Package 'GenomicTools'

March 9, 2020

Type Package

Title Collection of Tools for Genomic Data Analysis

Version 0.2.9.7

Date 2020-03-09

Author Daniel Fischer

Maintainer Daniel Fischer <daniel.fischer@luke.fi>

Depends R (>= 3.3), gMWT (>= 1.1), Rcpp (>= 0.9.13), data.table (>= 1.9.6), GenomicTools.fileHandler (>= 0.1.5.8)

Imports circlize, stringr, snpStats

Suggests knitr, rmarkdown

LinkingTo Rcpp, RcppArmadillo

Description A loose collection of tools for the analysis of expression and genotype data, currently with the main focus on (e)QTL and MDR analysis.

VignetteBuilder knitr

License GPL (>= 2)

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-03-09 06:30:02 UTC

R topics documented:

`GenomicTools-package` *Loose Collection of Expression and Genotype Analysis Tools, including (e)QTL and MDR.*

Description

GenomicTools is a loose collection of analysis tools for gene expression and genotype data. Currently it provides methods for eQTL, QTL and MDR analysis. It also contains an MDR ensemble classifier. This package is the continuation of the genomic data analysis parts of the GeneticTools package, that is also available on Cran.

Details

Package:	GenomicTools
Type:	Package
Version:	0.2.9.7
Date:	2020-03-09
License:	GPL
LazyLoad:	yes

Author(s)

Daniel Fischer

Maintainer: Daniel Fischer <daniel.fischer@luke.fi>

annotTrack *Example Annotation Track*

Description

An example for a typical annotation track.

Usage

```
data(annotTrack)
```

Format

A data table with 1000 rows, each representing one annotation with 11 columns as provided from standard gtf format from Ensembl.

Details

This is an example homo sapiens annotation track as it was downloaded from the Ensembl ftp download page. In total there are 1000 annotations from the human genome in release 85.

Source

http://www.ensembl.org/info/data/ftp/index.html

Examples

```
# The object was created from the downloaded Ensembl file as follows
## Not run:
  ensGTF <- importGTF(file="Homo_sapiens.GRCh38.85.gtf.gz")
  annotTrack <- ensGTF[1:1000,]
  save(annotTrack,file="annotTrack.rda")

## End(Not run)

data(annotTrack)
annotTrack
```

eQTL *Perform an eQTL Analysis*

Description

This function performs an eQTL analysis.

Usage

```
eQTL(gex=NULL, xAnnot = NULL, xSamples = NULL, geno=NULL, genoSamples = NULL,
    windowSize = 0.5, method = "directional", mc = 1, sig = NULL, which = NULL,
     testType = "asymptotic", nper = 2000, verbose = TRUE, MAF=0.05,
     IHaveSpace = FALSE)
```

Arguments

gex	Matrix or Vector with expression values.
xAnnot	Location annotations for the expression values.
xSamples	Sample names for the expression values (optional).
geno	Genotype data.
genoSamples	Sample names for the genotype values (optional).
windowSize	Size of the window around the center gene (in Mb).
method	Method of choice for the eQTL.
mc	Amount of cores for parallel computing.
sig	Significance level for the eQTL testing.
which	Names of genes for that the eQTL should be performed.
testType	Type of test, either permutation or asymptotic.
nper	Sets the amount of permutations, if permuation tests are used.
verbose	Logical, shall the method give extensive feedback.
MAF	Filter for Minor Allele Frequency.
IHaveSpace	Logical, security switch for operations that require lots of memory.

Details

This function performs an eQTL analysis and offers different types of tests. The type of test can be specified with the method option and possible options are "LM" and "directional". The option "LM" fits for each SNP within a predefined window of size windowSize (in MB) around a gene a linear model for the genotype information and the corresponding gene expression. The null hypothesis for each test is then that the slope is equal to zero and the alternative is that it is not zero.

The option "directional" applies a new directional test based on probabilistic indices for triples as described in Fischer, Oja, et al. (2014). Being $\mathbf{x}_0 = (x_{01}, x_{02}, \ldots, x_{0N_0})', \mathbf{x}_1 = (x_{11}, x_{12}, \ldots, x_{1N_1})'$ and $\mathbf{x}_2 = (x_{21}, x_{22}, \ldots, x_{2N_2})'$ the expression values that are linked to the three genotype groups $0, 1$ and 2 with underlying distributions F_0, F_1 and F_2. We first calculate the probabilistic indices $P_{0,1,2} = \frac{1}{N_0 N_1 N_2} \sum_i \sum_j \sum_k I(x_{0i} < x_{1j} < x_{2k})$ and $P_{2,1,0} = \frac{1}{N_0 N_1 N_2} \sum_i \sum_j \sum_k I(x_{2i} < x_{1j} < x_{0k})$. These are the probabilities that the expression values of the three groups follow a certain order, that is what we would expect for possible eQTLs. The null hypothesis that we have then in mind is that the expression values from these three groups have the same distribution $H_0 : F_0 = F_1 = F_2$ and the two alternatives are that the distributions have a certain stochastical order $H_1 : F_0 < F_1 < F_2$ and $H_2 : F_2 < F_1 < F_0$.

The test is applied for the two probabilistic indices $P_{0,1,2}$ and $P_{2,1,0}$ and combines the two resulting p-values $p_{012} = p_1$ and $p_{210} = p_2$ from previous tests then as overall p-value $\min(2 \min(p_1, p_2), 1)$.

In the two-group case (this means only two different genotypes are present for a certain SNP) a two-sided Wilcoxon rank-sum test is applied.

The gene expressions used in the eQTL are specified in `gex`. If several genes should be tested, then `gex` is a matrix and each column refers to a gene and each row to an individuum. The column names of this matrix must match then with the names used in the annotation object `xAnnot`. Sample names can either be given as row names in the matrix or as separate vector in `xSamples`. If only gene expressions of one gene should be tested then `gex` can be a vector.

The genotype information is provided in the `geno` object. Here one can i.a. specify the ped-file name of a ped/map file pair. The function then imports the genotype information using the function `importPED`. In that case, the map file has to have the same filename as the ped file (despite the file extension...). In case the genotype information has been imported already earlier using `importPED` the resulting `PedMap` object can also be given as a parameter for `geno`.

The `xAnnot` object carries the annotation information for the gene expressions. Usually, the annotations are downloaded e.g. from the Ensembl page and then imported to R with the `importGTF` function. The resulting object can then be used as input for `xAnnot`. Casting options to feed in also other formats/objects are currently under development.

The option `genoSamples` is used in case that the sample names in the ped/map file (or `geno` object) do not match with `rownames(gex)` given in the expression matrix. The vector `genoSamples` is as long as the `geno` object has samples, but gives then for each row in `geno` the corresponding name in the `gex` object. The function finds then also the smallest union between the two data objects. If there are repeated measurements per individual for the genotypes we take by default only the first appearance in the data and neglect all successive values. Currently this cannot be changed. In case this behavior is not desired, the user has to remove the corresponding rows from `geno` before starting the calculation.

If the code is executed on a Linux OS the user can specify with the `mc` option the amount of CPU cores used for the calculation.

If the `sig` option is set to a certain significance level, then the method only reports those SNPs that are tested to be significant. This can reduce the required memory drastically, especially in the case of trans-eQTL.

The method tests for trans-eQTLs (all combinations of SNPs and genes) if the `windowSize` is set to `NULL`. Be aware that this might lead to long lasting calculations. For trans-eQTL calculations it is advisable to define a significance level in `sig` so that only significant results are stored. If all results are required, the option `sig=NULL` has to be set and in addition `IHaveSpace=TRUE`. This additional parameter is necessary as a trans-eQTL with full output creates a huge output that often exceeds the system properties of a normal desktop PC.

Note: The directional test currently supports only exact p-values based on permutation tests, but asymptotic implementations are developed and will be soon available also.

Value

A list of class `eqtl` containing the values

`gex`	The `gex` object from the function call.
`geno`	The `geno` object from the function call.
`xAnnot`	The `xAnnot` object from the function call.
`genoSamples`	The `genoSamples` object from the function call.

`windowSize` The `windowSize` object from the function call.

and an incapsulated list `eqtl` where each list item is a tested gene location and contains the items

`ProbeLoc`	Used position of that gene. (Only different from 1 if multiple locations are considered.)
`TestedSNP`	Details about the considered SNPs.
`p.values`	P values of the test.
`GeneInfo`	Details about the center gene.

Author(s)

Daniel Fischer

References

Fischer, D., Oja, H., Sen, P.K., Schleutker, J., Wahlfors, T. (2014): Generalized Mann-Whitney Type Tests for Microarray Experiments, Scandinavian Journal of Statistics, 3, pages 672-692, doi: 10.1111/sjos.12055.

Fischer, D., Oja, H. (2013): Mann-Whitney Type Tests for Microarray Experiments: The R Package gMWT, submitted article.

Examples

```
## Not run:
# Make the example data available
data("annotTrack")   # Standard gtf file, imported with importGTF
data("geneEXP")      # Matrix with gene expression
data("genotData")    # A imported Ped/Map filepair, using importPED

# Transform gtf to bed format (not necessarily required)
annot.bed <- gtfToBed(annotTrack)

# cis-eQTL
################################################

# Most basic cis-eQTL runs:
EQTL1 <- eQTL(gex=geneEXP[,1:10] , xAnnot = annotTrack, geno= genotData)

# Same run, if gtf has been transformed to bed previously
EQTL1.1   <- eQTL(gex=geneEXP[,1:10] , xAnnot = annot.bed, geno= genotData)

# Same run, when the genotype data wasn't loaded and should be loaded
# here instead
EQTL1.2 <- eQTL(gex=geneEXP[,1:10] , xAnnot = annotTrack,
+               geno= file.path("Datasets","genotypes.ped"))

# Full set of genes, this time filtered with column names
EQTL2   <- eQTL(gex=geneEXP, xAnnot = annot.bed, geno= genotData,
+               which = colnames(geneEXP)[1:20])
```

```
# Single vector of gene expression values, underlying gene is specified
# in the which option
EQTL3   <- eQTL(gex=as.vector(geneEXP[,1]) , xAnnot = annot.bed,
+               geno= genotData, which="ENSG00000223972")

# Same call, but this time is the corresponding column not casted
EQTL3.1 <- eQTL(gex=geneEXP[,1] , xAnnot = annot.bed, geno= genotData,
+               which="ENSG00000223972")

# Same call, but instead of the name the row number in the gtf/bed
# file is provided
EQTL3.2 <- eQTL(gex=geneEXP[,1] , xAnnot = annot.bed, geno= genotData,
+               which=1)

# The same expression values are now assigned to three different genes
EQTL4   <- eQTL(gex=as.vector(geneEXP[,1]) , xAnnot = annot.bed,
+               geno= genotData, which=1:3)

# Trans-eQTL
#######################################

# Trans eQTL for the first and the last gene in our expression matrix
EQTL5   <- eQTL(gex=geneEXP[,c(1,1000)] , xAnnot = annot.bed,
+               geno= genotData, windowSize = NULL)

# Same call, this time distributed to 8 cores (ony available on
# Linux computers)
EQTL5   <- eQTL(gex=geneEXP[,c(1,1000)] , xAnnot = annot.bed,
+               geno= genotData, windowSize = NULL, mc=8)

# Expression values from the first gene are used to test the 100st
# gene for trans-eQTL
EQTL6   <- eQTL(gex=as.vector(geneEXP[,1]) , xAnnot = annot.bed,
+               geno= genotData, windowSize = NULL, which=100)

## End(Not run)
```

---

geneEXP *Simulated Expression Data*

Description

Simulated expression data.

Usage

```
data(geneEXP)
```

Format

A matrix with 50 rows representing individuals and 1000 genes arranged in the columns.

Details

Just simulated, numerical expression data for testing purposes. The genes are the same names as they can be found in the `annotTrack` object, the data consists out of 50 samples. The first 30 rows represent cases, the last 20 the controls. The expression data follows simply a N(0,1) distribution, however, the first 100 genes were shifted by +1.

Source

Own simulation, code can be given upon request.

Examples

```
data(Xgene)
```

`genotData` *Simulated Genotype Data*

Description

Simulated genotype data

Usage

```
data(genotData)
```

Format

A `PedMap` object as provided from `importPED`

Details

Small simulated genotype dataset that can be used for testing the method.

Source

Own simulation, code can be given upon request.

Examples

```
data(genotData)
```

getKEGGOrganisms *Get a list of organisms in the KEGG database*

Description

This function gets a list of organisms in the KEGG database.

Usage

```
getKEGGOrganisms(url="http://rest.kegg.jp/list/organism")
```

Arguments

url	Link to KEGG database.

Details

This function extracts a list of KEGG database organism names.

Value

A data.frame with organism names.

Author(s)

Daniel Fischer

getKEGGPathway *Get a list of genes in a specific KEGG pathway.*

Description

This function gets a list of organisms in the KEGG database.

Usage

```
getKEGGPathway(pathway)
```

Arguments

pathway	Pathway name

Details

This function delievers a vector with KEGG pathway information.

Value

A vector with pathways genes and entries. For a overview of all availabl pathway names in a specific organism use the getKEGGPathwayOverview function.

Author(s)

Daniel Fischer

Examples

```
  ## Not run:
  getKEGGPathway(pathway = "bta00620")

## End(Not run)
```

`getKEGGPathwayImage` *Get a visualization of a specific KEGG pathway.*

Description

This function gets a visualizatin of a specific KEGG pathway.

Usage

```
getKEGGPathwayImage(pathway, folder=NULL)
```

Arguments

`pathway`	Pathway name
`folder`	Path to where the image is stored

Details

thi function downloads a png figure of a specifi pathway and stores it in a given folder

Value

An image.

Author(s)

Daniel Fischer

Examples

```
 ## Not run:
  getKEGGPathwayImage(pathway = "bta00620")

## End(Not run)
```

getKEGGPathwayOverview *Get a list of pathways in a specific organism.*

Description

This function gets a list of pathways in a specific organism.

Usage

```
getKEGGPathwayOverview(code)
```

Arguments

code	Species code

Details

This function delievers a vector with KEGG pathway codes and names for a specific species. For a list of genes in a specific pathway, use the function getKEGGPathway.

Value

A vector with pathways IDs and names.

Author(s)

Daniel Fischer

Examples

```
## Not run:
getKEGGPathwayOverview(code = "bta")

## End(Not run)
```

getRSLocation *Get the location of an SNP.*

Description

This function gets the exact location of a SNPs, based on a specific database.

Usage

```
getRSLocation(rs, species=NULL)
```

Arguments

rs	SNP rs number
species	Species identifier

Details

This function can download from any specifi ensembl database the location of an SNP (as given as RS number) for a specific species.

Value

A list with Chr and Location (BP) of an SNP.

Author(s)

Daniel Fischer

Examples

```
# Depending on the internet connection the following call
# can take up 10 seconds and hence does not pass the Cran
# checks. This is why it is wrapped into \dontrun

  ## Not run:
    getRSLocation("rs16033432", species="Gallus_gallus")

## End(Not run)
```

---

gtfToBed *Extract the Chromosomal Information Required in bed Format from an imported gtf table.*

Description

This function creates a matrix of gene annotations in bed format, based on the information given in an importet gtf table.

Usage

```
gtfToBed(gtf, correctBases=TRUE)
```

Arguments

gtf	An imported gtf table.
correctBases	Logical subtract 1 from bases

Details

After a gtf file was imported with `importGTF` this function transforms it into a data.frame with bed formatting. The option `correctBases` adjusts the coordinates in the bed object according to the bed standard. The base counting in gtf starts with 1, wheras in bed-files they start with 0.

Value

A data.frame in bed format having the four columns `Chr`, `Start`, `Stop` and `Gene`

Author(s)

Daniel Fischer

Examples

```
## Not run:
annotTrack <- read.table(file="Homo_sapiens.GRCh37.70.gtf",sep="\t")
annotBed <- gtfToBed(annotTrack)

## End(Not run)
```

`mdr` *Perform a MDR.*

Description

This function performs a Multifactor Dimensionality Reduction (MDR).

Usage

```
mdr(X, status, fold=2, t=NULL, top=3, NAvalues=NA, cvc=0,
    fix=NULL, verbose=FALSE)
```

Arguments

`X`	Matrix with genotype information, see details.
`status`	Vector with group information of individuals in `X`.
`fold`	Maximum dimension of used contingency tables, see details.
`t`	Threshold for high/low risk.
`top`	Length of each top list.
`NAvalues`	Label of missing data.
`cvc`	Number of cross-validation splits.
`fix`	Column number of the SNP to be fixed.
`verbose`	Logical, if detailed status messages shall be given.

Details

The matrix `X` contains the genotype information, each column corresponds to a SNP, each row to an individuum. SNPs need to be coded as 0,1,2. In case the matrix `X` is not given in 0,1,2 format the function `recodeData` recodes the data into the required 0,1,2 format.

The `status` vector is as long as `X` has individuals/rows and specifies the group labels for each individual. Healthy individual need to be encoded as 0 and cases as 1. If the labeling is different the smaller values are used as controls and the larger one as cases.

The `fold` option specifies up to which dimension the contingency tables should be used. The current maximum is four.

The `t` option gives the threshold for the classification between high and low risk classes. The default is the ratio of the groups sizes.

With the `top` option the amount of results are set.

Setting the `fix` option to a column number, forces the mdr function to include that particular SNP into the result.

Missing data is labeled in different ways. The definition of missing data is given to the `NAvalues` option. By default missing data is encoded as NA, another possible option is e.g. 3. The downstream analysis ignors then missing data. Missing data is interally coded as 9999, so do not use this value to encode other genotypes.

The number of cross-validation splits can be set with the `cvc` option. If `cvc` is larger than 0, then the origianl data is split into `cvc`-many equally large subsets and the mdr function is called for each of them. Then, for each results from the top results of the full data is checked, in how many splits they also appear in the top result list.

Value

An object of class `mdr`.

Author(s)

Daniel Fischer

References

Moore JH, Gilbert JC, Tsai CT, Chiang FT, Holden T, Barney N, White BC. (2006): A flexible computational framework for detecting, characterizing, and interpreting statistical patterns of epistasis in genetic studies of human disease susceptibility. J Theor Biol.2006 Jul 21;241(2):252-61.

Examples

```
data(mdrExample)
mdrSNP <- mdrExample[,1:20]
fit.mdr <- mdr(mdrSNP, mdrExample$Class, fold=4, top=5)
fit.mdr
fit.mdr <- mdr(mdrSNP, mdrExample$Class)
fit.mdr
```

mdrEnsemble *Ensemble Classification Using an Existing MDR Object*

Description

This function takes an existing MDR object and a set of new observations and classifies them according to the best n sets of MDR classifiers.

Usage

```
mdrEnsemble(mdr, data=NULL, new.status=NULL, fold=NULL)
```

Arguments

mdr	The MDR object.
data	The new data.
new.status	Optional, the true status of the new data.
fold	Level of SNP-SNP interaction.

Details

This function takes an exisitng MDR object and a set of new observations and classifies them then using the top results of it, as it was defined in the previous MDR run.

Value

A classification of the new observations.

Author(s)

Daniel Fischer

Examples

```
## Not run:
data(mdrExample)
mdrSNP.train <- mdrExample[1:350,1:20]
mdrSNP.test <- mdrExample[351:400,1:20]
fit.mdr <- mdr(mdrSNP.train, mdrExample$Class[1:350], fold=2, top=20)
ensResult <- mdrEnsemble(fit.mdr, data = mdrSNP.test)

## End(Not run)
```

mdrExample *Example Data for MDR*

Description

This is one example data of the Java MDR program.

Usage

```
data(mdrExample)
```

Format

A data frame.

Details

This is an example dataset to show the use of the mdr function.

Source

https://sourceforge.net/projects/mdr/

Examples

```
data(mdrExample)
mdrExample[1:5,1:5]
```

phenoData *Simulated Phenotype Data*

Description

Simulated phenotype data.

Usage

```
data(phenoData)
```

Format

A matrix with 50 rows representing individuals and 10 phenotypes arranged in the columns.

Details

Just simulated, numerical phenotype data for testing purposes. The data consists out of 50 samples. The first 30 rows represent cases, the last 20 the controls. The phenotype data follows simply a N(0,1) distribution, however, the first four phenotypes were shifted by +1.

Source

Own simulation, code can be given upon request.

Examples

```
data(phenoData)
```

`plot.eqtl` *Plot an eqtl Object*

Description

The function offers informative plots for an `eqtl` object.

Usage

```
 ## S3 method for class 'eqtl'
plot(x, file = NULL, which = NULL, sig = 0.01, verbose = TRUE,
                  centered = TRUE, log = FALSE, mc.cores = 1, genome = NULL, ...)
```

Arguments

`x`	Object of class `eqtl`.
`file`	Store set of graphics under that file name.
`which`	Specifies for which genes should the plot be created.
`sig`	Chosen significance level.
`verbose`	Logical, extended feedback of the function.
`centered`	Logical, plot should be centered around center gene.
`log`	Logical, y-axis scale is log(base=10)-scaled.
`mc.cores`	Amount of cores for parallel computing.
`genome`	The name of an existing genome.
`...`	Additional plotting parameters.

Details

This function plots the test results of an `eqtl` object. Typically is the tested gene in the center and the p-values of associated SNPs are visualized. Monomorphic SNPs and those that were missing are separately plotted. Test results that are smaller than the value given to `sig` are marked in red. The y.axis can be switched to log10 scale by setting the logical parameter `log=TRUE` in that case are bars instead of dots plotted. If the y-axis is on log-scale it is also possible to give a second `eqtl` object to the function and plot the test results for both.

The annotation feature is currently under development and only available in limited form.

Author(s)

Daniel Fischer

Examples

```
# Load the data
  data("annotTrack")
  data("geneEXP")
  data("genotData")
## Not run:
# A cis eQTL for 10 different genes:
  EQTL1 <- eQTL(gex=geneEXP[,1:10] , xAnnot = annotTrack,
                geno= genotData)

# A trans-eQTl for two different genes:
  EQTL2   <- eQTL(gex=geneEXP[,c(1,1000)] , xAnnot = annot.bed,
                  geno= genotData, windowSize = NULL)

# Visualize the second cis-eQTL:
  plot(EQTL1, which=2)

# Visualize the trans-eQTL
  plot(EQTL2)

## End(Not run)
```

---

plot.mdr *Plot an MDR Object*

Description

The function offers informative plots for an `mdr` object.

Usage

```
 ## S3 method for class 'mdr'
plot(x, which=NULL, ...)
```

Arguments

`x`	Object of class `mdr`.
`which`	Specifies for which association fold should the plot be created.
`...`	Additional plotting parameters.

Details

This function plots the density of the precision of an `mdr` object.

Author(s)

Daniel Fischer

Examples

```
# Perform the MDR
# res <- mdr(X=temp,status=status,fold=3,top=20)

# plot(res)
```

`plot.qtlRes` *Plot an qtlRes Object*

Description

The function offers informative plots for an `qtlRes` object.

Usage

```
 ## S3 method for class 'qtlRes'
plot(x, which=NULL, sig=0.01, verbose=TRUE,
                       log=FALSE, genome=NULL, pch=1, ...)
```

Arguments

`x`	Object of class `qtlRes`.
`which`	Specifies for which phenotypes the plot should be created.
`sig`	Chosen significance level.
`verbose`	Logical, extended feedback of the function.
`log`	Logical, y-axis scale is log(base=10)-scaled.
`genome`	Data frame with Chromosome names and lengths.
`pch`	Plot symbol to by used.
`...`	Additional plotting parameters.

Details

This function plots the test results of an `qtlRes` object as a Manhattan plot. In case only an `qtlRes` object is provided, then the genome information are estimated from this object. Alternatively, a data frame with genome information can be provided. In this case genone has to be a data.frame with the two columns `chr` and `length`, indicating the Chromosome names (similar as the ones given in `qtlRes`) and the corresponding lengths. The information for the human genome, Ensembl build 68, is also included can be used with using the option `genome="Human68"`.

Author(s)

Daniel Fischer

Examples

```
## Not run:
data("phenoData")
data("genotData")

qtlresult <- QTL(pheno=phenoData, geno=genotData, which = 3:4)

plot(qtlresult, which=2)


## End(Not run)
```

---

print.eqtl *Print an eqtl Object*

Description

Prints an `eqtl` object.

Usage

```
 ## S3 method for class 'eqtl'
print(x, which=NULL, sig=0.01, output="bed", ...)
```

Arguments

`x`	Object of class `eqtl`.
`which`	Which center gene should be printed.
`sig`	Significance level.
`output`	Output format.
`...`	Additional parameters.

Details

The function prints SNPs in the surroundings of a gene from an `eqtl` object.

By default all genes are considered, subsets can be defined with the `which` option. The `sig` option gives the threshold which results should be shown.

Author(s)

Daniel Fischer

Examples

```
## Not run:
myeqtl <- eQTL(geneMatrix,genoData,singleLoc,genoSamples,singleSamples,windowSize,method="LM")
myeqtl
print(myeqtl, sig=0.05)

## End(Not run)
```

`print.mdr` *Print an mdr Object*

Description

Prints an `mdr` object.

Usage

```
 ## S3 method for class 'mdr'
print(x,...)
```

Arguments

`x`	Object of class `mdr`.
`...`	Additional parameters.

Details

The function prints an `mdr` object.

Author(s)

Daniel Fischer

Examples

```
# res <- mdr(X=temp,status=status,fold=3,top=20)
# res
```

QTL *Perform a QTL Analysis*

Description

This function performs a QTL analysis.

Usage

```
QTL(pheno, phenoSamples=NULL, geno=NULL, genoSamples=NULL,
    method="LM", mc=1, sig=NULL, testType="asymptotic",
    nper=2000, which=NULL, verbose=TRUE)
```

Arguments

pheno	Matrix or Vector with phenotype values.
phenoSamples	Sample names for the phenotype values, see details (optional).
geno	Genotype data.
genoSamples	Sample names for the genotype values, see details (optional).
method	Method of choice for the QTL, see details.
mc	Amount of cores for parallel computing.
sig	Significance level for the QTL testing, see details.
testType	Type of significance test, see details.
nper	Sets the amount of permutations, if permuation tests are used.
which	Names of phenotypes for that the QTL should be performed.
verbose	Logical, if the method should report intermediate results.

Details

This function performs a QTL analysis and offers different types of tests. The type of test can be specified with the method option and possible options are "LM" and "directional". The option "LM" fits for each provided SNP a linear model for the genotype information and the corresponding phenotype values. The null hypothesis for each test is then that the slope is equal to zero and the alternative is that it is not zero.

The "directional" option applies a new directional test based on probabilistic indices for triples as described in Fischer, Oja, et al. (2014). Being $\mathbf{x}_0 = (x_{01}, x_{02}, \ldots, x_{0N_0})', \mathbf{x}_1 = (x_{11}, x_{12}, \ldots, x_{1N_1})'$ and $\mathbf{x}_2 = (x_{21}, x_{22}, \ldots, x_{2N_2})'$ the phenotype values that are linked to the three genotype groups $0, 1$ and 2 with underlying distributions F_0, F_1 and F_2. We first calculate the probabilistic indices $P_{0,1,2} = \frac{1}{N_0 N_1 N_2} \sum_i \sum_j \sum_k I(x_{0i} < x_{1j} < x_{2k})$ and $P_{2,1,0} = \frac{1}{N_0 N_1 N_2} \sum_i \sum_j \sum_k I(x_{2i} < x_{1j} < x_{0k})$. These are the probabilities that the phenotype values of the three groups follow a certain order what we would expect for possible QTLs. The null hypothesis that we have then in mind is that the phenotype values from these three groups have the same distribution $H_0 : F_0 = F_1 = F_2$ and the two alternatives are that the distributions have a certain stochastical order $H_1 : F_0 < F_1 < F_2$ and $H_2 : F_2 < F_1 < F_0$.

The test is applied for the two probabilistic indices $P_{0,1,2}$ and $P_{2,1,0}$ and combines the two resulting p-values $p_{012} = p_1$ and $p_{210} = p_2$ from previous tests then as overall p-value $\min(2\min(p_1, p_2), 1)$. In the two-group case (this means only two different genotypes are present for a certain SNP) a two-sided Wilcoxon rank-sum test is applied.

The phenotype values used in the QTL are specified in `pheno`. If several phenotypes should be tested, then `pheno` is a matrix and each column refers to a phenotype and each row to an individuum. Sample names can either be given as row names in the matrix or as separate vector in `phenoSamples`. If only values of one phenotype should be tested then `pheno` can be a vector.

The genotype information is provided in the `geno` object. Here one can i.a. specify the file name of a ped/map file pair. The function then imports the genotype information using the function `importPED`. In case the genotype information has been imported already earlier using `importPED` the resulting `PedMap` object can also be given as a parameter for `geno`.

The option `genoSamples` is used in case that the sample names in the ped/map file (or SnpMatrix) do not match with `rownames(pheno)` given in the expression matrix. The vector `genoSamples` is as long as the `geno` object has samples, but gives then for each row in `geno` the corresponding name in the `pheno` object. The function finds then also the smallest union between the two data objects. If there are repeated measurements per individual for the genotypes we take by default only the first appearance in the data and neglect all successive values. Currently this cannot be changed. In case this behavior is not desired, the user has to remove the corresponding rows from `geno` before starting the calculation.

If the code is executed on a Linux OS the user can specify with the `mc` option the amount of CPU cores used for the calculation.

If the `sig` option is set to a certain significance level, then the method only reports those SNPs that are tested to be significant. This can reduce the required memory drastically, but shouldn't be used in case the results will be later plotted as Manhattan plot.

Value

A list of class `qtl` containing the values

`pheno`	The `pheno` object from the function call.
`geno`	The `geno` object from the function call.
`genoSamples`	The `genoSamples` object from the function call.
`windowSize`	The `windowSize` object from the function call.

and an incapsulated list `qtl` where each list item is a tested phenotype and contains the items

`ProbeLoc`	Used position of that gene. (Only different from 1 if multiple locations are considered.)
`TestedSNP`	Details about the considered SNPs.
`p.values`	P values of the test.
`GeneInfo`	Details about the center gene.

Author(s)

Daniel Fischer

References

Fischer, D., Oja, H., Sen, P.K., Schleutker, J., Wahlfors, T. (2014): Generalized Mann-Whitney Type Tests for Microarray Experiments, Scandinavian Journal of Statistics, 3, pages 672-692, doi: 10.1111/sjos.12055.

Fischer, D., Oja, H. (2013): Mann-Whitney Type Tests for Microarray Experiments: The R Package gMWT, submitted article.

`recodeData` *Recode a Genotype Data Matrix.*

Description

This function recodes a genotype data matrix in a format as expected from `eqtl` or `mdr`.

Usage

```
recodeData(X)
```

Arguments

`X`	Matrix with genotype information.

Details

This function recodes the values given in the data matrix `X` (typically `AA`, `AB` and `BB`) and substitutes it with 0,1 and 2. Missing values are encoded as 3.

Value

A matrix with same dimension as the input, but in 0,1,2 encoding.

Author(s)

Daniel Fischer

See Also

`eQTL` , `mdr`

Examples

```
# genotData <- read.table("MDR_format_ready_BCR.txt",header=T)

# temp <- recodeData(genotData)
```

seqlength *Get the sequence length of a fa-object.*

Description

This function determines the length of a fa-object.

Usage

```
seqlength(x)
```

Arguments

x	The fa-object

Details

This is basically a wrapper around the nchar-function

Value

An vector containing the sequence lengths of an fa-object.

Author(s)

Daniel Fischer

summary.eqtl *Summarize an eqtl Object*

Description

Summarizes and prints an eqtl object in an informative way.

Usage

```
 ## S3 method for class 'eqtl'
summary(object, sig=0.01, ...)
```

Arguments

object	Object of class eqtl.
sig	Significance level to print.
...	Additional parameters.

Details

This function gives a summary of an `eqtl` object.

Author(s)

Daniel Fischer

Examples

```
# First perform an eQTL
# lm.myEQTL <- eQTL(gex=ourExpression, geno=genotData, xAnnot=xAnnotDF, windowSize=1,
#                   genoSamples=genoSamples, method="LM")

# summary(lm.myEQTL)
```

Index