Package 'rqubic'

October 17, 2020

Type Package

Title Qualitative biclustering algorithm for expression data analysis in R

Version 1.34.0

Date 2015-04-09

Author Jitao David Zhang, with inputs from Laura Badi and Martin Ebeling

Maintainer Jitao David Zhang <jitao_david.zhang@roche.com>

Description This package implements the QUBIC algorithm introduced by Li et al. for the qualitative biclustering with gene expression data.

Imports methods, Biobase, BiocGenerics, biclust

Suggests RColorBrewer

Collates onload.R AllClasses.R AllMethods.R c_output_parser.R r_qubic_implementation.R rqubic_to_c_funcs.R combineBiclust.R

biocViews Microarray, Clustering

License GPL-2

LazyLoad yes

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

git_branch RELEASE_3_11

git_last_commit 555a638

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

R topics documented:

`rqubic-package` *Qualitative biclustering algorithm for expression data analysis*

Description

QUBIC is a qualitative biclustering algorithm for high-throughput expression data analysis. `rqubic` package implements this algorithm in R, partly with the codes contributed by Haibao Tang and Qin Ma (version 0.23 released without any limitation).

The `rqubic` package also provides parsers for the command line tool of qubic written in C.

Details

Package:	rqubic
Type:	Package
Version:	1.5
Date:	2011-04-11
License:	LGPL-2
LazyLoad:	yes

Part of the source code in C is modified from the source code of the QUBIC command line tool (in C) provided by Haibao Tang and Qin Ma <maqin@csbl.bmb.uga.edu>, downloaded from http://csbl.bmb.uga.edu/~maqin/bicluster/ on 01.03.2011, version 0.23.

Source code of QUBIC also uses open-source data structure library codes. See the README file included in the QUBIC command line tool source.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>, Laura Badi and Martin Ebeling Maintainer: Jitao David Zhang <jitao_david.zhang@roche.com>

References

Li et al. (2009) *QUBIC: a qualitative biclustering algorithm for analyses of gene expression data* Nucleic Acids Research 37:e101

`combineBiclusts-methods`

Combine two or more Biclust objects into one

Description

Combining two or more Biclust objects into one object. These objects must be of same dimension, namely have same numbers of features and samples, although the numbers of biclusters do not have to be the same (and usually are not).

Methods

signature(x = "Biclust", y = "Biclust") Method for any Biclust objects

signature(x = "QUBICBiclusterSet", y = "QUBICBiclusterSet") Method for QUBICBiclusterSet only. Besides combining biclusters, they will also combine parameters and information stored in each QUBICBiclusterSet into the returning object.

signature(x = "list", y = "missing") Method for a list of Biclust or QUBICBiclusterSet objects.

Examples

```
data(sample.ExpressionSet, package="Biobase")
re1.discret <- quantileDiscretize(sample.ExpressionSet, rank=1L)
re1.sel.seeds <- generateSeeds(re1.discret, minColWidth=2L)
re1.blocks <- quBicluster(re1.sel.seeds,
    re1.discret,
    report.no=50L,
    filter.proportion=0.1)

re2.discret <- quantileDiscretize(sample.ExpressionSet, rank=2L)
re2.sel.seeds <- generateSeeds(re2.discret, minColWidth=2L)
re2.blocks <- quBicluster(re2.sel.seeds,
    re2.discret,
    report.no=50L,
    filter.proportion=0.1)

re3.discret <- quantileDiscretize(sample.ExpressionSet, rank=3L)
re3.sel.seeds <- generateSeeds(re3.discret, minColWidth=2L)
re3.blocks <- quBicluster(re2.sel.seeds,
    re2.discret,
    report.no=50L,
    filter.proportion=0.1)

re12.blocks <- combineBiclusts(re1.blocks, re2.blocks)
re123.blocks <- combineBiclusts(re1.blocks, re2.blocks, re3.blocks)
re123.list.blocks <- combineBiclusts(list(re1.blocks, re2.blocks,
re3.blocks))
stopifnot(identical(re123.blocks, re123.list.blocks))
```

---

eSetDimName *Get dimname from an eSet object*

Description

This function is implemented to automatically validate and choose feature (sample) names from the user input. This function is exported for the purpose of easing other Bioconductor developers performing the similar job, and is not tended to be called by end-user directly.

Usage

```
eSetDimName(eset, input, type = c("feature", "sample"))
```

Arguments

eset	An object of eSet class, mostly an ExpressionSet class.
input	The user input, see details below
type	Either 'feature' or 'sample', indicating which dimension should be determined

Details

The input can be one of the following three possibilities:

- Missing. Depending on the type, the results of calling featureNames ("feature") or sampleNames ("sample") on the eset object will be returned.
- A character string of length 1. Depending on the type, it is first to be machted to the column names of either fData or pData results of the eset object. If found, the values in that column are returned (coerced to characters if necessary). If not found, the function stops by raising an error.
- A character vector of the length equal to one of the two dimensions of the *eset*. In this scenario, the function only validates the equality of the length, coerces the input into characters, and return them.

If none of the scenarios above was met, the function stops by raising an error.

Value

A vector of characters, the length of which determined by the dimension of the input object.

Note

A special case arises if one of the dimensions of the eset object is 1: In this case, the input value is interpreted as the new name and returned. No column name match will take place in this case.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

sampleNames, featureNames, fData, pData

writeQubicInputFile calls the function.

Examples

```
data(sample.ExpressionSet, package="Biobase")
sub.eset <- sample.ExpressionSet[1:3, 1:3]

## usage one:
eSetDimName(sub.eset, type="feature")
eSetDimName(sub.eset, type="sample")

## usage two
```

```
## "sex" is one column in the pData(sub.eset)
eSetDimName(sub.eset, input="sex", type="sample")
## Not run: eSetDimName(sub.eset, input="foo", type="sample")

## usage three
eSetDimName(sub.eset, input=paste("Sample", 1:3), type="sample")
## Not run: eSetDimName(sub.eset, input=paste("Sample", 1:4),
type="sample")
## End(Not run)

## special case: dim equals to one
eSetDimName(sub.eset[,1], input="foo", type="sample")
```

`fcFilter` *Feature-Condition Filter*

Description

Filter Biclusters by feature and concition counts. Biclusters with fewer features or conditions than specified thresholds are removed.

Usage

```
fcFilter(object, ...)
```

Arguments

`object`	A `Biclust` object
`...`	Two parameters are accepted: `feat.min` and `cond.min`. They indicate the minimum number of features (conditions) in biclusters, which are to be accepted. It is allowed to only specify one threshold. If both are specified, only biclusters fulfilling both criterion are accepted.

Value

A `Biclust` object.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
data(sample.ExpressionSet, package="Biobase")
rqubic.example.discret <- quantileDiscretize(sample.ExpressionSet, rank=2L)

rqubic.example.sel.seeds <- generateSeeds(rqubic.example.discret, minColWidth=2L)

rqubic.example.blocks <- quBicluster(rqubic.example.sel.seeds,
    rqubic.example.discret,
    report.no=200L,
    filter.proportion=0.1)
```

```
print(rqubic.example.blocks)
print(fcFilter(rqubic.example.blocks,feat.min=10))
print(fcFilter(rqubic.example.blocks,cond.min=2))
print(fcFilter(rqubic.example.blocks,feat.min=10, cond.min=2))
```

---

fcFilter-methods *Feature-Condition Filter*

Description

Feature-Condition Filter for biclusters

Methods

`signature(object = "Biclust")` Use `help("fcFilter")` see help and examples

---

features-methods *Extract features and conditions*

Description

Generic function `features` and `conditions`, as well as auxillary count functions, are implemented for `Biclust` objects.

They can be used in one of the following forms:

1. Used on a `Biclust`, and without specifying index, `features` or `conditions` returns the unique and ordered features or conditions involved in at least one bicluster, and `featureCount` or `conditionCount` returns the length of repsective vectors. To get the feature/condition numbers in each bicluster of the set, use `BCfeatureCount`/`BCconditionCount`.
2. Used on a `Biclust` and provided index (indices), the features/conditions or their counts are returned for specified biclusters.

In addition, `featureNames` and `sampleNames` are of the same implementation as `features` and `conditions`.

Methods

`signature(object = "QUBICBicluster")` Information about all the biclusters.

`signature(object = "Biclust", index = "missing")` Information about all the biclusters in the set.

`signature(object = "Biclust", index = "ANY")` Information about selected biclusters in the set, the index can be integers or logical variables for subsetting.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

Guojun Li, Qin Ma, Haibao Tang, Andrew H. Paternson and Ying Xu (2009) *QUBIC: a qualitative biclustering algorithm for analyses of gene expression data*. Nucleic Acids Research, 37:e101

See Also

See other methods implemented for the Biclust class in the biclust package. And the methods specific for QUBICBiclusterSet.

Examples

```
library(Biobase)
library(biclust)
example.file <- system.file("extdata", "sampleExpressionSet.blocks", package="rqubic")

example.block <- parseQubicBlocks(example.file)

head(features(example.block))
featureCount(example.block)
head(conditions(example.block))
conditionCount(example.block)

BCfeatureCount(example.block)
BCfeatures(example.block)[1:2]
BCconditionCount(example.block)
BCconditions(example.block)[1:2]

head(featureNames(example.block))
head(sampleNames(example.block))
```

---

generateSeeds-methods *Generate seeds for biclustering*

Description

generateSeeds takes either matrix or an ExpressionSet object to generate seeds. Seeds are defined as pairs of genes (edges) which share coincident expression levels in samples. The higher the coincidence, the higher the score of the seeds will be. The seeds are generated by subsequent comparing each pair of genes. When all seeds have been produced, they are sorted by the coincidence scores and returned as an object. See the details section for notes on implementation.

Methods

In the rqubic package, generateSeeds currently supports two data types: ExpressionSet (an inherited type of eSet, or numeric matrix.

Both methods requires in addition a parameter, minColWidth, specifying the minimum number of conditions shared by the two genes of each seed. Its default value is 2. When this default value is used, the minimum coincidence score is defined as $max(2, ncol/20)$, where $ncol$ represents the number of conditions. When a non-default value is provided, the value is used to select seeds.

signature(object = "eSet") An object representing expression data. Note that the exprs must be a matrix of integers, otherwise the method warns and coerces the storage mode of matrix into integer.

signature(object = "matrix") A matrix of integers. In case filled by non-integers, the method warns and coerces the storage mode into integer

Details

The function compares all pairs of genes, namely all edges of a complete graph composed by genes. The weight of each edge is defined as the number of samples, in which two genes have the same expression level. This weight, also known as the *coincidence score*, reflects the co-regulation relationship between two genes.

The seed is chosen by picking edges with higher scores than the minimum score, provided by the `minColWidth` parameter (default: 2).

To implement such a selection algorithm, a *Fibonacci heap* is constructed in the C codes. Its size is predefined as a constant, which should be reduced in case the gene number is too large to run the algorithm. A new seed, which was selected by having a higher coincidence score than the minimum, is inserted to the heap. And dependent on whether the heap is full or not, it is either inserted by squeezing the minimum seed out, or put into the heap directly.

Once the heap is filled by examining all pairs of genes, it is dumped into an array of edge pointers, with decreasingly ordered edge pointers by their scores. This array is captured as an external pointer, attached as an attribute of an `rqubicSeeds` object.

An `rqubicSeeds` object holds an integer, which records the height of the heap. It has (besides the class identifier) two attributes: one for the external pointer, and the other one for the threshold of the coincidence score.

Note

In the `rqubic` implementation, the variable `arr_c[i][j]` holds the level symbols ($-1, 0, 1$ in the default case), whereas in the `QUBIC` implementation, this variable holds the index of level symbols, and the level symbols are saved in the global variable `symbols`.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
data(sample.ExpressionSet, package="Biobase")
sample.disc <- quantileDiscretize(sample.ExpressionSet)
sample.seeds <- generateSeeds(sample.disc)
sample.seeds

## with higher threshold of incidence score
sample.seeds.higher <- generateSeeds(sample.disc, minColWidth=5)
sample.seeds.higher
```

`parseQubicBlocks` *Parse QUBIC Command Line Tool Output Files*

Description

These functions parse output files of the QUBIC command line tool developed by Ma et al.

Usage

```
parseQubicRules(filename)
parseQubicChars(file, check.names=FALSE, ...)
parseQubicBlocks(filename)
```

Arguments

filename	Input filename
file	Input filename
check.names	logical, should the column names be checked?
...	other parameters passed to the read.csv function

Details

Parse QUBIC Command Line Tool Output Files

Value

parseQubicRules and parseQubicChars both return a data frame.

parseQubicBlocks returns an instance of QUBICBiclusterSet class.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

http://csbl.bmb.uga.edu/~maqin/bicluster/

Examples

```
getRqubicFile <- function(filename) system.file("extdata", filename, package="rqubic")

## parse QUBIC rules
rule.file <- getRqubicFile("sampleExpressionSet.rules")
rqubic.sample.rule <- parseQubicRules(rule.file)

## parse QUBIC chars
chars.file <- getRqubicFile("sampleExpressionSet.chars")
rqubic.sample.chars <- parseQubicChars(chars.file)

## parse QUBIC blocks
block.file <- getRqubicFile("sampleExpressionSet.blocks")
rqubic.sample.data <- parseQubicBlocks(block.file)
```

---

quantileDiscretize *Discretize expression matrix for qualitative biclustering*

Description

Performs recursive quantilizations on gene expression data across samples, to quantileDiscretize gene expression matrix. The quantile parameter q determines the estimated proportion of differentially expressed genes (*2q* as for both up- and down-regulatons). The rank parameter r determines how many discrete levels should differentially expressed genes (or outliers) have. See details below.

Usage

```
quantileDiscretize(x, ...)
```

Arguments

x	It can be an object of the eSet class or inheriting it. The most commonly used form is an linkS4class{ExpressionSet} class. Alternatively, it can be a numeric matrix.
...	Currently, the ... accepts two parameter: q and rank, explained below.

- qEstimated proportion of conditions where gene is up- or down-regulated, value between $(0, 0.5)$, default value is set to 0.06. By specifying q one estimates that in 2q of all conditions, the expression value of a gene is considered as outlier.
- rankRanks (levels) of outliers, a positive integer, default is 1L. By default, all conditions get one label for each gene in $-1, 0, 1$, representing down expression, not changing and high expression respectively. In case $rank > 1$, the outliers are further divided into *rank* levels by applying recursive quantilization with equal intervals.

Details

Parameter q corresponds to the command line option -q in the QUBIC command line tool, and the rank option corresponds to -r.

For each gene, the algorithm applies quantile discretization first to divide conditions into negative (lower), un-changed and positive (higher) expressions. Negative and positive expressed conditions are considered as *outliers*. For outliers in each direction, the algorithm tries to further quantileDiscretize the expression values in case $rank > 1$.

This second discretization step is performed by dividing the sorted outliers into $rank$ tandom groups with equal conditions. A label is assigned to each of these tandom groups, in the following order:

$$-1, -2, \ldots, -rank$$

for outliers with negative expression, from the *most negative group* to the *least negative group* (not the other way around!).

Similarly, for positive outliers, labels in the order of

$$rank, rank - 1, \ldots, 1$$

are assigned to tandom groups from the *least positive group* to the *most positive group*.

That is, signs of labels indicate the direction of gene expression change, and the absolute value represents the quantileDiscretized *rank* in the outliers.

Value

An object of the same class as the input parameter, with the exprs slot replaced by the quantileDiscretized matrix, which is a matrix of integer.

Note

Note that the resulting discrete matrix of this implementation can be slighly different from the one used by the QUBIC command line tool.

The main reason for this is the internal data type: while QUBIC uses float to represent expression matrix, we use double to represent the matrix.

It has the advantages of interfacing to R, having higher precision and avoiding errors caused by floating presentation. It is implemented with potential larger costs of memory, however for test data

sets (for example the ALL dataset with more than 120 samples and 12000 genes) the peak memory use (<100M) as well as the execution time (CPU time 0.028s) are well under control.

The differentially is especially often observed when there are many tied values. These cases however are very rare cases and we assume they will not affect the results to a large extent.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

Li et al. (2009) *QUBIC: a qualitative biclustering algorithm for analyses of gene expression data* Nucleic Acids Research 37:e101

See Also

`parseQubicChars` parses the quantileDiscretized matrix by the QUBIC command line tool into a data frame.

Examples

```
library(Biobase)
data(sample.ExpressionSet, package="Biobase")
sample.disc <- quantileDiscretize(sample.ExpressionSet)
exprs(sample.disc)[1:6, 1:6]

## Equivalent to pass a numeric matrix
sample.mat.disc <- quantileDiscretize(exprs(sample.ExpressionSet))
sample.mat.disc[1:6, 1:6]
## Not run: identical(exprs(sample.disc),sample.mat.disc)

## with multiple ranks
sample.rank3 <- quantileDiscretize(sample.ExpressionSet, rank=3)
exprs(sample.rank3)[1:6, 1:6]
```

QUBICBiclusterSet-class

Class "QUBICBiclusterSet"

Description

Object representing a set of biclusters identified by the QUBIC algorithm. The class structure inherits the `Biclust` class in the `biclust` package.

Objects from the Class

Created by functions parsing the output files of QUBIC command line tool, or functions calling QUBIC algorithm implementations in R.

Not intended to be created manually by end-users. However, interested users are invited to review the source code or use the `showClass` method to view the construction of the class.

Slots

See the class structure of `Biclust`. The slots `Parameter` and `Info` have been filled with lists releveant to the QUBIC algorithm, and all items should be accessed by S4-methods to make sure the consistency.

Methods

Svalue `signature(object = "QUBICBiclusterSet", index = "missing")`: Return S values of QUBIC biclusters as a vector

Svalue `signature(object = "QUBICBiclusterSet", index = "numeric")`: S values of specified bicluster(s) are returned. Index is one or a vector of integers. Non-integers will be coereced.

[`signature(x = "QUBICBiclusterSet", i = "ANY", j = "missing", drop = "missing")`: Returning a subset of the current QUBICBiclusterSet.

parameter `signature(object = "Biclust", index = "character")`: return an input parameter specified by the parameter name

parameter `signature(object = "Biclust", index = "missing")`: return a list of input parameters used by the biclustering algorithm, for example QUBIC

info `signature(object = "Biclust", index = "ANY")`: return information of the biclusters. For end-users, specific information accessors should be preferred, for example `features`, `conditions` and `Svalue`

info `signature(object = "Biclust", index = "missing")`: return all information of the biclusters in a list. For end-users, specific information accessors should be preferred, for example `features`, `conditions` and `Svalue`

show `signature(object = "QUBICBiclusterSet")`: showing method

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

Guojun Li, Qin Ma, Haibao Tang, Andrew H. Paternson and Ying Xu (2009) *QUBIC: a qualitative biclustering algorithm for analyses of gene expression data*. Nucleic Acids Research, 37:e101

See Also

`Biclust` is the basic block accomodating biclusters identified by the QUBIC algorithm.

Examples

```
showClass("QUBICBiclusterSet")
```

quBicluster *Qualitative Biclustering*

Description

The function takes seeds and quantileDiscretized ExpressionSet as input, biclusters the data and returns an object holding biclusters. Users may control the report number of clusters, tolerance of incoherent genes (or conditions), as well as the filtering of redundant clusters.

Usage

```
quBicluster(seeds, eset, report.no = 100L, tolerance = 0.95, filter.proportion = 1)
```

Arguments

seeds	An object of the S3-class rqubicSeeds, representing seeds generated from the quantileDiscretized expression data
eset	Discretized expression data
report.no	Number of biclusters that should be reported. Detected biclusters are ranked by the S-score, which is defined by the product of gene counts and sample counts. They are ordered and the top ones are reported.
tolerance	Percentage of tolerated incoherent samples, 0.95 by default
filter.proportion	Proportion of a cluster, over which the cluster is considered as redudant. Each bicluster is compared to all better ranking biclusters, and the overlapping proportion is measured by the proportion of the product of overlapping samples and overlapping genes, to the product samples and genes. If the proportion is larger than the given threshold, the block will be considered redundant and therefore not reported. Setting the threshold to 1 (default) does not perform any filtering.

Details

The function calls a C routine to perform the biclustering. Currently the routine returns blocks with fewer samples specified by the minimum column number, due to the set of tolerance values. This might be changed in the fewer versions.

Value

An object of the QUBICBiclusterSet-class, holding all biclusters.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

Li et al. (2009) *QUBIC: a qualitative biclustering algorithm for analyses of gene expression data* Nucleic Acids Research 37:e101

See Also

quantileDiscretize and generateSeeds

Examples

```
data(sample.ExpressionSet, package="Biobase")
rqubic.example.discret <- quantileDiscretize(sample.ExpressionSet, rank=2L)

rqubic.example.sel.seeds <- generateSeeds(rqubic.example.discret, minColWidth=2L)

rqubic.example.blocks <- quBicluster(rqubic.example.sel.seeds,
    rqubic.example.discret,
    report.no=200L,
    filter.proportion=0.1)

## print features in each bicluster
BCfeatures(rqubic.example.blocks)
```

---

readBiclusterResults *Import bicluster results from plain text files*

Description

This function complements the functionality of writeBiclusterResults in the biclust package. It constructs a Biclust object from a plain text file.

Usage

```
readBiclusterResults(filename, featureNames, sampleNames, delimiter = ";", ...)
```

Arguments

filename	Character, name of the file storing biclustering information
featureNames	Optional character vector, feature names of the underlying expression dataset. See details.
sampleNames	Optional character vector, sample names of the underlying expression dataset. See details.
delimiter	Character used to separate features, samples and counts of them.
...	Other parameters passed to the readLines function.

Details

Currently output files written by the writeBiclusterResults function does not contain original feature names or sample names in the expression dataset from which biclusters were mined. The featureNames and sampleNames allow to use this information to construct a Biclust object that has the same dimension as the original expression dataset.

Value

A Biclust object

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

In case several biclustering algorithms were applied to the same expression dataset, they can be combined with `combineBiclusts` once the results were read from plain texts.

Examples

```
temp <- tempfile()
library(biclust)
data(BicatYeast, package="biclust")
res <- biclust(BicatYeast, method=BCBimax(), number=5)
writeBiclusterResults(temp, res,"CC with delta 1.5",dimnames(BicatYeast)[[1]],dimnames(BicatYeast)[[2]], del
res.back <- readBiclusterResults(temp, delimiter=";")
res.full.back <- readBiclusterResults(temp,featureNames=rownames(BicatYeast), sampleNames=colnames(BicatYeas
```

---

writeQubicInputFile *Write an ExpressionSet object into the file format required by the QUBIC command line tool*

Description

The QUBIC commmand line tool (developed by Ma et al.) requires a tab-limited data matrix as input file, with some special requirements (see details below). This function takes an object of ExpressionSet and outputs the file.

Usage

```
writeQubicInputFile(x, file = "", featureNames, sampleNames)
```

Arguments

`x`	An object inheriting the `eSet` class, most commonly an `ExpressionSet` object, representing expression data of features across samples.
`file`	Filename to output, or a connection to write to (e.g. `stdout()`).
`featureNames`	Specifies the feature names. It can be left blank, in which case the result of calling `featureNames` on `x` will be used. Alternatively, it can be one character string, specifying which column in the `fData` should be used. The third possibility, it can be a vector of characters, with the same length as features in the object. In the last option, all other types will be converted to characters.
`sampleNames`	Specifies the sample names. It can be left blank, in which case the result of calling `sampleNames` on `x` will be used. Alternatively, it can be one character string, specifying which column in the `pData` should be used. The third possibility, it can be a vector of characters, with the same length as features in the object. In the last option, all other types will be converted to characters.

Details

The description of the data format can be checked by running the QUBIC tool in the command line mode, with the option *-h* (for *help*). A special requirement, which makes it different from the results of the `write.table` function in R, is that before the sample names (column names), an "o" must be added.

Value

No visible value will be returned, the function is called for its side effect.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`eSetDimName`, `write.table`

Examples

```
tmpfile <- tempfile()
data(sample.ExpressionSet, package="Biobase")
sub.eset <- sample.ExpressionSet[1:3, 1:3]

## write to standard output
writeQubicInputFile(sub.eset)

## write to a temporary file
writeQubicInputFile(sub.eset, tmpfile)
head(readLines(tmpfile))

## specify names with one column name in fData/pData
writeQubicInputFile(sub.eset, file="", sampleNames="sex")

## alternatively specifiy names manually
writeQubicInputFile(sub.eset, file="",sampleNames=paste("Sample", 1:3))
```

Index