon Diaz Uriarte

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [states](#), [modelAveraging](#), [plot.RJaCGH](#), [trace.plot](#),

Examples

```
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1), rnorm(100,
0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))
jp <- list(sigma.tau.mu=rep(0.5, 5), sigma.tau.sigma.2=rep(0.3, 5),
sigma.tau.beta=rep(0.7, 5), tau.split.mu=0.5, tau.split.beta=0.5)
fit.chrom <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Chrom",
burnin=10, TOT=100, jump.parameters=jp, k.max = 5)
summary(fit.chrom)
```

trace.plot

Trace plot for 'RJaCGH' object

Description

A trace plot with the trajectory of the Markov Chain.

Usage

```
trace.plot(x, k = NULL, array = NULL, Chrom = NULL, main.text = NULL)
```

Arguments

x	any of RJaCGH, RJaCGH.Chrom, RJaCGH.Genome, RJaCGH.array objects
k	Model to plot (i.e., number of hidden states). If NULL, the most visited is taken.
array	the name of the array to plot.
Chrom	the number of the chromosome to plot.
main.text	Main text of the plot

Details

This is simply a call to `matplot` to show the values sampled in the chain.

The colors does not correspond to any particular level of gain/loss.

Value

A plot is drawn.

Author(s)

Oscar M. Rueda and Ramon Diaz

References

Rueda OM, Diaz-Uriarte R. Flexible and Accurate Detection of Genomic Copy-Number Changes from aCGH. PLoS Comput Biol. 2007;3(6):e122

See Also

[RJaCGH](#), [summary.RJaCGH](#), [modelAveraging](#), [plot.RJaCGH](#), [states](#),

Examples

```
y <- c(rnorm(100, 0, 1), rnorm(10, -3, 1), rnorm(20, 3, 1), rnorm(100,
0, 1))
Pos <- sample(x=1:500, size=230, replace=TRUE)
Pos <- cumsum(Pos)
Chrom <- rep(1:23, rep(10, 23))
jp <- list(sigma.tau.mu=rep(0.5, 5), sigma.tau.sigma.2=rep(0.3, 5),
sigma.tau.beta=rep(0.7, 5), tau.split.mu=0.5, tau.split.beta=0.5)
fit.genome <- RJaCGH(y=y, Pos=Pos, Chrom=Chrom, model="Genome",
burnin=10, TOT=100, jump.parameters=jp, k.max = 5)
trace.plot(fit.genome, array="array1")
```

Index

*Topic **datasets**

`sniijders`, 29

*Topic **models**

`genomePlot`, 2

`modelAveraging`, 3

`normal.HMM.likelihood.NH.C`, 4

`plot.pREC_S`, 6

`plot.Q.NH`, 7

`plot.RJaCGH`, 8

`pREC_A`, 11

`pREC_S`, 13

`print.pREC_A`, 15

`print.pREC_S`, 17

`print.summary.RJaCGH`, 18

`Q.NH`, 19

`relabelStates`, 20

`RJaCGH`, 22

`simulateRJaCGH`, 27

`smoothMeans`, 28

`states`, 30

`summary.RJaCGH`, 31

`trace.plot`, 33

`density`, 32

`genomePlot`, 2

`gm00143` (*sniijders*), 29

`gm01524` (*sniijders*), 29

`gm01535` (*sniijders*), 29

`gm01750` (*sniijders*), 29

`gm02948` (*sniijders*), 29

`gm03134` (*sniijders*), 29

`gm03563` (*sniijders*), 29

`gm03576` (*sniijders*), 29

`gm04435` (*sniijders*), 29

`gm05296` (*sniijders*), 29

`gm07081` (*sniijders*), 29

`gm07408` (*sniijders*), 29

`gm10315` (*sniijders*), 29

`gm13031` (*sniijders*), 29

`gm13330` (*sniijders*), 29

`hclust`, 6, 7

`image`, 6, 7

`matplot`, 33

`modelAveraging`, 3, 9–12, 15, 16, 19, 26, 31–33

`normal.HMM.likelihood.NH.C`, 4

`plot.pREC_S`, 6, 15

`plot.Q.NH`, 7, 9

`plot.RJaCGH`, 4, 8, 19, 26, 29, 31–33

`pREC_A`, 11, 13, 15, 16, 25, 26

`pREC_S`, 6, 7, 12, 13, 17, 25, 26

`print`, 17, 18

`print.pREC_A`, 12, 15

`print.pREC_S`, 15, 17

`print.summary.RJaCGH`, 18

`Q.NH`, 19

`relabelStates`, 20, 23, 24, 26

`RJaCGH`, 4, 10, 12, 15, 16, 19, 21, 22, 28–33

`simulateRJaCGH`, 27

`smoothMeans`, 9, 10, 28

`sniijders`, 29

`states`, 3, 4, 9–12, 15, 16, 19, 26, 30, 32, 33

`summary.RJaCGH`, 4, 10, 18, 26, 31, 31, 33

`trace.plot`, 4, 10, 19, 26, 31, 32, 33