Package 'GRIN2'

October 29, 2024

Title Genomic Random Interval (GRIN)

Version 1.0

Description The GRIN2 package is an improved version of GRIN software that streamlines its use in practice to analyze genomic lesion data, accelerate its computing, and expand its analysis capabilities to answer additional scientific questions including a rigorous evaluation of the association of genomic lesions with RNA expression.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports biomaRt, circlize, ComplexHeatmap, data.table, dplyr, EnsDb.Hsapiens.v75, ensembldb, forcats, GenomeInfoDb, ggplot2, graphics, grDevices, grid, gridGraphics, Gviz, magrittr, stats, stringr, survival, tibble, tidyselect, utils, writexl

Suggests knitr, rmarkdown

NeedsCompilation no

Author Abdelrahman Elsayed [aut, cre],
Stanley Pounds [aut]

Maintainer Abdelrahman Elsayed <aelsayed@stjude.org>

Depends R (>= 4.2.0)

LazyData true

VignetteBuilder knitr

Contents

`alex.boxplots` *Prepare Box Plots of Expression Data by Lesion Groups*

Description

Function return box plots for expression data by lesion groups for selected number of genes based on a specified q-value of the kruskal-wallis test results.

Usage

```
alex.boxplots(alex.data, alex.kw.results, q, gene.annotation)
```

Arguments

`alex.data`	Output of the alex.prep.lsn.expr function. It's a list of three data tables that include "row.mtch", "alex.expr" with expression data, "alex.lsn" with lesion data. Rows of alex.expr, and "alex.lsn" matrices are ordered by gene ensembl IDs and the columns are ordered by patient ID.
`alex.kw.results`	ALEX Kruskal-Wallis test results (output of the KW.hit.express function).

`q`	minimum q value for a gene to be included in output PDF file of box plots.
`gene.annotation`	Gene annotation data either provided by the user or retrieved from ensembl BioMart database using get.ensembl.annotation function included in the GRIN2.0 library. Data.frame should has four columns: "gene" which is the ensembl ID of annotated genes, "chrom" which is the chromosome on which the gene is located, "loc.start" which is the gene start position, and "loc.end" the gene end position.

Value

Function return a PDF file with box plots for expression data by lesion groups for selected number of genes based on a specified q-value of the kruskal-wallis test results (one gene per page).

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023) Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`alex.prep.lsn.expr()`, `KW.hit.express()`

Examples

```
## Not run:
data(expr.data)
data(lesion.data)
data(hg19.gene.annotation)

# prepare expression, lesion data and return the set of genes with both types of data available
# ordered by gene IDs in rows and patient IDs in columns:
alex.data=alex.prep.lsn.expr(expr.data, lesion.data,
                             hg19.gene.annotation, min.pts.expr=5, min.pts.lsn=5)

# run KW test for association between lesion groups and expression level of the same gene:
alex.kw.results=KW.hit.express(alex.data, hg19.gene.annotation, min.grp.size=5)

# open a PDF to add boxplots, one gene per page
pdf("ALEX-boxplots-KW-q0.000001.pdf",width = 8,height = 5, onefile = TRUE)
alex.boxplots(alex.data, alex.kw.results, 0.00001, hg19.gene.annotation)
dev.off()

## End(Not run)
```

`alex.pathway` *Associate Lesions with Expression Data on the Pathway Level*

Description

Function compute the distance between subjects in the dataset based on the lesions that affect different genes assigned to the pathway of interest and return two panels of lesion and expression data of ordered subjects based on the computed distances.

Usage

```
alex.pathway(alex.data, lsn.data, pathways, selected.pathway)
```

Arguments

`alex.data`	output of the alex.prep.lsn.expr function. It's a list of three data tables that include "row.mtch", "alex.expr" with expression data, "alex.lsn" with lesion data. Rows of alex.expr, and "alex.lsn" matrices are ordered by gene ensembl IDs and columns are ordered by patient ID.
`lsn.data`	Lesion data in a GRIN compatible format. data.frame should has five columns that include "ID" with patient ID, "chrom" which is the chromosome on which the lesion is located, "loc.start" which is the lesion start position, "loc.end" the lesion end position and "lsn.type" which is the lesion type for example gain, loss, mutation, fusion, etc...
`pathways`	data.frame with three columns "gene.name" that has gene symbols, "ensembl.id" with gene ensembl ID and "pathway" that has the pathway name.
`selected.pathway`	The pathway of interest.

Details

Function compute the distance between subjects in th dataset based on lesions affecting different genes assigned to the pathway of interest and return two panels of lesion and expression data of ordered subjects based on the computed distances. Function also return a data.frame with lesion and expression data of the pathway genes ordered based on the hierarchical clustering analysis (same order of the subjects in the lesion and expression panels of the figure).

Value

Function will return two panels figure of lesion and expression data of ordered subjects based on the computed distances of lesions in all genes assigned to the pathway of interest. The function will also return:

`ordered.path.data`	data.frame with lesion and expression data of the pathway genes ordered based on the hiearchial clustering analysis (same order of the subjects in the lesion and expression panels of the figure).

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

See Also

alex.prep.lsn.expr(), hclust()

Examples

```
## Not run:
data(expr.data)
data(lesion.data)
data(hg19.gene.annotation)
data(pathways)

# prepare expression, lesion data and return the set of genes with both types of data available
# ordered by gene IDs in rows and patient IDs in columns:
alex.data=alex.prep.lsn.expr(expr.data, lesion.data,
                             hg19.gene.annotation, min.pts.expr=5, min.pts.lsn=5)

# use lesions in all genes assigned to the jak_pathway as an example pathway:
alex.path=alex.pathway(alex.data, lesion.data, pathways, "Jak_Pathway")

## End(Not run)
```

---

alex.prep.lsn.expr *Prepare Lesion and Expression Data for Kruskal-Wallis Test*

Description

The function prepares lesion and expression data matrices for the KW.hit.express function that runs the kruskal-Wallis test for the association between lesion groups and expression level of each gene with available lesion and expression data.

Usage

```
alex.prep.lsn.expr(
  expr.mtx,
  lsn.data,
  gene.annotation,
  min.expr = NULL,
  min.pts.lsn = NULL
)
```

Arguments

expr.mtx	Normalized log2 transformed expression data provided by the user with genes in rows and subjects in columns (first column "ensembl.ID" should be gene ensembl IDs).
lsn.data	Lesion data in GRIN compatible format. Data frame should has five columns that include "ID" with patient ID, "chrom" which is the chromosome on which the lesion is located, "loc.start" which is the lesion start position, "loc.end" the lesion end position and "lsn.type" which is the lesion type for example gain, loss, mutation, fusion, etc...

`gene.annotation`	Gene annotation data either provided by the user or retrieved from ensembl BioMart database using get.ensembl.annotation function included in the GRIN2.0 library. Data.frame should has four columns: "gene" which is the ensembl ID of annotated genes, "chrom" which is the chromosome on which the gene is located, "loc.start" which is the gene start position, and "loc.end" the gene end position.
`min.expr`	Minimum allowed expression level of the gene (the sum of expression level of the gene in all patients; useful to exclude genes with very low expression)
`min.pts.lsn`	Minimum number of patients with any type of lesions in a certain gene otherwise the gene will be excluded from the lesion matrix.

Details

The function use prep.lsn.type.matrix function to prepare the lesion matrix that has each gene represented in one row with all lesion types included. Next, the function will prepare lesion and expression data matrices for the KW.hit.express function that runs the kruskal-Wallis test. It only keep genes with both lesion and expression data with rows ordered by ensembl ID and columns ordered by patient's ID.

Value

A list with the following components:

`alex.expr`	Expression data with gene ensembl IDs as row names and patient IDs as column names. Rows are ordered by ensembl ID and columns ordered by patient IDs.
`alex.lsn`	Lesion data for genes in the expression data matrix with gene ensembl IDs as row names and patient IDs as column names. Rows are ordered by ensembl ID and columns ordered by patient IDs.
`alex.row.mtch`	Data.frame of two columns with ensembl ID of genes in the expression and lesion data matrices (ID should be the same in the two columns).

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`KW.hit.express()`

Examples

```
## Not run:
data(expr.data)
data(lesion.data)
data(hg19.gene.annotation)

# prepare expression, lesion data and return the set of genes with both types of data available
# ordered by gene IDs in rows and patient IDs in columns:
```

```
alex.data=alex.prep.lsn.expr(expr.data, lesion.data,
                             hg19.gene.annotation, min.expr=1, min.pts.lsn=5)

## End(Not run)
```

`alex.waterfall.plot` *Prepare Waterfall Plot of Lesion and Expression Data*

Description

Function return a waterfall plot for expression data by lesion groups of a selected gene.

Usage

```
alex.waterfall.plot(waterfall.prep, lsn.data, lsn.clrs = NULL, delta = 0.5)
```

Arguments

`waterfall.prep`	Output of the alex.waterfall.prep function. It's a list of three data tables that include "gene.lsn.exp" that has patient ID, lesion type that affect this gene if any and expression level of the selected gene, "lsns" which is a data table with all lesions affecting the gene of interest in a GRIN compatible format and "stats" which is one row with the Kruskal-Wallis test result (output of the KW.hit.express function).
`lsn.data`	Lesion data in a GRIN compatible format.
`lsn.clrs`	Assigned colors per lesion types. If not specified, colors will be automatically assigned using default.grin.colors function.
`delta`	Spacing argument for the waterfall plot.

Details

Function return a waterfall plot for expression data by lesion groups of a selected gene. This plot offers a side by side graphical representation of lesion and expression data for each patient where lesion groups are ordered alphabetically. For each lesion category, expression data is ordered from the lowest to the highest with patient with the median expression of the gene in the middle of the panel.

Value

Function return a waterfall plot for expression data by lesion groups of a selected gene.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

alex.prep.lsn.expr(), KW.hit.express(), alex.waterfall.prep()

Examples

```
## Not run:
data(expr.data)
data(lesion.data)
data(hg19.gene.annotation)

# prepare expression, lesion data and return the set of genes with both types of data available
# ordered by gene IDs in rows and patient IDs in columns:
alex.data=alex.prep.lsn.expr(expr.data, lesion.data,
                             hg19.gene.annotation, min.expr=1, min.pts.lsn=5)

# run KW test for association between lesion groups and expression level of the same gene:
alex.kw.results=KW.hit.express(alex.data, hg19.gene.annotation, min.grp.size=5)

# To prepare lesion and expression data for a waterfall plot (WT1 gene):
WT1.waterfall.prep=alex.waterfall.prep(alex.data, alex.kw.results, "WT1", lesion.data)

# waterfall plot of WT1 gene:
WT1.waterfall.plot=alex.waterfall.plot(WT1.waterfall.prep, lesion.data)

## End(Not run)
```

---

alex.waterfall.prep *Prepare Lesion and Expression Data for Waterfall Plots*

Description

Function prepares lesion and expression data of a selected gene for the alex.waterfall.plot function.

Usage

```
alex.waterfall.prep(alex.data, alex.kw.results, gene, lsn.data)
```

Arguments

alex.data	output of the alex.prep.lsn.expr function. It's a list of three data tables that include "row.mtch", "alex.expr" with expression data, "alex.lsn" with lesion data. Rows of alex.expr, and "alex.lsn" matrices are ordered by the gene ensembl IDs and columns are ordered by patient IDs.
alex.kw.results	ALEX Kruskal-Wallis test results (output of the KW.hit.express function).
gene	Gene name or ensembl ID of the gene of interest.
lsn.data	Lesion data in a GRIN compatible format. Object should has five columns that include "ID" with patient ID, "chrom" which is the chromosome on which the lesion is located, "loc.start" which is the lesion start position, "loc.end" the lesion end position and "lsn.type" which is the lesion category for example gain, loss, mutation, fusion, etc...

Details

Function prepares lesion and expression data of a selected gene for the alex.waterfall.plot function. It return a data table with patient ID, lesion types that affect each patient if any and expression level of the gene of interest. It also extract the kruskal-wallis test result and all lesions that affect the gene of interest.

Value

A list of four components:

`gene.lsn.exp`	Data table with three columns("ID" with patient ID, "gene.name_lsn" has the type of lesion affecting the patient which can be none, gain, mutation, multiple, etc.. and "gene.name_expr" which has the expression level of the gene of interst in this particular patient.
`lsns`	Data table with all lesions affecting the gene of interst in a GRIN compatible format extracted from the lesion data file.
`stats`	One row with the Kruskal-Wallis test result for the gene of interst (output of the KW.hit.express function).
`gene.ID`	Gene name of the gene of interst.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`alex.prep.lsn.expr()`, `KW.hit.express()`

Examples

```
## Not run:
data(expr.data)
data(lesion.data)
data(hg19.gene.annotation)

# prepare expression, lesion data and return the set of genes with both types of data available
# ordered by gene IDs in rows and patient IDs in columns:
alex.data=alex.prep.lsn.expr(expr.data, lesion.data,
                             hg19.gene.annotation, min.expr=1, min.pts.lsn=5)

# run KW test for association between lesion groups and expression level of the same gene:
alex.kw.results=KW.hit.express(alex.data, hg19.gene.annotation, min.grp.size=5)

# To prepare lesion and expression data for a waterfall plot (WT1 gene):
WT1.waterfall.prep=alex.waterfall.prep(alex.data, alex.kw.results, "WT1", lesion.data)

## End(Not run)
```

chrom.lsn.plot *Chromosome Lesion Plot*

Description

Function plot lesion data of a selected lesion type or all lesion groups located on a certain region such as a chromosome band or the whole specified chromosome .

Usage

```
chrom.lsn.plot(
  grin.res,
  genome,
  lsn.clrs = NULL,
  chrom = NULL,
  plot.start = NULL,
  plot.end = NULL,
  lesion.grp = NULL,
  spec.lsn.clr = NULL,
  expand = 5e-04,
  hg38.transcripts = NULL,
  hg19.cytoband = NULL,
  hg38.cytoband = NULL
)
```

Arguments

`grin.res`	GRIN results (output of the grin.stats function).
`genome`	Function support either "hg19" or "hg38" genome assemblies based on the genome assembly that has been used to prepare the lesion data.
`lsn.clrs`	Lesion colors for the regional gene plot (If not provided by the user, colors will be automatically assigned using default.grin.colors function).
`chrom`	chromosome number
`plot.start`	Start position of the locus of interest.
`plot.end`	End position of the locus of interest.
`lesion.grp`	Lesion type of interest (if not specified the plot will return all lesion groups)
`spec.lsn.clr`	Assigned color for the lesion type of interest (should be specified when lesion.grp is specified).
`expand`	Controls ratio of the specified locus (start and end position) to the whole plot with default value = 0.0005 (setting expand=0 will only plot the specified locus from the start to the end position without any of the upstream or downstream regions.
`hg38.transcripts`	transcripts data retrieved from annotation hub for hg38 version 110 (should be only specified if genome="hg38").
`hg19.cytoband`	hg19 chromosome bands start and end data in base pair (should be only specified if genome="hg19").
`hg38.cytoband`	hg38 chromosome bands start and end data in base pair (should be only specified if genome="hg38").

Details

Function will return a plot with either lesions of a certain lesion type specified by the user or all lesions of different lesion groups allocated to either a specific part of the chromosome such as a chromosome band or the whole length of the chromosme.As compared the the grin.gene.plot function, chrom.lsn.plot will not return a transcripts track which will allow the user to specify a large region or even the whole chromosome. A chromosome ideogram is added on the top of the lesion plot with the specified part of the chromosome encircled in red.

Value

Function will return a plot of a pre-specfied lesion type or all lesion affecting a specific locus or the whole chromosome.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`grin.stats()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)
data(hg19.chrom.size)
data(hg19_cytoband)
data(hg38_cytoband)

# run GRIN analysis using grin.stats function
grin.results=grin.stats(lesion.data,
                        genome.version="Human_GRCh37")

# Plots Showing Different Types of Lesions Affecting a region of Interest without plotting the
# transcripts track (this will allow plotting a larger locus of the chromosome such as a
# chromosome band using hg19 genome assembly):
locus.plot=chrom.lsn.plot(grin.results, genome="hg19", hg19.cytoband=hg19_cytoband,
                          chrom=9, plot.start=19900000, plot.end=25600000,
                          lesion.grp = "loss", spec.lsn.clr = "blue")

# Plots Showing Different Types of Lesions Affecting the whole chromosome:
chrom.lsn=chrom.lsn.plot(GRIN.results, genome="hg19", hg19.cytoband=hg19_cytoband,
                          chrom=9, plot.start=1, plot.end=141000000)

# for hg38 genome assembly:
ah <- AnnotationHub()
gtf.V110 <- ah[["AH113665"]]

# run GRIN analysis using grin.stats function
```

```
grin.results=grin.stats(lesion.data,
                        genome.version="Human_GRCh38")
# to plot deletions on the p21.3 band of chromosome 9:
chrom.lsn.plot(grin.results, genome="hg38", hg38.transcripts="gtf.v110",
           hg38.cytoband=hg38_cytoband, chrom=9, plot.start=19900000, plot.end=25600000,
               lesion.grp = "loss", spec.lsn.clr = "blue")

# to plot all lesions on chromosome 9:
chrom.lsn.plot(grin.results, genome="hg38", hg38.transcripts="gtf.v110",
               hg38.cytoband=hg38_cytoband, chrom=9, plot.start=1, plot.end=141000000)

## End(Not run)
```

`clin.data` *Example T-ALL Dataset Clinical Data*

Description

Clinical data file showing demographic and clinical outcomes of 265 newly diagnosed T-cell Acute Lymphoblastic Leukemia (T-ALL) patients that was reported by Liu, Yu, et al. (2017).

Usage

```
clin.data
```

Format

`clin.data`**:**

A data frame with 265 rows and 11 columns:

ID Patient identifier

Sex Patient gender

Race Patient race

Age_Days Patient age in days

WBC White Blood Cell (WBC) count

MRD29 Minimal Residual Disease (MRD) percentage

MRD.binary MRD as a categorical variable (0 if MRD<=0.1 or 1 if MRD>0.1)

os.time Overall survival time in years (time between diagnosis and either the last follow-up or death)

os.censor Survival status (0=alive at the last follow-up or, 1=dead)

efs.time Event-free survival time in years

efs.censor Event indicator (0=censored without event or, 1=event)

Source

Data was extracted from the supplementary material tables of the published Liu, Yu, et al. (2017) manuscript https://www.nature.com/articles/ng.3909#Sec27 and the publicly available clinical data on TARGET database. The two files were merged and selected list of variables were kept in the final clinical data file.

`compute.gw.coordinates`
Compute Genome-wide Coordinates

Description

The function assign plotting coordinates necessary for the genome-wide lesion plot.

Usage

```
## S3 method for class 'gw.coordinates'
compute(grin.res, scl = 1e+06)
```

Arguments

`grin.res`	GRIN results (output of the grin.stats function).
`scl`	length of chromosome units in base pairs. Default is 1,000,000 which means that each chromosome will be divided into multiple pieces each is 1 million base pair in length.

Details

The function divides each chromosome into multiple units based on the specified scl value. In addition, it orders and adds two columns x.start and x.end to the chromosme size file (x.start for chr2 is equal to x.end of chr1). Function also adds x.start and x.end columns to lesion and gene annotation data files (x.start is the start position of the lesion or the gene divided by scl and x.end is the end position of the lesion or the gene divided by scl taking into consideration that the start position of the chromosomes is added consecutively based on the chromosomes length).

Value

Function return a list of GRIN results with the following changes to allow adding genome-wide plotting coordinates:

`gene.hits`	No changes, a data table of GRIN results that includes gene annotation, number of subjects and number of hits affecting each locus, p and FDR adjusted q-values showing the probability of each locus to be affected by one or a constellation of multiple types of lesions.
`gene.lsn.data`	No changes, each row represent a gene overlapped by a certain lesion. Column "gene" shows the overlapped gene ensembl ID, and ID column has the patient ID
`lsn.data`	input lesion data with two additional columns (x.start and x.end). x.start is the start position of the lesion divided by scl and x.end is the end position of the lesion divided by scl taking into consideration that the start position of the chromosomes is added consecutively based on the chromosomes length.
`gene.data`	input gene annotation data with two additional columns (x.start and x.end). x.start is the start position of the gene divided by scl and x.end is the end position of the gene divided by scl taking into consideration that the start position of the chromosomes is added consecutively based on the chromosomes length.

`chr.size`	data table showing the size of the 22 autosomes, in addition to X and Y chromosomes in base pairs with two additional columns (x.start and x.end). x.start is the start position of the chromosome divided by scl and x.end is the end position of the chromosome divided by scl taking into consideration that the start position of the chromosomes is added consecutively based on the chromosomes length.
`gene.index`	data.frame with overlapped gene-lesion data rows that belong to each chromosome in the gene.lsn.data table.
`lsn.index`	data.frame that shows the overlapped gene-lesion data rows taht belong to each lesion in the gene.lsn.data table.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`grin.stats()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)
data(hg19.chrom.size)

# Run GRIN model using grin.stats function
grin.results=grin.stats(lesion.data,
                        hg19.gene.annotation,
                         hg19.chrom.size)
# assign genomewide coordinates and prepare the results for the genomewide.lsn.plot function
genome.coord=compute.gw.coordinates(grin.results)

## End(Not run)
```

`count.hits` *Count Gene Lesion Hits*

Description

The function computes the number of hits affecting each gene by lesion category. It also compute the number of subjects with a hit in each annotated gene by lesion category as well.

Usage

```
## S3 method for class 'hits'
count(ov.data)
```

Arguments

`ov.data`	a list of six data.frames that represent the output results of the find.gene.lsn.overlaps function.

Details

The function use the output of the find.gene.lsn.overlaps function and return the number of unique subjects affected by each lesion category in the provided list of annotated genes and regulatory features (nsubj stats). It also count the number of hits affecting each loci per lesion category (nhits stats). For example, if NOTCH1 gene was found affected by three different mutations in the same subject, this patient will be considered as one subject in the nsubj stats but in the nhits stats for this event will be counted as 3 mutations that affect NOTCH1 gene.

Value

A list with the following components:

`lsn.data`	Input lesion data
`lsn.index`	data.frame that shows the overlapped gene-lesion data rows taht belong to each lesion in the gene.lsn.data table.
`gene.data`	Input gene annotation data
`gene.index`	data.frame with overlapped gene-lesion data rows that belong to each chromosome in the gene.lsn.data table.
`nhit.mtx`	A data.frame with number of hits in each gene by lesion type (number of columns will be equal to the number of lesion types in the lsn.type column).
`nsubj.mtx`	A data.frame with number of affected subjects by lesion type in each annotated gene.
`gene.lsn.data`	Each row represent a gene overlapped by a certain lesion. Column "gene" shows the overlapped gene and "ID" column has the patient ID.
`glp.data`	data.frame ordered by gene and lesions start position. Gene start position is coded as 1 in the cty column and gene end position is coded as 4. Lesion start position is coded as 2 in the cty column and lesion end position is coded as 3.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`prep.gene.lsn.data()`, `find.gene.lsn.overlaps()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)

# prepare gene and lesion data for later computations:
prep.gene.lsn=prep.gene.lsn.data(lesion.data,
                                 hg19.gene.annotation)

# determine lesions that overlap each gene (locus):
gene.lsn.overlap=find.gene.lsn.overlaps(prep.gene.lsn)

# count number of subjects affected by different types of lesions and number of hits that affect
# each locus:
count.nsubj.nhits=count.hits(gene.lsn.overlap)

## End(Not run)
```

`default.grin.colors` *Default GRIN Colors*

Description

Function assigns default colors for each lesion group in the whole set of GRIN plots.

Usage

```
default.grin.colors(lsn.types)
```

Arguments

`lsn.types`	Unique lesion types as specified in the lesion data file.

Details

The function specifies 10 colors for different lesion types. If the number of lesion types is more than 10, the user will be asked to specify the colors manually.

Value

Function return a vector of colors assigned to each unique lesion type.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

Examples

```
## Not run:
data(lesion.data)

lsn.types=unique(lesion.data$lsn.type)
# assign colors for different lesion categories using default.grin.colors function:
default.grin.colors(lsn.types)

## End(Not run)
```

`expr.data` *Example T-ALL Dataset Gene Expression Data*

Description

Gene expression data file showing log2 normalized expression level of 420 genes (rows) in 265 newly diagnosed T-cell Acute Lymphoblastic Leukemia (T-ALL) patients in columns that was reported by Liu, Yu, et al. (2017).

Usage

```
expr.data
```

Format

`expr.data`**:**

A data frame with 420 rows and 265 columns:

gene Ensembl IDs of the list of 420 genes included in the dataset ...

Source

Data was extracted from the supplementary material tables of the published Liu, Yu, et al. (2017) manuscript https://www.nature.com/articles/ng.3909#Sec27

`find.gene.lsn.overlaps` *Find Gene Lesion Overlaps*

Description

The function use the output of the prep.gene.lsn.data function to find lesion-gene overlaps.

Usage

```
find.gene.lsn.overlaps(gl.data)
```

Arguments

`gl.data`	a list of five data.frames that represent the output results of the prep.gene.lsn.data function.

Value

A list with the following components:

`lsn.data`	Input lesion data
`gene.data`	Input gene annotation data
`gene.lsn.data`	data.frame ordered by gene and lesions start position. Gene start position is coded as 1 in the cty column and gene end position is coded as 4. Lesion start position is coded as 2 in the cty column and lesion end position is coded as 3.
`gene.lsn.hits`	data.frame on which each row represent a gene overlapped by a certain lesion. The data.frame has 11 columns that include "gene" with ensembl ID of the overlapped gene, "gene.chrom", "gene.loc.start" and "gene.loc.end" with data for the chromosome on which the gene is located, start and end positions of the gene. In addition, column "ID" has the ID of the patient with a lesion that overlapped this gene, "lsn.chrom", "lsn.loc.start", "lsn.loc.end" and "lsn.type" have data for the chromosome, lesion start, lesion end positions and the lesion type respectively.
`gene.index`	data.frame that shows row start and row end for each chromosome in the gene.lsn.data table
`lsn.index`	data.frame that shows row start and row end for each lesion in the gene.lsn.data table

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`prep.gene.lsn.data()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)

# prepare gene and lesion data for later computations:
prep.gene.lsn=prep.gene.lsn.data(lesion.data,
                                 hg19.gene.annotation)

# determine lesions that overlap each gene (locus):
gene.lsn.overlap=find.gene.lsn.overlaps(prep.gene.lsn)

## End(Not run)
```

`genomewide.log10q.plot`

Genomewide log10q Plot

Description

The function return a genome-wide plot based on -log(10) q-value of each of the evaluated annotated genes or lesion boundaries on each chromosome. The plot is lesion type specific (gain, loss, mutation, etc...).

Usage

```
genomewide.log10q.plot(
  grin.res,
  lsn.grps,
  lsn.colors = NULL,
  max.log10q = NULL
)
```

Arguments

`grin.res`	GRIN results evaluating annotated genes or lesion boundaries (output of the grin.stats function using either lesion boundaries or annotated genes as a marker input file).
`lsn.grps`	Selected lesion groups to be added to the plot.
`lsn.colors`	Colors assigned to each lesion group (if NULL, lesion colors will be assigned automatically using default.grin.colors function).
`max.log10q`	Maximum log10 q value to be added to the plot. All boundaries or genes with -log10 q smaller than this value will be set automatically to max.log10q.

Value

The function return a genome-wide plot based on -log(10) q-value of each of the evaluated annotated genes or lesion boundaries to be affected by a certain type of lesions (gain, loss, mutation, etc...).

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

See Also

`grin.lsn.boundaries()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.chrom.size)

# This analysis is lesion type specific. So, user should first data extract data for a specific
# lesion group of interest for example gains from the lesion data file:
```

```
gain=lesion.data[lesion.data$lsn.type=="gain",]
# Return lesion boundaries for gains:
lsn.bound.gain=grin.lsn.boundaries(gain, hg19.chrom.size)
# Run GRIN analysis Using Lesion Boundaries as Markers Instead of the Gene Annotation File:
GRIN.results.gain.bound=grin.stats(gain, lsn.bound.gain, hg19.chrom.size)
# Return genomewide -log10q plot for association between lesion boundaries and gain:
genomewide.log10q.plot(GRIN.results.gain.bound, lsn.grps=c("gain"),
                       lsn.colors=c("gain" = "red"), max.log10q = 10)

# instead of lesion boundaries, users can also plot -log10q values for annotated genes using
# genes annotation data as a marker data file:
grin.results=grin.stats(lesion.data,
                        hg19.gene.annotation,
                        hg19.chrom.size)

genomewide.log10q.plot(grin.results, lsn.grps=c("gain"), lsn.colors=c("gain" = "red"),
                       max.log10q = 10)

# User can run this same analysis for other lesion types such as mutations and deletions.

## End(Not run)
```

genomewide.lsn.plot *Genome-wide Lesion Plot*

Description

Function return a genomewide lesion plot for all lesion types affecting different chromosomes.

Usage

```
genomewide.lsn.plot(
  grin.res,
  ordered = FALSE,
  pt.order = NULL,
  lsn.colors = NULL,
  max.log10q = NULL
)
```

Arguments

`grin.res`	GRIN results (output of the grin.stats function).
`ordered`	By default the function will order the patient IDs alphabetically. However, users can specify a certain patient's order in the genomewide lesion plot by specifying ordered=TRUE and pass a data frame with new patient's order to the pt.order argument.
`pt.order`	data.frame of two columns "ID" that has patient IDs matching the unique IDs in the lesion data file and "pts.order" that has the new patient's order listed as numbers that range from 1:n.patients (Should be only specified if ordered=TRUE).
`lsn.colors`	a vector of lesion colors (If not provided by the user, colors will be automatically assigned using default.grin.colors function).
`max.log10q`	Maximum log10 q value for genes in the GRIN results table to be added to the plot. If max.log10q=100 for example, all -log10q values>100, will be adjusted to 100 in the plot.

Details

The function use the genome-wide plotting coordinates obtained from the compute.gw.coordinates function and plot the whole set of lesions affecting subjects included in the dataset in the middle panel of the figure. Two additional side panels show the number of affected subjects and -log10 q value of each locus to be affected by all different types of lesions.

Value

The function return a genome-wide lesion plot (all chromosomes) in the middle panel. For each locus, Panel on the left shows -log10 q value and the Panel on the right show the number of subjects affected by all different types of lesions color coded by lesion category.

Author(s)

Abdelrahman Elsayed <abdelrahman.elsayed@stjude.org> and Stanley Pounds <stanley.pounds@stjude.org>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

compute.gw.coordinates()

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)
data(hg19.chrom.size)

# Run GRIN model using grin.stats function
grin.results=grin.stats(lesion.data,
                        genome.version="Human_GRCh37")
# Human_GRCh38 can be used in case of hg38 genome assembly

# prepare the genomewide lesion plot using genomewide.lsn.plot function with patient IDs ordered
# alphabetically:
genomewide.plot=genomewide.lsn.plot(grin.results, max.log10q=50)

# To pass certain patients order to the genomewide.lsn.plot function, the user should specify
# ordered=TRUE:
genomewide.lsn.plot(GRIN.results, ordered = TRUE, pt.order = new.pts.order, max.log10q=50)

## End(Not run)
```

`get.chrom.length` *Get Chromosome Length*

Description

Retrieve chromosome size data from chr.info txt files available on the UCSC genome browser based on the user specified genome assembly.

Usage

```
get.chrom.length(genome.assembly)
```

Arguments

`genome.assembly`	User can specify one of four supported genome assemblies that include "Human_GRCh38", "Human_GRCh37", "Mouse_HGCm39" and "Mouse_HGCm38".

Details

Based on the genome assembly specified by the user, the function will directly retrieve chromosome size data from chr.info txt file available on the UCSC genome browser.

Value

A data table with the following two columns:

`chrom`	column has the chromosome number denoted as 1, 2, X, Y, etc..
`size`	column has the chromosome size in base pairs.

Author(s)

Abdelrahman Elsayed <abdelrahman.elsayed@stjude.org> and Stanley Pounds <stanley.pounds@stjude.org>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`read.chromInfo()`

Examples

```
## Not run:
# To retreive chromosome size data for hg19 genome assembly:
hg19.chrom.size=get.chrom.length("Human_GRCh37")
# To retreive chromosome size data for hg38 genome assembly:
hg38.chrom.size=get.chrom.length("Human_GRCh38")

## End(Not run)
```

`get.ensembl.annotation` *Get Ensembl Gene and Regulatory Features Annotation Data*

Description

Function directly retrieve gene and regulatory features annotation data from Ensembl BioMart database based on the specified genome assembly.

Usage

```
get.ensembl.annotation(genome.assembly)
```

Arguments

`genome.assembly`	User can specify one of four genome assemblies that include "Human_GRCh38", "Human_GRCh37", "Mouse_HGCm39" and "Mouse_HGCm38".

Details

Based on the genome assembly specified by the user, the function will directly retrieve gene and regulatory features annotation data from ensembl BioMart database. Annotation data include enesembl ID, the chromosome on which the gene is located, gene start and gene end position, gene name, gene description, biotype, chromosome strand and chromosome band. Gene classes (biotypes) include protein coding genes, long noncoding RNAs (lncRNAs), microRNAs (miRNAs), small nuclear RNAs (snRNA), small nucleolar RNAs (snoRNA), immunoglobulins (IGs), T-cell receptors (TCRs) and pseudogens. Regulatory features data retrieved from Ensembl regulatory build are categorized in 6 classes that include promoters, promoter flanking regions, predicted enhancers, CTCF binding sites, transcription factor (TF) binding sites and the open chromatin regions.Ensembl first imports publicly available data from different large epigenomic consortia such as ENCODE, Roadmap Epigenomics and Blueprint. All high-throughput sequencing data sets are then uniformly processed using the Ensembl Regulation Sequence Analysis (ERSA) pipeline to generate signal tracks for enriched regions also referred to as annotated features or peaks. Segmentation data provide information about promoter, promoter flanking regions, enhancers and CTCF binding sites. If TF binding probability is >0 in areas outside previously mentioned regions, it takes a label of TF binding site. If any open chromatin region did not overlap with the above features, it takes a label of unannotated open chromatin. users will also have the chance to use a list of experimentally verified enhancers/transcription start sites (TSS) using the CAGE (Cap Analysis of Gene Expression) experiment on a multitude of different primary cells and tissues from the Functional Annotation of the Mouse/Mammalian Genome (FANTOM5) project.

Value

A list of three components:

`gene.annotation`	A 9 columns data.frame with gene annotation data that include enesembl ID, chromosome, gene start and gene end position, gene name, gene description, biotype, chromosome strand, and chromosome band.

`reg.annotation.predicted`
: A 5 columns data.frame with regulatory features annotation data directly retreived from Ensembl regulatory build that include enesembl ID, chromosome, description(promoter, enhancer, etc..), feature start and end positions.

`reg.annotation.validated`
: A 5 columns data.frame with regulatory features annotation data for experimentally verified features retreived from FANTOM5 project that include feature ID, chromosome, description(enhancer, transcription start site (TSS)), feature start and end positions.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

Zerbino, Daniel R., et al. (2015). The ensembl regulatory build.

Kinsella, Rhoda J., et al. (2011). Ensembl BioMarts: a hub for data retrieval across taxonomic space.

See Also

`useEnsembl()`, `getBM()`

Examples

```
## Not run:
# Retreive annotation data for human hg19 genome assembly:
hg19.ann=get.ensembl.annotation("Human_GRCh37")
# gene annotation data:
hg19.gene.annotation=hg19.ann$gene.annotation
# regulatory features annotation data retrieved from Ensembl regulatory build:
hg19.reg.annotation=hg19.ann$reg.annotation.predicted
# regulatory features annotation data retrieved from FANTOM5 project:
hg19.fantom.annotation=hg19.ann$reg.annotation.validated

# Retreive annotation data for human hg38 genome assembly:
hg38.ann=get.ensembl.annotation("Human_GRCh38")
# gene annotation data:
hg38.gene.annotation=hg38.ann$gene.annotation
# regulatory features annotation data retrieved from Ensembl regulatory build:
hg38.reg.annotation=hg38.ann$reg.annotation
# regulatory features annotation data retrieved from FANTOM5 project:
hg38.fantom.annotation=hg38.ann$reg.annotation.validated

## End(Not run)
```

grin.assoc.lsn.outcome   *Associate Lesions with Clinical Outcomes*

Description

The function run association analysis between the binary lesion matrix (output of prep.binary.lsn.mtx function) and clinical outcomes of interest such as Minimal Residual Disease (MRD), Event-free Survival (EFS) and Overall Survival (OS), etc...

Usage

```
grin.assoc.lsn.outcome(
  lsn.mtx,
  clin.data,
  annotation.data,
  clinvars,
  covariate = NULL
)
```

Arguments

`lsn.mtx`	Binary lesion matrix in which each type of lesions affecting certain gene is represented in a separate row for example ENSG00000148400_gain. If the gene is affected by this specific type of lesion, patient entry will be coded as 1 or 0 otherwisw. This matrix is the output of the prep.binary.lsn.mtx function.
`clin.data`	Clinical data table in which the first column "ID" should has the patient ID.
`annotation.data`	Gene annotation data either provided by the user or retrieved from ensembl BioMart database using get.ensembl.annotation function included in the GRIN2.0 library. Data.frame should has four columns: "gene" which is the ensembl ID of annotated genes, "chrom" which is the chromosome on which the gene is located, "loc.start" which is the gene start position, and "loc.end" the gene end position.
`clinvars`	Clinical outcome variables of interest (survival variables such as EFS and OS should be first coded as survival objects using Surv function and added as new columns to the clinical data file, binary variables such as MRD should be coded as 0, 1).
`covariate`	Covariates that the model will adjust for if any.

Details

The function run association analysis between the binary lesion matrix in which each type of lesions affecting certain gene is represented in a separate row (output of prep.binary.lsn.mtx function) and clinical outcomes.Function will run logistic regression models for association between each gene-lesion pair with numeric variables such as MRD that should be coded as 0 if the patient is MRD-negative and 1 if the patient is MRD positive. Function will also run COX-Proportional hazard models for association between lesions and survival objects such as Event-free survival (EFS) and oveall survival (OS). EFS and OS should be first coded as survival objects using Surv function and added as new columns to the clinical data file. If specified, the models can be also adjusted for one or a group of covariates such as risk group assignment, gender, age, etc...

Value

Function returns a results table that has gene annotation data, and multiple columns showing results of the logistic regression model for association with binary variables such as MRD that include odds.ratio, lower and upper 95 confidence interval (CI), model p and FDR adjusted q values, in addition to the number of patients with/without lesion who experienced or did not experience the event. Results table will also include results of COXPH models for association between lesions with survival variables such as EFS, OS that include COXPH hazard ratio, lower and upper 95 CI, model p and FDR adjusted q values, in addition to the number of patients with/without the lesion who experienced or did not experience the event.

Author(s)

Abdelrahman Elsayed <abdelrahman.elsayed@stjude.org> and Stanley Pounds <stanley.pounds@stjude.org>

References

Andersen, P. and Gill, R. (1982). Cox's regression model for counting processes, a large sample study.

Therneau, T., Grambsch, P. (2000) Modeling Survival Data: Extending the Cox Model.

Dobson, A. J. (1990) An Introduction to Generalized Linear Models.

See Also

glm(), coxph()

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)
data(clin.data)

# prepare lesion data and find gene lesion overlaps:
gene.lsn=prep.gene.lsn.data(lesion.data, hg19.gene.annotation)
gene.lsn.overlap= find.gene.lsn.overlaps(gene.lsn)

# Prepare a binary lesion matrix for genes affected by a certain type of lesion in at least
# 5 subjects using prep.binary.lsn.mtx function:
lsn.binary.mtx=prep.binary.lsn.mtx(gene.lsn.overlap, min.ngrp=5)

# Prepare EFS and OS survival objects and add two new columns to the clinical data file:
clin.data$EFS <- Surv(clin.data$efs.time, clin.data$efs.censor)
clin.data$OS <- Surv(clin.data$os.time, clin.data$os.censor)
# define clinical outcome variables to be included in the analysis:
clinvars=c("MRD.binary", "EFS", "OS")

# Run association analysis between lesions in the binary lesion matrix and clinical variables
# in the clinvars object:
assc.outcomes=grin.assoc.lsn.outcome(lsn.binary.mtx,
                                     clin.data,
                                     hg19.gene.annotation,
                                     clinvars)

# to adjust the models for one or a group of covariates:
```

```
assc.outcomes.adj=grin.assoc.lsn.outcome(lsn.binary.mtx,
                                         clin.data,
                                         hg19.gene.annotation,
                                         clinvars,
                                         covariate="Sex")

## End(Not run)
```

grin.barplt — *GRIN Bar Plot*

Description

Function return a stacked bar plot with number of patients affected by all different types of lesions in a pre-specified list of genes of interest.

Usage

```
grin.barplt(grin.res, count.genes, lsn.colors = NULL)
```

Arguments

`grin.res`	GRIN results (output of the grin.stats function).
`count.genes`	vector with gene names of a list of genes to be added to the bar plot.
`lsn.colors`	Lesion colors (If not provided by the user, colors will be automatically assigned using default.grin.colors function).

Details

Function will use the input list of gene names and extract the number of patients affected by all different types of lesions in those genes from the GRIN results table (output of the grin.stats function).

Value

Function return a stacked bar plot with number of patients affected by all different types of lesions in the pre-specified list of genes of interest.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

See Also

`grin.stats()`

Examples

```
## Not run:
data(lesion.data)

# run GRIN analysis using grin.stats function
grin.results=grin.stats(lesion.data,
                        genome.version="Human_GRCh37")
# users can use "Human_GRCh38" for hg38 genome assembly

# specify a list of genes to be included in the bar plot (driver genes)
count.genes=as.vector(c("TAL1", "FBXW7", "PTEN", "IRF8","NRAS",
                        "BCL11B", "MYB", "LEF1","RB1", "MLLT3", "EZH2", "ETV6", "CTCF",
                        "JAK1", "KRAS", "RUNX1", "IKZF1", "KMT2A", "RPL11", "TCF7",
                        "WT1", "JAK2", "JAK3", "FLT3"))
# return the stacked barplot
grin.barplt(grin.results, count.genes)

## End(Not run)
```

`grin.lsn.boundaries` *GRIN Evaluate Lesion Boundaries*

Description

The function evaluates Copy number variations that include gain and deletions as boundaries based on unique lesion start and end positions. This analysis is lesion type specific and covers the entire genome.

Usage

```
grin.lsn.boundaries(lsn.data, chrom.size)
```

Arguments

`lsn.data`	Lesion data file that should be limited to include either gain or deletions. If gains are splitted to gain and amplifications based on the log2Ratio value of the CNV segmentation file, the two categories can be included in the same data table, same for homozygous and heterozygous deletions.
`chrom.size`	Chromosize size table that should include two columns "chrom" with the chromosome number and "size" with the chromosome size in base pairs.

Details

The function evaluates Copy number variations that include gain and deletions as boundaries and return a table of ordered boundaries based on the unique start and end positions of different lesions on each chromosome. If gains are splitted to gain and amplifications based on the log2Ratio value of the CNV segmentation file, the two categories can be included in the same analysis, same for homozygous and heterozygous deletions. Boundary will be the region between each unique start and end positions where large size lesions will be splitted into multiple boundaries based on other smaller size lesions that affect the same region in other patients if any. This analysis is meant to cover the entire genome, so regions without any annotated genes or regulatory features will be incuded will be assesed in the analysis. The first boundary for each chromosome will start from the first nucleotide base on the chromosome till the start position of the first lesion that affect the chromosome. Similarly, the last boundary will start from the end position of the last lesion that affect the chromosome till the last base on the chromosome.

Value

Function return a data.frame with five columns:

`gene`	Ordered boundaries by unique start and end positions of different lesions on each chromosome.
`chrom`	Chromosome on which the bounday is located.
`loc.start`	Boundary start position.
`loc.end`	Boundary end position.
`diff`	Boundary size in base pairs.

Author(s)

Abdelrahman Elsayed <abdelrahman.elsayed@stjude.org> and Stanley Pounds <stanley.pounds@stjude.org>

Examples

```
## Not run:
data(lesion.data)
data(hg19.chrom.size)

# This analysis is lesion type specific. So, user should first data extract data for a specific
# lesion group of interest for example gains from the lesion data file:
gain=lesion.data[lesion.data$lsn.type=="gain",]
# Return lesion boundaries for gains:
lsn.bound.gain=grin.lsn.boundaries(gain, hg19.chrom.size)
# Run GRIN analysis Using Lesion Boundaries markers Instead of the gene annotation file:
GRIN.results.gain.bound=grin.stats(gain, lsn.bound.gain, hg19.chrom.size)

# To extract deletions from the lesion data file
loss=lesion.data[lesion.data$lsn.type=="loss",]
# To return lesion boundaries for deletions:
lsn.bound.loss=grin.lsn.boundaries(loss, hg19.chrom.size)
# Run GRIN analysis Using Lesion Boundaries markers Instead of the gene annotation file:
GRIN.results.loss.bound=grin.stats(loss, lsn.bound.loss, hg19.chrom.size)

# same analysis can be done for mutations and structural rearrangments.

## End(Not run)
```

`grin.oncoprint.mtx` *GRIN OncoPrint Matrix*

Description

Function use GRIN results table and prepare the lesion matrix that the user can pass to the oncoprint function from ComplexHeatmap package to geneate an OncoPrint for a selcted list of genes.

Usage

```
grin.oncoprint.mtx(grin.res, oncoprint.genes)
```

Arguments

`grin.res`	GRIN results (output of the grin.stats function).
`oncoprint.genes`	Vector of ensembl IDs for the selected list of genes to be added to the OncoPrint.

Details

Function will use the input list of ensembl IDs to prepare a data table of lesions that affect these genes (each row is a gene and each column is a patient ID). This lesion matrix is compatible and can be passed to oncoprint function in ComplexHeatmap library to prepare an OncoPrint for lesions in the selected list of genes.

Value

Function uses the output results of grin.stats function and return data table of lesions that affect a group of selected genes (each row is a gene and each column is a patient ID).

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`grin.stats()`

Examples

```
## Not run:
data(lesion.data)

# Run GRIN analysis using grin.stats function:
grin.results=grin.stats(lesion.data,
                        genome.version="Human_GRCh37")

# specify a list of genes to be included in the oncoprint (driver genes):
oncoprint.genes=as.vector(c("ENSG00000148400", "ENSG00000171862", "ENSG00000171843",
                            "ENSG00000156531", "ENSG00000162367", "ENSG00000096968",
                            "ENSG00000105639", "ENSG00000118513","ENSG00000102974",
                            "ENSG00000133703"))

# prepare the oncoprint lesion matrix:
oncoprint.mtx=grin.oncoprint.mtx(grin.results,
                                 oncoprint.genes)

# user can also specify a list of top significant genes in the GRIN constellation test:
# for example: select genes affected by two types of lesion with q2.nsubj<0.01:
```

```
genes.const = grin.results$gene.hits[grin.results$gene.hits$q2.nsubj < 0.01, ]
# get ensembl.ids for this list of genes
selected.genes=as.vector(genes.const$gene)
oncoprint.mtx.const=grin.oncoprint.mtx(grin.results,
                                       selected.genes)

## End(Not run)
```

---

grin.stats *GRIN Statistics Output*

Description

The function run the Genomic Random Interval (GRIN) analysis to determine whether a certain locus has an abundance of lesions or a constellation of multiple types of lesions that is statistically significant.

Usage

```
grin.stats(lsn.data, gene.data = NULL, chr.size = NULL, genome.version = NULL)
```

Arguments

`lsn.data`	data.frame with lesion data prepared by the user in a GRIN compatible format. Object should has five columns that include "ID" with patient ID, "chrom" which is the chromosome on which the lesion is located, "loc.start" which is the lesion start position, "loc.end" the lesion end position and "lsn.type" which is the lesion category for example gain, loss, mutation, fusion, etc... For Single Nucleotide Variants (SNVs), loc.start will be the same as loc.end. For Copy Number Alterations (CNAs) such as gain and deletions, loc.start and loc.end should be the gain or deletion start and end positions respectively. For structural rearrangements such as inversions and translocations, each rearrangement should be coded in two different lines, one line for chromosome A involved in the translocation break-point and the second line for chromosome B break-point. For inversions on the same chromosome, the two lines will include the two breakpoints of the inversion. An example lesion data in a GRIN compatible format can be found at the GRIN2.0 package data folder (lesion.data.rda).
`gene.data`	data.frame with the gene annotation data either provided by the user or directly retreived from ensembl BioMart database using get.ensembl.annotation function included in the GRIN2.0 library if the genome.version is specified. Object should has four columns "gene" which is the ensembl ID of annotated genes to which the lesion data will be overlapped, "chrom" which is the chromosome on which the gene is located, "loc.start" which is the gene start position, and "loc.end" the gene end position.
`chr.size`	data.frame with the size of the 22 autosomes, in addition to X and Y chromosomes in base pairs. It should has two columns that include "chrom" with the chromosome number and "size" for the size of the chromosome in base pairs. Chromosome size data can be either provided by the user or directly retreived from UCSC genome browser using get.chrom.length function included in the GRIN2.0 library if genome.version is specified.

`genome.version`	Genome assembly should be only specified if the user selected not to provide gene annotation, chromosome size files, and directly retreive those files from ensembl BioMart database, and UCSC genome browsers using get.ensembl.annotation and get.chrom.length functions respectively. Currently, the function support four genome assemblies that include "Human_GRCh38", "Human_GRCh37", "Mouse_HGCm39", and "Mouse_HGCm38".

Details

The function run the Genomic Random Interval (GRIN) analysis and evaluates the probability of each gene locus to be affected by different types of lesions based on a convolution of independent but non-identical Bernoulli distributions to determine whether this locus has an abundance of lesions that is statistically significant.In addition, FDR-adjusted q value is computed for each locus based on Pounds & Cheng (2006) estimator of the proportion of tests with a true null (pi.hat). The function also evaluates if a certain locus is affected by a constellation of multiple types of lesions and return the GRIN results table.

Value

A list with the following components:

`gene.hits`	data table of GRIN results that include gene annotation, number of subjects affected by each lesion type for example gain, loss, mutation, etc.., and number of hits affecting each locus. The GRIN results table will also include P and FDR adjusted q-values showing the probability of each locus of being affected by one or a constellation of multiple types of lesions.
`lsn.data`	input lesion data
`gene.data`	input gene annotation data
`gene.lsn.data`	each row represent a gene overlapped by a certain lesion. Column "gene" shows the overlapped gene ensembl ID and "ID"" column has the patient ID.
`chr.size`	data table showing the size of the 22 autosomes, in addition to X and Y chromosomes in base pairs.
`gene.index`	data.frame with overlapped gene-lesion data rows that belong to each chromosome in the gene.lsn.data table.
`lsn.index`	data.frame that shows the overlapped gene-lesion data rows taht belong to each lesion in the gene.lsn.data table.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`prep.gene.lsn.data()`, `find.gene.lsn.overlaps()`, `count.hits()`, `prob.hits()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)
data(hg19.chrom.size)

# if gene annotation and chromosome size files will be provided by the user:
grin.results=grin.stats(lesion.data,
                        hg19.gene.annotation,
                        hg19.chrom.size)

# to directly retrieve gene annotation and chromosome size files from Ensembl BioMart database,
# and UCSC genome browsers using get.ensembl.annotation and get.chrom.length functions respectively:
# for the GRCH37 (hg19) genome assembly:
grin.results=grin.stats(lesion.data, genome.version="Human_GRCh37")
# for the GRCH38 (hg38) genome assembly:
grin.results=grin.stats(lesion.data, genome.version="Human_GRCh38")

## End(Not run)
```

`grin.stats.lsn.plot` *GRIN Statistics Lesions Plot*

Description

Function return a plot with all types of lesions that spans either a gene or regulatory feature of interest with GRIN statistics added.

Usage

```
grin.stats.lsn.plot(grin.res, feature = NULL, lsn.clrs = NULL, expand = 5e-04)
```

Arguments

`grin.res`	GRIN results for genes or regulatory features (output of the grin.stats function)
`feature`	Feature ensembl ID from Ensembl regulatory build or FANTOM5 project. An ensembl ID of a gene can be provided as well.
`lsn.clrs`	Assigned colors per lesion types. If not specified, colors will be automatically assigned using default.grin.colors function.
`expand`	Controls ratio of the feature locus (start and end position) to the whole plot with default value = 0.0005 (setting expand=0 will only plot the locus from the start to the end position without any of the upstream or downstream regions of the feature).

Details

Function return a plot with all lesions that affect either a gene regulatory feature of interest. Top panel of the plot will has all different types of lesions affecting the loci color coded according to the figure legend. Lower panel of the plot has all the GRIN statistics of the feature that include number of subjects affected by each type of lesions, -log10 p, and –log10q values showing if the feature is significantly affected by the corresponding lesion category. This plot has no panel for transcripts table as regulatory features typically do not have this kind of information.

Value

Function return a plot with all types of lesions that spans either a gene or regulatory feature of interest in addition to the locus GRIN statistics without adding the transcripts panel.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`grin.stats()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.chrom.size)

# Retrieve gene and regulatory features annotation data from Ensembl BioMart:
hg19.ann=get.ensembl.annotation("Human_GRCh37")
# Human_GRCh38 can be used instead of Human_GRCh37 for hg38
# get regulatory features annotation data from Ensembl regulatory build:
hg19.reg.annotation=hg19.ann$reg.annotation.predicted
# get regulatory features annotation data from FANTOM5 project:
hg19.reg.FANTOM=hg19.ann$reg.annotation.validated

# Run GRIN for computationally predicted regulatory features from Ensembl regulatory build:
grin.results.reg=grin.stats(lesion.data,
                            hg19.reg.annotation,
                            hg19.chrom.size)

# Plots showing different types of lesions affecting regulatory feature of interest:
reg.plot=grin.stats.lsn.plot(grin.results.reg, feature="ENSR00002029697")

# For experimentally verified regulatory features from FANTOM5 project:
grin.results.FANTOM=grin.stats(lesion.data,
                               hg19.reg.FANTOM5,
                               hg19.chrom.size)

# Plots showing different types of lesions affecting regulatory feature from FANTOM5 Project:
reg.FANTOM=grin.stats.lsn.plot(grin.results.FANTOM, feature="p6@NRAS,0.2452")

## End(Not run)
```

`hg19.chrom.size` *Chromosome Length Data File*

Description

The file has the size of 22 autosomes in addition to X and Y chromosomes in base pairs directly retrieved from chr.info txt files available on the UCSC genome browser using get.chrom.length function and "Human-GRCh37" as a genome assembly option (hg19).

Usage

```
hg19.chrom.size
```

Format

`hg19.chrom.size`**:**

A data frame with 24 rows and 2 columns:

chrom The chromosome number.

size The chromosome length in base pairs.

Source

Chromosome size data directly retrieved from chr.info txt files available on the UCSC genome browser using get.chrom.length function and "Human-GRCh37" as a genome assembly option (hg19).

`hg19.gene.annotation` *Example Gene Annotation Data File*

Description

The file has an example annotation data of 420 genes (same set of genes in the gene expression data file) directly retrieved from Ensembl BioMart database using get.ensembl.annotation function and "Human-GRCh37" as a genome assembly option (hg19).

Usage

```
hg19.gene.annotation
```

Format

`hg19.gene.annotation`**:**

A data frame with 420 rows and 9 columns:

gene Column has the gene ensembl ID

chrom The chromosome on which the gene is located

loc.start Gene start position in base pairs

loc.end Gene end position in base pairs

description Description of the gene name

gene.name Gene symbol

biotype Gene classes that include protein coding genes, long noncoding RNAs (lncRNAs), microRNAs (miRNAs), small nuclear RNAs (snRNA), small nucleolar RNAs (snoRNA), immunoglobulins (IGs), T-cell receptors (TCRs) and pseudogens.

chrom.strand The chromosome strand on which the gene is located forward (1) or reverse (-1).

chrom.band The chromosome band on which the gene is located.

Source

Data was directly retrieved from Ensembl BioMart database using get.ensembl.annotation function and "Human-GRCh37" as a genome assembly option (hg19).

`hg19_cytoband` *GRCh37 Chromosome Cytobands*

Description

The dataset has the start and end positions in base pairs of all 22 autosomes in addition to X and Y chromosome cytobands for Human-GRCh37 (hg19) genome assembly.

Usage

```
hg19_cytoband
```

Format

`hg19_cytoband`**:**

A data frame with 862 rows and 5 columns:

chrom The chromosome number.

chromStart The cytoband start position on the chromosome in base pairs.

chromEnd The cytoband end position on the chromosome in base pairs.

name The cytoband name.

gieStain The coloring scheme of the cytobands.

Source

The Chromosome cytobands data file was downloaded from the UCSC genome browser for GRCh37 genome assembly `https://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/`.

`hg38_cytoband` *GRCh38 Chromosome Cytobands*

Description

The dataset has the start and end positions in base pairs of all 22 autosomes in addition to X and Y chromosome cytobands for Human-GRCh38 (hg38) genome assembly.

Usage

```
hg38_cytoband
```

Format

`hg38_cytoband`**:**

A data frame with 1,549 rows and 5 columns:

chrom The chromosome number.

chromStart The cytoband start position on the chromosome in base pairs.

chromEnd The cytoband end position on the chromosome in base pairs.

name The cytoband name.

gieStain The coloring scheme of the cytobands.

Source

The Chromosome cytobands data file was downloaded from the UCSC genome browser for GRCh38 genome assembly `https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/`.

`KW.hit.express` *Associate Lesion with Expression Data Using Kruskal-Wallis Test*

Description

Function uses Kruskal-Wallis test to evaluate the association between lesion groups and expression level of the same corresponding gene.

Usage

```
KW.hit.express(alex.data, gene.annotation, min.grp.size = NULL)
```

Arguments

`alex.data`	output of the alex.prep.lsn.expr function. It's a list of three data tables that include "row.mtch", "alex.expr" with expression data, "alex.lsn" with lesion data. Rows of alex.expr, and "alex.lsn" matrices are ordered by gene ensembl IDs and columns are ordered by patient ID.

`gene.annotation`	Gene annotation data either provided by the user or retrieved from ensembl BioMart database using get.ensembl.annotation function included in the GRIN2.0 library. Data.frame should has four columns: "gene" which is the ensembl ID of annotated genes, "chrom" which is the chromosome on which the gene is located, "loc.start" which is the gene start position, and "loc.end" the gene end position.
`min.grp.size`	Minimum number of subjects in a lesion group to be included in the KW test (there should be at least two groups with number of patients > min.grp.size) to run the KW test for a certain gene.

Details

The function uses the ensembl IDs in each row of the row.mtch file and run the Kruskal-Wallis test for association between lesion groups of the gene in the "hit.row" column with expression level of the gene in the "expr.row" column. IDs in the two columns should be the same if the KW test will be used to evaluate association between lesion groups and expression level of the same corresponding gene. If the same patient is affected with multiple types of lesions in the same gene for example gain AND mutations, the entry will be denoted as "multiple" and patients without any type of lesions will be coded as "none".

Value

A data table with multiple columns that include:

`gene`	ensembl ID of the gene of interest.
`gene.name`	Gene name of the gene of interest.
`p.KW`	Kruskal-Wallis test p-value.
`q.KW`	Kruskal-Wallis test FDR adjusted q-value.
`_n.subjects`	Multiple columns with number of subjects with each type of lesion affecting the gene, number of subjects without any lesion and number of subjects with multiple types of lesions.
`_mean`	Multiple columns with mean expression level of the gene in subjects with each type of lesion, mean expression in subjects without any lesion and mean expression in subjects with multiple types of lesions.
`_median`	Multiple columns with median expression of the gene in subjects with each type of lesion, median expression in subjects without any lesion and median expression in subjects with multiple types of lesions.
`_sd`	Multiple columns with standard deviation of the expression level of the gene in subjects with each type of lesion, standard deviation in subjects without any lesion and standard deviation in subjects with multiple types of lesions.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Myles Hollander and Douglas A. Wolfe (1973), Nonparametric Statistical Methods. New York: John Wiley & Sons. Pages 115–120.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`alex.prep.lsn.expr()`

Examples

```
## Not run:
data(expr.data)
data(lesion.data)
data(hg19.gene.annotation)

# prepare expression, lesion data and return the set of genes with both types of data available
# ordered by gene IDs in rows and patient IDs in columns:
alex.data=alex.prep.lsn.expr(expr.data, lesion.data,
                             hg19.gene.annotation, min.expr=1, min.pts.lsn=5)

# run Kruskal-Wallis test for association between lesion groups and expression level of the
# same corresponding gene:
alex.kw.results=KW.hit.express(alex.data, hg19.gene.annotation, min.grp.size=5)

## End(Not run)
```

`lesion.data` *Example T-ALL Dataset Lesion Data*

Description

Lesion data file showing copy number variations, single nucleotide variations and structrual rearrangments affecting 265 newly diagnosed T-cell Acute Lymphoblastic Leukemia (T-ALL) patients that was reported by Liu, Yu, et al. (2017).

Usage

```
lesion.data
```

Format

`lesion.data`**:**

A data frame with 6,887 rows and 5 columns:

ID patient identifier for the patient affected by the genomic lesion

chrom the chromosome on which the lesion is located

loc.start the lesion start position in base pairs

loc.end the lesion end position in base pairs

lsn.type the lesion type for example gain, loss, mutation, fusion, etc...

Source

extracted from the supplementary material tables of the published Liu, Yu, et al. (2017) manuscript https://www.nature.com/articles/ng.3909#Sec27

lsn.transcripts.plot *Lesions Gene Transcripts Plot*

Description

Function prepare a plot with all types of lesions that spans either a gene or a small region of interest.

Usage

```
lsn.transcripts.plot(
  grin.res,
  genome,
  gene = NULL,
  lsn.clrs = NULL,
  chrom = NULL,
  plot.start = NULL,
  plot.end = NULL,
  lesion.grp = NULL,
  spec.lsn.clr = NULL,
  extend.left = NULL,
  extend.right = NULL,
  expand = 5e-04,
  hg38.transcripts = NULL,
  hg19.cytoband = NULL,
  hg38.cytoband = NULL
)
```

Arguments

`grin.res`	GRIN results (output of the grin.stats function).
`genome`	either "hg19" or "hg38" genome assemblies can be specified based on the genome assembly that has been used to prepare the lesion data.
`gene`	Gene name of interest.
`lsn.clrs`	Lesion colors for the regional gene plot (If not provided by the user, colors will be automatically assigned using default.grin.colors function).
`chrom`	chromosome number (should be only specified in the locus plots where plot.start and plot.end for the locus of interest are specified).
`plot.start`	start position of the locus of interest.
`plot.end`	end position of the locus of interest.
`lesion.grp`	lesion group of interest (should be only specified in locus plots when chrom, plot.start, plot.end are specified).
`spec.lsn.clr`	color assigned to the lesion of interest (should be specified when chrom, plot.start, plot.end and lesion.grp are specified).
`extend.left`	specified number will be used to manually align the left side of the gene transcripts track directly retrieved from ensembl database with the gene lesions track if needed.
`extend.right`	specified number will be used to manually align the right side of the gene transcripts track directly retrieved from ensembl database with the gene lesions track if needed.

`expand`	Controls ratio of the gene locus (start and end position) to the whole plot with default value = 0.0005 (setting expand=0 will only plot the gene locus from the start to the end position without any of the upstream or downstream regions of the gene).
`hg38.transcripts`	transcripts data retrieved from annotation hub for hg38 version 110 (should be only specified if genome="hg38").
`hg19.cytoband`	hg19 chromosome bands start and end data in base pair (should be only specified if genome="hg19").
`hg38.cytoband`	hg38 chromosome bands start and end data in base pair (should be only specified if genome="hg38").

Details

Function return a plot with all lesions that affect either a gene or a region of interest. Top panel of the regional gene plot has the transcripts track with all transcripts annotated to the gene of interest directly retrieved from ensembl database. The middle panel will has all different types of lesions affecting the gene color coded according to the figure legend. Lower panel of the plot has all the GRIN statistics of the gene that include number of subjects affected by each type of lesions, -log10 p, and –log10q values showing if the gene is significantly affected by the corresponding lesion category. If a certain locus is specified, only transcripts track and the lesion panel will be returned (GRIN results panel will not be added to the plot).

Value

Function will return either a gene plot with the transcripts track, lesions panel and GRIN statistic for the gene of interest or a plot with all lesions and transcripts alligned to a certain locus of interest if chrom, plot.start and plot.end were specified.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`grin.stats()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)
data(hg19.chrom.size)
data(hg19_cytoband)
data(hg38_cytoband)

# run GRIN analysis using grin.stats function
grin.results=grin.stats(lesion.data,
                        hg19.gene.annotation,
                           hg19.chrom.size)

# Plots showing different types of lesions affecting a gene of interest with a transcripts
# track that show all the gene transcripts retrieved from Ensembl (hg19 genome assembly):
WT1.gene.plot=lsn.transcripts.plot(grin.results, genome="hg19", gene="WT1",
                                    hg19.cytoband=hg19_cytoband)

# Plots showing different types of lesions affecting a region of interest with a transcripts
# track added to the plot:
locus.plot=lsn.transcripts.plot(grin.results, genome="hg19", hg19.cytoband=hg19_cytoband,
                                chrom=9, plot.start=21800000, plot.end=22200000,
                                lesion.grp = "loss", spec.lsn.clr = "blue")

# for GRCh38 (hg38) genome assembly:
ah <- AnnotationHub()
# retrieve gene transcripts for human GRCh38 genome assembly from Ensembl (version 110):
gtf.V110 <- ah[["AH113665"]]

# run GRIN analysis using grin.stats function
grin.results=grin.stats(lesion.data,
                       genome.version="Human_GRCh38")
# specifying genome version will directly retreive gene annotation and chrome size data

# Plots showing different types of lesions affecting a gene of interest with a transcripts
# track that show all the gene transcripts retrieved from Ensembl (hg38 genome assembly):
lsn.transcripts.plot(grin.results, genome="hg38", gene="WT1", hg38.transcripts=gtf.V110,
                       hg38.cytoband=hg38_cytoband)

# Plots showing different types of lesions affecting a region of interest with a transcripts
# track added to the plot:
lsn.transcripts.plot(grin.results, genome="hg38", hg38.transcripts="gtf.v110",
                     hg38.cytoband=hg38_cytoband, chrom=9, plot.start=21800000,
                     plot.end=22200000, lesion.grp = "loss", spec.lsn.clr = "blue")

## End(Not run)
```

onco.print.props *Oncoprint proportions*

Description

The function order lesion types based on their average size and assign the proportion of the oncoprint rectangle that should be color filled based on the average size of each lesion type.

Usage

```
onco.print.props(lsn.data, clr = NULL, hgt = NULL)
```

Arguments

`lsn.data`	data.frame with 5 columns including "ID" which is the subject identifier, "chrom" which is the chromosome on which the lesion is located, "loc.start" with lesion start position, "loc.end" which is the lesion end position), and "lsn.type" which is the lesion category for example gain, mutation, etc..).

`clr`	Lesion colors (If not provided by the user, colors will be automatically assigned using default.grin.colors function).
`hgt`	Manually assign the proportion of the oncoprint rectangle that should be color filled for each lesion group.

Details

Some patients might be affected by two or more lesion types in the same gene for example gain AND mutations. To get all lesion types represented in the same rectangle in the oncoprint, this function order lesion types based on the average size of each type and assign the proportion of the oncoprint rectangle that should be color filled based on the average size of each lesion type. Color filled proportion of the oncoprint rectangles can be also specified by the user for each lesion type based on the hgt parameter.

Value

Function return a list of three lists specifying the color assigned to each lesion type, the proportion of the rectangle that should be color filled in the oncoprint based on the average size of each lesion type, and the legend parameters.

Author(s)

Lakshmi Patibandla <`LakshmiAnuhya.Patibandla@stjude.org`>, Abdelrahman Elsayed <`abdelrahman.elsayed@stj`
and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

Examples

```
## Not run:
data(lesion.data)
onco.props=onco.print.props(lesion.data, hgt = c("gain"=4, "loss"=3, "mutation"=2, "fusion"=1))
# if hgt argument is not specified, the lesion category "mutation" for single point mutations will
# be assigned size=1 because it has the smallest average lesion size and will have the smallest
# proportion of the filled oncoprint rectangles 1/4=0.25

## End(Not run)
```

`order.index.gene.data` *Order Index Gene Data*

Description

This function order and index gene annotation data by chromosome on which the gene is located, gene start, and end positions.

Usage

```
order.index.gene.data(gene.data)
```

Arguments

`gene.data`	data.frame with gene annotation data either provided by the user or retrieved from ensembl BioMart database using get.ensembl.annotation function included in the GRIN2.0 library. data.frame should has four columns that include "gene" which is the ensembl ID of the annotated genes to which the lesion data will be overlapped, "chrom" which is the chromosome on which the gene is located, "loc.start" which is the gene start position, and "loc.end" the gene end position.

Value

A list with the following components:

`gene.data`	Input gene annotation data
`gene.index`	data.frame with two columns of ordered row start and row end based on the number of genes annotated to each chromosome.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

Examples

```
## Not run:
data(hg19.gene.annotation)

ordered.genes=order.index.gene.data(hg19.gene.annotation)

## End(Not run)
```

`order.index.lsn.data` *Order Index Lesion Data*

Description

This function order and index lesion data by lesion type, the chromosome on which the lesion is located , and subject.

Usage

```
order.index.lsn.data(lsn.data)
```

Arguments

`lsn.data`	data.frame with lesion data prepared by the user in a GRIN compatible format. The data.frame should has five columns that include "ID" which is a column with id of the patient affected by the lesion, "chrom" which is the chromosome on which the lesion is located, "loc.start" which is the lesion start position, "loc.end" the lesion end position and "lsn.type" which is the lesion type for example gain, loss, mutation, fusion, etc...

Value

A list with the following components:

`lsn.data`	Input lesion data
`lsn.index`	data.frame with row start and row end for each type of lesions affecting each subject on a certain chromosome. For example, if a certain patient is affected by 1 deletion on chromosome 5, row start wil be equal to row end for loss on chromosome 5. However, if the patient is affected by 4 deletions, difference between row.start and row.end will be 3.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

Examples

```
## Not run:
data(lesion.data)

ordered.lsn=order.index.lsn.data(lesion.data)

## End(Not run)
```

`pathways` *List of Genes Annotated to a Group of Pathways*

Description

The dataset has a list of genes annotated to a group of different pathways.

Usage

```
pathways
```

Format

`pathways`:
A data frame with 121 rows and 3 columns:

gene.name	Gene symbol.
ensembl.id	Gene ensembl ID.
pathway	The pathway to which the gene is annotated.

Source

Data was extracted from the supplementary material tables of the published Liu, Yu, et al. (2017) manuscript `https://www.nature.com/articles/ng.3909#Sec27`

`prep.binary.lsn.mtx` *Prepare Binary Lesion Matrix*

Description

Prepares a lesion matrix with each gene affected by a certain type of lesion as a row and each patient as a column.

Usage

```
prep.binary.lsn.mtx(ov.data, min.ngrp = 0)
```

Arguments

`ov.data`	list of six data.frames that represent the output results of the find.gene.lsn.overlaps function.
`min.ngrp`	if specified, rows with number of patients affected by a specific type of lesion that's less than the specified number will be discarded (default is 0; function will return all genes affected by a lesion in at least one patient), for example if only one patient is affected by gain in MYB gene.

Details

The function uses the output results of the find.gene.lsn.overlaps function and create a binary lesion matrix with each gene affected by certain lesion type as a row and each patient as a column. Row-names are labelled as gene.ID_lesion.type (for example: ENSG00000118513_gain for gains affecting MYB gene). The entry for each patient in the table will be denoted as 1 if the patient is affected by this specific type of lesion in the gene, for example gain in MYB gene (ENSG00000118513) or 0 otherwise.

Value

The function returns a binary lesion matrix with each row labelled as gene.ID_lesion.type and each column is a patient. Entry for each patient in the table will be denoted as 1 if the gene is affected by this specific type of lesion or 0 otherwise.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`prep.gene.lsn.data()`, `find.gene.lsn.overlaps()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)

# prepare gene and lesion data for later computations:
prep.gene.lsn=prep.gene.lsn.data(lesion.data,
                                  hg19.gene.annotation)

# determine lesions that overlap each gene (locus):
gene.lsn.overlap=find.gene.lsn.overlaps(prep.gene.lsn)

# prepare the lesion binary matrix with a minimum of 5 patients affected by the lesion to be
# included in the final matrix:
lsn.binary.mtx=prep.binary.lsn.mtx(gene.lsn.overlap, min.ngrp=5)

## End(Not run)
```

`prep.gene.lsn.data` *Prepare Gene and Lesion Data*

Description

This function prepare gene and lesion data for later GRIN computations.

Usage

```
prep.gene.lsn.data(lsn.data, gene.data, mess.freq = 10)
```

Arguments

`lsn.data`	data.frame with lesion data prepared by the user in a GRIN compatible format. The data.frame should has five columns that include "ID" which is the patient ID, "chrom" which is the chromosome on which the lesion is located, "loc.start" which is the lesion start position, "loc.end" the lesion end position and "lsn.type" which is the lesion type for example gain, loss, mutation, fusion, etc...
`gene.data`	gene annotation data with four required columns: "gene" has ensembl ID, "chrom" which is chromosome on which the gene is located, "loc.start" gene start position, "loc.end" which is the gene end position
`mess.freq`	message frequency: display message every mess.freq^th lesion block (default is 10).

Details

This function order and index gene and lesion data for later computations. Output of this function is used to ovelap gene and lesion data using find.gene.lsn.overlaps function.

Value

A list with the following components:

`lsn.data`	Input lesion data.
`gene.data`	Input gene annotation data.
`gene.lsn.data`	data.frame ordered by gene and lesions start positions. Gene start position is coded as 1 in the cty column and gene end position is coded as 4. Lesion start position is coded as 2 in the cty column and lesion end position is coded as 3.
`gene.index`	data.frame that shows row start and row end for each chromosome in the gene.lsn.data table.
`lsn.index`	data.frame that shows row start and row end for each lesion in the gene.lsn.data table.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`order.index.gene.data()`, `order.index.lsn.data()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)

# prepare gene and lesion data for later computations:
prep.gene.lsn=prep.gene.lsn.data(lesion.data,
                                 hg19.gene.annotation)

## End(Not run)
```

`prep.lsn.type.matrix` *Prepare Lesion Type Matrix*

Description

The function prepare a lesion matrix with all types of lesions affecting certain gene as a row and each patient as a column.

Usage

```
prep.lsn.type.matrix(ov.data, min.ngrp = 0)
```

Arguments

`ov.data`	list of six data.frames that represent the output results of the find.gene.lsn.overlaps function.
`min.ngrp`	if specified, rows with number of patients affected by all different types of lesions that's less than the specified number will be discarded (default is 0; will return all genes affected by any type of lesions in at least one patient).

Details

The function returns a lesion matrix with each row as a gene and each column is a patient. If a gene is affected by one type of lesions in a certain patient, the entry will be labelled by lesion type (for example: gain OR mutation). However, if the same gene is affected by more than one type of lesions in a certain patient (for example: gain AND mutation), the entry will be labelled as "multiple". If the gene is not affected by any lesion, the entry for this patient will be labelled as "none".

Value

The function returns a lesion matrix with all types of lesions affecting certain gene as a row and each patient as a column.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`prep.gene.lsn.data()`, `find.gene.lsn.overlaps()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)

# prepare gene and lesion data for later computations:
prep.gene.lsn=prep.gene.lsn.data(lesion.data,
                                 hg19.gene.annotation)

# determine lesions that overlap each gene (locus):
gene.lsn.overlap=find.gene.lsn.overlaps(prep.gene.lsn)

# prepare the lesion matrix with a minimum of 5 patients affected by any type of lesion in the
# gene to be included in the final matrix
lsn.type.mtx=prep.lsn.type.matrix(gene.lsn.overlap, min.ngrp=5)

## End(Not run)
```

`prob.hits` *Find Probablity of Locus Hit*

Description

The function evaluates the probability of a locus to be affected by one or a constellation of multiple types of lesions.

Usage

```
prob.hits(hit.cnt, chr.size = NULL)
```

Arguments

`hit.cnt`	output results of the count.hits function with number of subjects and number of hits affecting each locus.
`chr.size`	data.frame with the size of the 22 autosomes, in addition to X and Y chromosomes in base pairs. The data.frame should has two columns "chrom" with the chromosome number and "size" for the size of the chromosome in base pairs.

Details

The function computes p-value for the probability of each locus (gene or regulatory feature) to be affected by different types of lesions based on a convolution of independent but non-identical Bernoulli distributions to determine whether a certain locus has an abundance of lesions that is statistically significant.In addition, FDR-adjusted q value is computed for each locus based on Pounds & Cheng (2006) estimator of the proportion of tests with a true null (pi.hat). The function also evaluates if a certain locus is affected by a constellation of multiple types of lesions and computes a p and adjusted q values for the locus to be affected by one type of lesions (p1), two types of lesions (p2), etc...

Value

A list with the following components:

`gene.hits`	data table of GRIN results that include gene annotation, number of subjects affected by each lesion type for example gain, loss, mutation, etc.., and number of hits affecting each locus. The GRIN results table will also include P and FDR adjusted q-values showing the probability of each locus of being affected by one or a constellation of multiple types of lesions.
`lsn.data`	input lesion data
`gene.data`	input gene annotation data
`gene.lsn.data`	each row represent a gene overlapped by a certain lesion. Column "gene" shows the overlapped gene ensembl ID and "ID"" column has the patient ID.
`chr.size`	data table showing the size of the 22 autosomes, in addition to X and Y chromosomes in base pairs.
`gene.index`	data.frame with overlapped gene-lesion data rows that belong to each chromosome in the gene.lsn.data table.
`lsn.index`	data.frame that shows the overlapped gene-lesion data rows taht belong to each lesion in the gene.lsn.data table.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`prep.gene.lsn.data()`, `find.gene.lsn.overlaps()`, `count.hits()`

Examples

```
## Not run:
data(lesion.data)
data(hg19.gene.annotation)
data(hg19.chrom.size)

# prepare gene and lesion data for later computations:
prep.gene.lsn=prep.gene.lsn.data(lesion.data,
                                 hg19.gene.annotation)

# determine lesions that overlap each gene (locus):
gene.lsn.overlap=find.gene.lsn.overlaps(prep.gene.lsn)

# count number of subjects affected by different types of lesions and number of hits that affect
# each locus:
count.subj.hits=count.hits(gene.lsn.overlap)
```

```
# compute the probability of each locus to be affected by one or a constellation of multiple
# types of lesion
hits.prob=prob.hits(count.subj.hits, hg19.chrom.size)

## End(Not run)
```

`top.alex.waterfall.plots` *Waterfall Plots for Lesion and Expression Data of Top Significant Genes*

Description

Function return waterfall plots for top significant genes in the KW results table based on the specified q value.

Usage

```
top.alex.waterfall.plots(out.dir, alex.data, alex.kw.results, q, lsn.data)
```

Arguments

`out.dir`	Path to the folder where the waterfall plots of selected genes based on the specified q value of the KW results table will be added.
`alex.data`	output of the alex.prep.lsn.expr function. It's a list of three data tables that include "row.mtch", "alex.expr" with expression data, "alex.lsn" with lesion data. Rows of alex.expr, and "alex.lsn" matrices are ordered by gene ensembl IDs and columns are ordered by patient ID.
`alex.kw.results`	ALEX Kruskal-Wallis test results (output of the KW.hit.express function).
`q`	Maximum allowed KW q-value threshold for a gene to be plotted based on the output of the KW.hit.express function.
`lsn.data`	Lesion data in a GRIN compatible format.

Details

Function will return waterfall plots for top significant genes in the KW results table based on the user specified q-value threshold of the KW test.The plots will be added to the user specified outdir folder.

Value

Function will return waterfall plots for top significant genes in the KW results table.

Author(s)

Abdelrahman Elsayed <`abdelrahman.elsayed@stjude.org`> and Stanley Pounds <`stanley.pounds@stjude.org`>

References

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`alex.prep.lsn.expr()`, `KW.hit.express()`, `alex.waterfall.prep()`, `alex.waterfall.plot()`

Examples

```
## Not run:
data(expr.data)
data(lesion.data)
data(hg19.gene.annotation)

# prepare expression, lesion data and return the set of genes with both types of data available
# ordered by gene IDs in rows and patient IDs in columns:
alex.data=alex.prep.lsn.expr(expr.data, lesion.data,
                               hg19.gene.annotation, min.pts.expr=5, min.pts.lsn=5)

# run KW test for association between lesion groups and expression level of the same gene:
alex.kw.results=KW.hit.express(alex.data, hg19.gene.annotation, min.grp.size=5)

# return waterfall plots for a list of top significant genes to the specified folder:
top.genes.waterfall=top.alex.waterfall.plots("Path to the folder/waterfall.top.genes/",
                                        alex.data, alex.kw.results, 0.0000001, lesion.data)

## End(Not run)
```

`write.grin.xlsx` *Write GRIN Results*

Description

The function Write GRIN results to an excel file with multiple sheets that include GRIN results, lesion data, gene annotation data, chromosome size, gene-lesion overlap and methods paragraph.

Usage

```
write.grin.xlsx(grin.result, output.file)
```

Arguments

`grin.result`	output results of the grin.stats function.
`output.file`	output file name ".xlsx".

Value

This function return an excel file with seven sheets that include:

`gene.hits`	data table of GRIN results that include gene annotation, number of subjects affected by each lesion type for example gain, loss, mutation, etc.., and number of hits affecting each locus. The GRIN results table will also include P and FDR adjusted q-values showing the probability of each locus of being affected by one or a constellation of multiple types of lesions.
`gene.lsn.data`	each row represent a gene overlapped by a certain lesion. Column "gene" shows the overlapped gene ensembl ID and "ID"" column has the patient ID.

`lsn.data`	input lesion data
`gene.data`	input gene annotation data
`chr.size`	data table showing the size of the 22 autosomes, in addition to X and Y chromosomes in base pairs.
`interpretation`	provides some details about the content of each sheet in the output excel file and interpretation of each column in the "gene.hits" GRIN results table.
`method.paragraph`	include a paragraph that explains the GRIN model and cite some references.

Author(s)

Stanley Pounds <`stanley.pounds@stjude.org`>

References

Pounds, Stan, et al. (2013) A genomic random interval model for statistical analysis of genomic lesion data.

Cao, X., Elsayed, A. H., & Pounds, S. B. (2023). Statistical Methods Inspired by Challenges in Pediatric Cancer Multi-omics.

See Also

`grin.stats()`

Examples

```
## Not run:
data(lesion.data)

# to directly retreive gene annotation and chromosome size files from Ensembl BioMart database,
# UCSC genome browsers and run the GRIN analysis:
grin.results=grin.stats(lesion.data, genome.version="Human_GRCh37")
# users can replace Human_GRCh37 with Human_GRCh38 for hg38 genome assembly

# Write GRIN results using write.grin.xlsx function:
results.xlsx=write.grin.xlsx(grin.results, "T-ALL-GRIN-result.xlsx")

## End(Not run)
```

Index