# Package 'hpgltools'

February 10, 2021

**Type** Package

**Title** A pile of (hopefully) useful R functions

**Version** 1.0

**Date** 2018-03-01

**Author** Ashton Trey Belew, Keith Hughitt

**Maintainer** Ashton Trey Belew <abelew@gmail.com>

**Description** This is a set of functions I have been using in my various analyses
in the El-Sayed laboratory. The set of tasks included herein run a
spectrum from preprocessing count-tables from RNAseq-like data,
through differential expression analyses, to post-processing tasks
like gene ontology enrichment. Along the way, these function seek
to make plotting analyses consistent, provide multiple entry-points
to the various tools, and handle corner cases which are not
flexibly handled by the packages this is based upon.

**License** GPL-2 | file LICENSE

**Suggests** affy, AnnotationDbi, AnnotationForge, AnnotationHub,
BiocGenerics, BiocManager, biomaRt, Biostrings, BRAIN, BSgenome,
caret, Category, cleaver, clusterProfiler, corpcor, corrplot, curl,
DBI, desc, DESeq, DESeq2, devEMF, devtools, directlabels, doParallel,
DOSE, doSNOW, DSS,
EBSeq, EDASeq, edgeR, EuPathDB,
fastcluster, fastICA, ffpe, fission,
genbankr, genefilter, GenomicRanges, GenomeInfoDb, genoPlotR, ggdendro,
ggrepel, ggstatsplot, ggthemes, goseq, GO.db, GOstats,
graph, GSVA, GSVAdata, gtools, gplots, gProfileR, gprofiler2, GSEABase,
Heatplus, Hmisc, Homo.sapiens, htmlwidgets, httr,
IHW, inflection, IRanges, isva, iterators,
jsonlite,
KEGGREST, KEGGgraph,
lattice, limma, locfit,
matrixStats, miscTools, MLSeq, motifRG, MSnbase, mygene, mzR,
openxlsx, OrganismDbi,
pander, parallel, pasilla, pathview, pcaMethods, plotly, plyr, preprocessCore, PROPER,
qvalue,

R.utils, RColorBrewer, RCurl, readr, reactome.db, readODS, readxl, reshape2, rGA-
DEM, Rgraphviz,
rhdf5, rjson, rmarkdown, robust, robustbase, Rsamtools, RSQLite, Rtsne,
rtracklayer, ruv, RUVSeq, rvest,
S4Vectors, scales, SeqTools, seqLogo, SmartSVA, statmod, stringi, stringr, surv-
Jamda, SWATH2stats,
taxize, testthat, tidyr, topGO, tximport,
UniProt.ws, uwot,
xCell, xml2,
Vennerable, venneuler

**Imports** data.table, dplyr,
foreach,
ggplot2, GenomicFeatures, glue,
knitr,
magrittr, methods,
rlang,
sva,
variancePartition

**Depends** Biobase, R6

**VignetteBuilder** knitr

**ByteCompile** true

**biocViews** DifferentialExpression

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**Collate** 'hpgltools.r'
'annotation_biomart.r'
'annotation_genbank.r'
'annotation_gff.r'
'annotation_kegg.r'
'annotation_microbesonline.r'
'annotation_orgdb.r'
'annotation_txt.r'
'annotation_uniprot.r'
'annotation_shared.r'
'expt.r'
'de_shared.r'
'de_basic.r'
'de_edger.r'
'de_deseq.r'
'de_ebseq.r'
'de_limma.r'
'de_plots.r'
'de_xlsx.r'
'normalize_shared.r'
'normalize_filter.r'
'normalize_convert.r'

'normalize_transform.r'
'normalize_norm.r'
'normalize_batch.r'
'plot_shared.r'
'plot_bar.r'
'plot_distribution.r'
'plot_dotplot.r'
'plot_heatmap.r'
'plot_hist.r'
'plot_point.r'
'plot_venn.r'
'gsva.r'
'ontology_shared.r'
'ontology_goseq.r'
'ontology_gprofiler.r'
'ontology_gostats.r'
'helpers_misc.r'
'ontology_topgo.r'
'ontology_clusterprofiler.r'
'ontology_plots.r'
'ontology_xlsx.r'
'ontology_kegg.r'
'dimension_reduction.r'
'plot_circos.r'
'plot_genplot.r'
'plot_misc.r'
'model_testing.r'
'alt_splicing.r'
'model_varpartition.r'
'tnseq.r'
'proteomics.r'
'plot_proteomics.r'
'variants.r'
'sequence.r'
'power_estimation.r'
'xlsx.r'
'mlseq.r'
'motif.r'

# R **topics documented:**

---

%:::%                         *R CMD check is super annoying about :::.*

---

## Description

In a fit of pique, I did a google search to see if anyone else has been annoyed in the same way as was I. Yihui Xie was, and in his email to r-devel in 2013 he proposed a game of hide-and-seek; which I am repeating here.

## Usage

```
pkg %:::% fun
```

## Arguments

| | |
|---|---|
| pkg | on the left hand side |
| fun | on the right hand side |

## Details

This just implements ::: as an infix operator that will not trip check.

---

add_conditional_nas | *Replace 0 with NA if not all entries for a given condition are 0.*

---

### Description

This will hopefully handle a troubling corner case in Volker's data: He primarily wants to find proteins which are found in one condition, but _not_ in another. However, due to the unknown unknown problem in DIA acquisition, answering this question is difficult. If one uses a normal expressionset or msnset or whatever, one of two things will happen: either the 0/NA proteins will be entirely removed/ignored, or they will lead to spurious 'significant' calls. MSstats, to its credit, does a lot to try to handle these cases; but in the case Volker is most interested, it will exclude the interesting proteins entirely.

### Usage

```
add_conditional_nas(expt, fact = "condition", method = "NA")
```

### Arguments

expt        Expressionset to examine.

fact        Experimental design factor to use.

method      Specify whether to leave the NAs as NA, or replace them with the mean of all
            non-NA values.

### Details

So, here is what I am going to do: Iterate through each element of the chosen experimental design factor, check if all samples for that condition are 0, if so; leave them. If not all the samples have 0 for the given condition, then replace the zero entries with NA. This should allow for stuff like rowMeans(na.rm=TRUE) to provide useful information.

Finally, this will add columns to the annotations which tell the number of observations for each protein after doing this.

### Value

New expressionset with some, but not all, 0s replaced with NA.

---

| | |
|---|---|
| all_adjusters | *Combine all surrogate estimators and batch correctors into one function.* |

---

## Description

For a long time, I have mostly kept my surrogate estimators and batch correctors separate. However, that separation was not complete, and it really did not make sense. This function brings them together. This now contains all the logic from the freshly deprecated get_model_adjust().

## Usage

```
all_adjusters(
  input,
  design = NULL,
  estimate_type = "sva",
  batch1 = "batch",
  batch2 = NULL,
  surrogates = "be",
  low_to_zero = FALSE,
  cpus = 4,
  na_to_zero = TRUE,
  expt_state = NULL,
  confounders = NULL,
  chosen_surrogates = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| input | Dataframe or expt or whatever as the data to analyze/modify. |
| design | If the data is not an expt, then put the design here. |
| estimate_type | Name of the estimator. |
| batch1 | Column in the experimental design for the first known batch. |
| batch2 | Only used by the limma method, a second batch column. |
| surrogates | Either a number of surrogates or a method to search for them. |
| expt_state | If this is not an expt, provide the state of the data here. |
| confounders | List of confounded factors for smartSVA/iSVA. |
| ... | Extra arguments passed along to other methods. |

## Details

This applies the methodologies very nicely explained by Jeff Leek at https://github.com/jtleek/svaseq/blob/master/recount.Rm and attempts to use them to acquire estimates which may be applied to an experimental model by either EdgeR, DESeq2, or limma. In addition, it modifies the count tables using these estimates

so that one may play with the modified counts and view the changes (with PCA or heatmaps or whatever). Finally, it prints a couple of the plots shown by Leek in his document. In other words, this is entirely derivative of someone much smarter than me.

### Value

List containing surrogate estimates, new counts, the models, and some plots, as available.

---

all_ontology_searches     *Perform ontology searches given the results of a differential expression analysis.*

---

### Description

This takes a set of differential expression results, extracts a subset of up/down expressed genes; passes them to goseq, clusterProfiler, topGO, GOstats, and gProfiler; collects the outputs; and returns them in a (hopefully) consistent fashion. It attempts to handle the differing required annotation/GOid inputs required for each tool and/or provide supported species in ways which the various tools expect.

### Usage

```
all_ontology_searches(
  de_out,
  gene_lengths = NULL,
  goids = NULL,
  n = NULL,
  z = NULL,
  lfc = NULL,
  p = NULL,
  overwrite = FALSE,
  species = "unsupported",
  orgdb = "org.Dm.eg.db",
  goid_map = "reference/go/id2go.map",
  gff_file = NULL,
  gff_type = "gene",
  do_goseq = TRUE,
  do_cluster = TRUE,
  do_topgo = TRUE,
  do_gostats = TRUE,
  do_gprofiler = TRUE,
  do_trees = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `de_out` | List of topTables comprising limma/deseq/edger outputs. |
| `gene_lengths` | Data frame of gene lengths for goseq. |
| `goids` | Data frame of goids and genes. |
| `n` | Number of genes at the top/bottom of the fold-changes to define 'significant.' |
| `z` | Number of standard deviations from the mean fold-change used to define 'significant.' |
| `lfc` | Log fold-change used to define 'significant'. |
| `p` | Maximum pvalue to define 'significant.' |
| `overwrite` | Overwrite existing excel results file? |
| `species` | Supported organism used by the tools. |
| `orgdb` | Provide an organismDbi/Orgdb to hold the various annotation data, in response to the shift of clusterprofiler and friends towards using them. |
| `goid_map` | Mapping file used by topGO, if it does not exist then goids_df creates it. |
| `gff_file` | gff file containing the annotations used by gff2genetable from clusterprofiler. |
| `gff_type` | Column to use from the gff file for the universe of genes. |
| `do_goseq` | Perform simple_goseq()? |
| `do_cluster` | Perform simple_clusterprofiler()? |
| `do_topgo` | Perform simple_topgo()? |
| `do_gostats` | Perform simple_gostats()? |
| `do_gprofiler` | Perform simple_gprofiler()? |
| `do_trees` | make topGO trees from the data? |
| `...` | Arguments to pass through in arglist. |

## Value

a list of up/down ontology results from goseq/clusterprofiler/topgo/gostats, and associated trees.

## See Also

**goseq clusterProfiler topGO goStats gProfiler GO.db**

## Examples

```
## Not run:
 many_comparisons = limma_pairwise(expt=an_expt)
 tables = many_comparisons$limma
 this_takes_forever = limma_ontology(tables, gene_lengths=lengthdb,
                                  goids=goids_df, z=1.5, gff_file='length_db.gff')

## End(Not run)
```

---

all_pairwise                          *Perform limma, DESeq2, EdgeR pairwise analyses.*

---

**Description**

This takes an expt object, collects the set of all possible pairwise comparisons, sets up experimental models appropriate for the differential expression analyses, and performs them.

**Usage**

```
all_pairwise(
  input = NULL,
  conditions = NULL,
  batches = NULL,
  model_cond = TRUE,
  modify_p = FALSE,
  model_batch = TRUE,
  filter = NULL,
  model_intercept = FALSE,
  extra_contrasts = NULL,
  alt_model = NULL,
  libsize = NULL,
  test_pca = TRUE,
  annot_df = NULL,
  parallel = TRUE,
  do_basic = TRUE,
  do_deseq = TRUE,
  do_ebseq = NULL,
  do_edger = TRUE,
  do_limma = TRUE,
  convert = "cpm",
  norm = "quant",
  verbose = TRUE,
  ...
)
```

**Arguments**

| | |
|---|---|
| input | Dataframe/vector or expt class containing count tables, normalization state, etc. |
| conditions | Factor of conditions in the experiment. |
| batches | Factor of batches in the experiment. |
| model_cond | Include condition in the model? This is likely always true. |
| modify_p | Depending on how it is used, sva may require a modification of the p-values. |
| model_batch | Include batch in the model? This may be true/false/"sva" or other methods supported by all_adjusters(). |

| | |
|---|---|
| filter | Added because I am tired of needing to filter the data before invoking all_pairwise(). |
| model_intercept | |
| | Use an intercept model instead of cell means? |
| extra_contrasts | |
| | Optional extra contrasts beyone the pairwise comparisons. This can be pretty neat, lets say one has conditions A,B,C,D,E and wants to do (C/B)/A and (E/D)/A or (E/D)/(C/B) then use this with a string like: "c_vs_b_ctrla = (C-B)-A, e_vs_d_ctrla = (E-D)-A, de_vs_cb = (E-D)-(C-B)". |
| alt_model | Alternate model to use rather than just condition/batch. |
| libsize | Library size of the original data to help voom(). |
| test_pca | Perform some tests of the data before/after applying a given batch effect. |
| annot_df | Annotations to add to the result tables. |
| parallel | Use dopar to run limma, deseq, edger, and basic simultaneously. |
| do_basic | Perform a basic analysis? |
| do_deseq | Perform DESeq2 pairwise? |
| do_ebseq | Perform EBSeq (caveat, this is NULL as opposed to TRUE/FALSE so it can choose). |
| do_edger | Perform EdgeR? |
| do_limma | Perform limma? |
| convert | Modify the data with a 'conversion' method for PCA? |
| norm | Modify the data with a 'normalization' method for PCA? |
| ... | Picks up extra arguments into arglist, currently only passed to write_limma(). |

## Details

Tested in test_29de_shared.R This runs limma_pairwise(), deseq_pairwise(), edger_pairwise(), basic_pairwise() each in turn. It collects the results and does some simple comparisons among them.

## Value

A list of limma, deseq, edger results.

## See Also

**limma DESeq2 edgeR** link{limma_pairwise} deseq_pairwise edger_pairwise basic_pairwise

## Examples

```
## Not run:
 lotsodata <- all_pairwise(input=expt, model_batch="svaseq")
 summary(lotsodata)
 ## limma, edger, deseq, basic results; plots; and summaries.

## End(Not run)
```

---

backup_file                        *Make a backup of an existing file with n revisions, like VMS!*

---

### Description

Sometimes I just want to kick myself for overwriting important files and then I remember using VMS and wish modern computers were a little more like it.

### Usage

```
backup_file(backup_file, backups = 4)
```

### Arguments

backup_file        Filename to backup.

backups            How many revisions?

---

base_size                          *The following sets the ggplot2 default text size.*

---

### Description

The following sets the ggplot2 default text size.

### Usage

```
base_size
```

### Format

An object of class numeric of length 1.

---

basic_pairwise *The simplest possible differential expression method.*

---

**Description**

Perform a pairwise comparison among conditions which takes nothing into account. It _only_ takes
the conditions, a mean value/variance among them, divides by condition, and returns the result. No
fancy nomalizations, no statistical models, no nothing. It should be the very worst method possible.
But, it should also provide a baseline to compare the other tools against, they should all do better
than this, always.

**Usage**

```
basic_pairwise(
  input = NULL,
  design = NULL,
  conditions = NULL,
  batches = NULL,
  model_cond = TRUE,
  model_intercept = FALSE,
  alt_model = NULL,
  model_batch = FALSE,
  force = FALSE,
  fx = "mean",
  ...
)
```

**Arguments**

| | |
|---|---|
| input | Count table by sample. |
| design | Data frame of samples and conditions. |
| conditions | Not currently used, but passed from all_pairwise() |
| batches | Not currently used, but passed from all_pairwise() |
| model_cond | Not currently used, but passed from all_pairwise() |
| model_intercept | |
| | Not currently used, but passed from all_pairwise() |
| alt_model | Not currently used, but passed from all_pairwise() |
| model_batch | Not currently used, but passed from all_pairwise() |
| force | Force as input non-normalized data? |
| fx | What function to use for mean/median? |
| ... | Extra options passed to arglist. |

**Details**

Tested in test_27de_basic.R This function was written after the corresponding functions in de_deseq.R, de_edger.R, and de_limma.R. Like those, it performs the full set of pairwise comparisons and returns a list of the results. As mentioned above, unlike those, it is purposefully stupid.

**Value**

Df of pseudo-logFC, p-values, numerators, and denominators.

**See Also**

**limma DESeq2 edgeR**

**Examples**

```
## Not run:
 expt <- create_expt(metadata="sample_sheet.xlsx", gene_info="annotations")
 basic_de <- basic_pairwise(expt)
 basic_tables <- combine_de_tables(basic_de)

## End(Not run)
```

---

batch_counts                    *Perform different batch corrections using limma, sva, ruvg, and cbcb-*
                                *SEQ.*

---

**Description**

I found this note which is the clearest explanation of what happens with batch effect data: https://support.bioconductor.org/p/7 Just to be clear, there's an important difference between removing a batch effect and modelling a batch effect. Including the batch in your design formula will model the batch effect in the regression step, which means that the raw data are not modified (so the batch effect is not removed), but instead the regression will estimate the size of the batch effect and subtract it out when performing all other tests. In addition, the model's residual degrees of freedom will be reduced appropriately to reflect the fact that some degrees of freedom were "spent" modelling the batch effects. This is the preferred approach for any method that is capable of using it (this includes DESeq2). You would only remove the batch effect (e.g. using limma's removeBatchEffect function) if you were going to do some kind of downstream analysis that can't model the batch effects, such as training a classifier. I don't have experience with ComBat, but I would expect that you run it on log-transformed CPM values, while DESeq2 expects raw counts as input. I couldn't tell you how to properly use the two methods together.

**Usage**

```
batch_counts(
  count_table,
  method = TRUE,
  design = NULL,
```

```
    batch1 = "batch",
    current_state = NULL,
    current_design = NULL,
    expt_state = NULL,
    surrogate_method = NULL,
    num_surrogates = NULL,
    low_to_zero = FALSE,
    cpus = 4,
    batch2 = NULL,
    noscale = TRUE,
    ...
)
```

## Arguments

| | |
|---|---|
| count_table | Matrix of (pseudo)counts. |
| design | Model matrix defining the experimental conditions/batches/etc. |
| batch1 | Column in the design table describing the presumed covariant to remove. |
| expt_state | Current state of the expt in an attempt to avoid double-normalization. |
| batch2 | Column in the design table describing the second covariant to remove (only used by limma at the moment). |
| noscale | Used for combatmod, when true it removes the scaling parameter from the invocation of the modified combat. |
| ... | More options for you! |
| batch | String describing the method to try to remove the batch effect (or FALSE to leave it alone, TRUE uses limma). |

## Value

The 'batch corrected' count table and new library size. Please remember that the library size which comes out of this may not be what you want for voom/limma and would therefore lead to spurious differential expression values.

## See Also

**limma edgeR RUVSeq sva cbcbSEQ**

## Examples

```
## Not run:
 limma_batch <- batch_counts(table, design, batch1='batch', batch2='strain')
 sva_batch <- batch_counts(table, design, batch='sva')

## End(Not run)
```

---

bioc_all                          *Grab a copy of all bioconductor packages and install them by type*

---

## Description

This uses jsonlite to get a copy of all bioconductor packages by name and then iterates through them
with BiocManager to install all of them. It performs a sleep between each installation in an attempt
to avoid being obnoxious. As a result, it will of a necessity take forever.

## Usage

```
bioc_all(
  release = NULL,
  mirror = "bioconductor.statistik.tu-dortmund.de",
  base = "packages",
  type = "software",
  suppress_updates = TRUE,
  suppress_auto = TRUE,
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| release | Bioconductor release to use, should probably be adjusted to automatically find it. |
| mirror | Bioconductor mirror to use. |
| base | Base directory on the mirror to download from. |
| type | Type in the tree to use (software or annotation) |
| suppress_updates | |
| | For BiocLite(), don't update? |
| suppress_auto | For BiocLite(), don't update? |
| force | Install if already installed? |

## Value

a number of packages installed

## See Also

**BiocManager**

## Examples

```
## Not run:
 go_get_some_coffee_this_will_take_a_while <- bioc_all()

## End(Not run)
```

---

cbcb_batch *A function suggested by Hector Corrada Bravo and Kwame Okrah for batch removal.*

---

## Description

During a lab meeting, the following function was suggested as a quick and dirty batch removal tool. It takes data and a model including a 'batch' factor, invokes limma on them, removes the batch factor, does a cross product of the fitted data and modified model and uses that with residuals to get a new data set.

## Usage

```
cbcb_batch(
  normalized_counts,
  model,
  batch1 = "batch",
  condition = "condition",
  matrix_scale = "linear",
  return_scale = "linear",
  method = "subtract"
)
```

## Arguments

normalized_counts

Data frame of log2cpm counts.

model          Balanced experimental model containing condition and batch factors.

batch1         Column containing the first batch's metadata in the experimental design.

condition      Column containing the condition information in the metadata.

matrix_scale   Is the data on a linear or log scale?

return_scale   Do you want the data returned on the linear or log scale?

method         I found a couple ways to apply the surrogates to the data. One method subtracts the residuals of a batch model, the other adds the conditional.

## Value

Dataframe of residuals after subtracting batch from the model.

## See Also

**limma** `voom` `lmFit`

## Examples

```
## Not run:
 newdata <- cbcb_batch_effect(counts, expt_model)

## End(Not run)
```

---

cbcb_combat                    *A modified version of comBatMod.*

---

### Description

This is a hack of Kwame Okrah's combatMod to make it not fail on corner-cases. This was mostly copy/pasted from https://github.com/kokrah/cbcbSEQ/blob/master/R/transform.R

### Usage

```
cbcb_combat(dat, batch, mod, noscale = TRUE, prior.plots = FALSE, ...)
```

### Arguments

| | |
|---|---|
| dat | Df to modify. |
| batch | Factor of batches. |
| mod | Factor of conditions. |
| noscale | The normal 'scale' option squishes the data too much, so this defaults to TRUE. |
| prior.plots | Print out prior plots? |
| ... | Extra options are passed to arglist |

### Value

Df of batch corrected data

### See Also

**sva** `ComBat`

### Examples

```
## Not run:
 df_new = cbcb_combat(df, batches, model)

## End(Not run)
```

---

cbcb_filter_counts *Filter low-count genes from a data set using cpm data and a threshold.*

---

### Description

This was a function written by Kwame Okrah and perhaps also Laura Dillon to remove low-count genes. It drops genes based on a cpm threshold and number of samples.

### Usage

```
cbcb_filter_counts(count_table, threshold = 1, min_samples = 2, libsize = NULL)
```

### Arguments

| | |
|---|---|
| count_table | Data frame of (pseudo)counts by sample. |
| threshold | Lower threshold of counts for each gene. |
| min_samples | Minimum number of samples. |
| libsize | Table of library sizes. |

### Value

Dataframe of counts without the low-count genes.

### See Also

**edgeR**

### Examples

```
## Not run:
 filtered_table <- cbcb_filter_counts(count_table)

## End(Not run)
```

---

check_plot_scale *Look at the range of the data for a plot and use it to suggest if a plot should be on log scale.*

---

### Description

There are a bunch of plots which often-but-not-always benefit from being displayed on a log scale rather than base 10. This is a quick and dirty heuristic which suggests the appropriate scale. If the data 'should' be on the log scale and it has 0s, then they are moved to 1 so that when logged they will return to 0. Similarly, if there are negative numbers and the intended scale is log, then this will set values less than 0 to zero to avoid imaginary numbers.

## Usage

```
check_plot_scale(data, scale = NULL, max_data = 10000, min_data = 10)
```

## Arguments

| | |
|---|---|
| data | Data to plot. |
| scale | If known, this will be used to define what (if any) values to change. |
| max_data | Define the upper limit for the heuristic. |
| min_data | Define the lower limit for the heuristic. |

---

choose_basic_dataset    *Attempt to ensure that input data to basic_pairwise() is suitable.*

---

## Description

basic_pairwise() assumes log2 data as input, use this to ensure that is true.

## Usage

```
choose_basic_dataset(input, force = FALSE, ...)
```

## Arguments

| | |
|---|---|
| input | An expressionset containing expt to test and/or modify. |
| force | If we want to try out other distributed data sets, force it in using me. |
| ... | future options, I think currently unused. |

## Value

data ready for basic_pairwise()

## See Also

**Biobase**

## Examples

```
## Not run:
 ready <- choose_basic_dataset(expt)

## End(Not run)
```

---

choose_binom_dataset *A sanity check that a given set of data is suitable for methods which assume a negative binomial distribution of input.*

---

### Description

Take an expt and poke at it to ensure that it will not result in troubled results.

### Usage

```
choose_binom_dataset(input, verbose = TRUE, force = FALSE, ...)
```

### Arguments

| | |
|---|---|
| input | Expressionset containing expt object. |
| force | Ignore every warning and just use this data. |
| ... | Extra arguments passed to arglist. |

### Details

Invoked by deseq_pairwise() and edger_pairwise().

### Value

dataset suitable for limma analysis

### See Also

**DESeq2 edgeR**

---

choose_dataset *Choose a suitable data set for Edger/DESeq*

---

### Description

The _pairwise family of functions all demand data in specific formats. This tries to make that consistent.

### Usage

```
choose_dataset(input, choose_for = "limma", force = FALSE, verbose = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `input` | Expt input. |
| `choose_for` | One of limma, deseq, edger, or basic. Defines the requested data state. |
| `force` | Force non-standard data? |
| `...` | More options for future expansion. |

## Details

Invoked by _pairwise().

## Value

List the data, conditions, and batches in the data.

## See Also

[choose_binom_dataset](#) [choose_limma_dataset](#) [choose_basic_dataset](#)

## Examples

```
## Not run:
 starting_data <- create_expt(metadata)
 modified_data <- normalize_expt(starting_data, transform="log2", norm="quant")
 a_dataset <- choose_dataset(modified_data, choose_for="deseq")
 ## choose_dataset should see that log2 data is inappropriate for DESeq2 and
 ## return it to a base10 state.

## End(Not run)
```

---

choose_limma_dataset     *A sanity check that a given set of data is suitable for analysis by limma.*

---

## Description

Take an expt and poke at it to ensure that it will not result in troubled limma results.

## Usage

```
choose_limma_dataset(
  input,
  force = FALSE,
  which_voom = "limma",
  verbose = TRUE,
  ...
)
```

**Arguments**

| | |
|---|---|
| `input` | Expressionset containing expt object. |
| `force` | Ingore warnings and use the provided data asis. |
| `which_voom` | Choose between limma'svoom, voomWithQualityWeights, or the hpgl equivalents. |
| `...` | Extra arguments passed to arglist. |

**Value**

dataset suitable for limma analysis

**See Also**

**limma**

---

| | |
|---|---|
| `choose_model` | *Try out a few experimental models and return a likely working option.* |

---

**Description**

The _pairwise family of functions all demand an experimental model. This tries to choose a consistent and useful model for all for them. This does not try to do multi-factor, interacting, nor dependent variable models, if you want those do them yourself and pass them off as alt_model.

**Usage**

```
choose_model(
  input,
  conditions = NULL,
  batches = NULL,
  model_batch = TRUE,
  model_cond = TRUE,
  model_intercept = FALSE,
  alt_model = NULL,
  alt_string = NULL,
  intercept = 0,
  reverse = FALSE,
  contr = NULL,
  surrogates = "be",
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `input` | Input data used to make the model. |
| `conditions` | Factor of conditions in the putative model. |
| `batches` | Factor of batches in the putative model. |
| `model_batch` | Try to include batch in the model? |
| `model_cond` | Try to include condition in the model? (Yes!) |
| `model_intercept` | |
| | Use an intercept model instead of cell-means? |
| `alt_model` | Use your own model. |
| `alt_string` | String describing an alternate model. |
| `intercept` | Choose an intercept for the model as opposed to 0. |
| `reverse` | Reverse condition/batch in the model? This shouldn't/doesn't matter but I wanted to test. |
| `contr` | List of contrasts.arg possibilities. |
| `surrogates` | Number of or method used to choose the number of surrogate variables. |
| `...` | Further options are passed to arglist. |

## Details

Invoked by the _pairwise() functions.

## Value

List including a model matrix and strings describing cell-means and intercept models.

## See Also

**stats** `model.matrix`

## Examples

```
## Not run:
 a_model <- choose_model(expt, model_batch=TRUE, model_intercept=FALSE)
 a_model$chosen_model
 ## ~ 0 + condition + batch

## End(Not run)
```

---

circos_arc                    *Write arcs between chromosomes in circos.*

---

### Description

Ok, so when I said I only do 1 chromosome images, I lied. This function tries to make writing arcs between chromosomes easier. It too works in 3 stages, It writes out a data file using cfgout as a basename and the data from df in the circos arc format into circos/data/bob_arc.txt It then writes out a configuration plot stanza in circos/conf/bob_arc.conf and finally adds an include to circos/bob.conf

### Usage

```
circos_arc(
  cfg,
  df,
  first_col = "seqnames",
  second_col = "seqnames.2",
  color = "blue",
  radius = 0.75,
  thickness = 3,
  ribbon = "yes",
  show = "yes",
  z = "0"
)
```

### Arguments

| | |
|---|---|
| cfg | Result of circos_prefix(), contains a bunch of useful material. |
| df | Dataframe with starts/ends and the floating point information. |
| first_col | Name of the first chromosome. |
| second_col | Name of the second chromosome. |
| color | Color of the chromosomes. |
| radius | Outer radius at which to add the arcs. |
| thickness | Integer thickness of the arcs. |
| ribbon | Print as a ribbon? |
| show | Show these arcs? |
| z | Correction parameter. |

### Details

In its current implementation, this only understands two chromosomes. A minimal amount of logic and data organization will address this weakness.

**Value**

The file to which the arc configuration information was written.

---

circos_heatmap *Write tiles of arbitrary heat-mappable data in circos.*

---

**Description**

This function tries to make the writing circos heatmaps easier. Like circos_plus_minus() and circos_hist() it works in 3 stages, It writes out a data file using cfgout as a basename and the data from df in the circos histogram format into circos/data/bob_heatmap.txt It then writes out a configuration plot stanza in circos/conf/bob_heatmap.conf and finally adds an include to circos/bob.conf

**Usage**

```
circos_heatmap(
  cfg,
  df,
  colname = "logFC",
  color_mapping = 0,
  min_value = NULL,
  max_value = NULL,
  basename = "",
  colors = NULL,
  color_choice = "spectral-9-div",
  scale_log_base = 1,
  outer = 0.9,
  rules = NULL,
  width = 0.08,
  spacing = 0.02
)
```

**Arguments**

| | |
|---|---|
| cfg | Result of circos_prefix(), contains a bunch of useful material. |
| df | Dataframe with starts/ends and the floating point information. |
| colname | Name of the column with the data of interest. |
| color_mapping | 0 means no overflows for min/max, 1 means overflows of min get a chosen color, 2 means overflows of both min/max get chosen colors. |
| min_value | Minimum value for the data. |
| max_value | Maximum value for the data. |
| basename | Make sure the written configuration files get different names with this. |
| colors | Colors of the heat map. |
| color_choice | Name of the heatmap to use, I forget how this interacts with color... |

| scale_log_base | Defines how the range of colors will be ranged with respect to the values in the data. |
|---|---|
| outer | Floating point radius of the circle into which to place the heatmap. |
| rules | some extra rules? |
| width | Width of each tile in the heatmap. |
| spacing | Radial distance between outer, inner, and inner to whatever follows. |

## Value

Radius after adding the histogram and the spacing.

---

| circos_hist | *Write histograms of arbitrary floating point data in circos.* |
|---|---|

---

## Description

This function tries to make the writing of histogram data in circos easier. Like circos_plus_minus() it works in 3 stages, It writes out a data file using cfgout as a basename and the data from df in the circos histogram format into circos/data/bob_hist.txt It then writes out a configuration plot stanza in circos/conf/bob_hist.conf and finally adds an include to circos/bob.conf

## Usage

```
circos_hist(
  cfg,
  df,
  colname = "logFC",
  basename = "",
  color = "blue",
  fill_color = "blue",
  fill_under = "yes",
  extend_bin = "no",
  thickness = "0",
  orientation = "out",
  outer = 0.9,
  width = 0.08,
  spacing = 0
)
```

## Arguments

| cfg | Result of circos_prefix(), contains a bunch of useful material. |
|---|---|
| df | Dataframe with starts/ends and the floating point information. |
| colname | Name of the column with the data of interest. |
| basename | Location to write the circos data (usually cwd). |

| color | Color of the plotted data. |
|---|---|
| fill_color | Guess |
| fill_under | The circos histogram fill under parameter |
| extend_bin | Extend bins? |
| thickness | histogram thickness. |
| orientation | facing in or out? |
| outer | Floating point radius of the circle into which to place the data. |
| width | Radial width of each tile. |
| spacing | Distance between outer, inner, and inner to whatever follows. |

### Value

Radius after adding the histogram and the spacing.

---

circos_ideogram            *Create the description of chromosome markings.*

---

### Description

This function writes ideogram files for circos.

### Usage

```
circos_ideogram(
  name = "default",
  conf_dir = "circos/conf",
  band_url = NULL,
  fill = "yes",
  stroke_color = "black",
  show_bands = "yes",
  fill_bands = "yes",
  thickness = "20",
  stroke_thickness = "2",
  label_font = "condensedbold",
  spacing_default = "0",
  spacing_break = "0",
  fill_color = "black",
  radius = "0.85",
  radius_padding = "0.05",
  label_size = "36",
  band_stroke_thickness = "2"
)
```

## Arguments

| | |
|---|---|
| name | Name of the configuration file to which to add the ideogram. |
| conf_dir | Where does the configuration live? |
| band_url | Provide a url for making these imagemaps? |
| fill | Fill in the strokes? |
| stroke_color | What color? |
| show_bands | Show the bands for the ideogram? |
| fill_bands | and fill them in? |
| thickness | How thick to color the lines |
| stroke_thickness | |
| | How much of them to fill in |
| label_font | What font to use. |
| spacing_default | |
| | How much space between elements. |
| spacing_break | Space between breaks. |
| fill_color | What color to fill |
| radius | Where on the circle to put them |
| radius_padding | How much to pad between radii. |
| label_size | How large to make the labels in px. |
| band_stroke_thickness | |
| | How big to make the strokes! |

## Value

The file to which the ideogram configuration was written.

---

| | |
|---|---|
| circos_karyotype | *Create the description of (a)chromosome(s) for circos.* |

---

## Description

This function tries to save me from having to get the lengths of arcs for bacterial chromosomes manually correct, and writes them as a circos compatible karyotype file. The outfile parameter was chosen to match the configuration directive outlined in circos_prefix(), however that will need to be changed in order for this to work in variable conditions. Next time I make one of these graphs I will do that I suspect. In addition, this currently only understands how to write bacterial chromosomes, that will likely be fixed when I am asked to write out a L.major karyotype. These defaults were chosen because I have a chromosome of this length that is correct.

## Usage

```
circos_karyotype(
  cfg,
  segments = 6,
  color = "white",
  fasta = NULL,
  lengths = NULL
)
```

## Arguments

| | |
|---|---|
| `cfg` | Result from circos_prefix(), contains a bunch of useful things. |
| `segments` | How many segments to cut the chromosome into? |
| `color` | Color segments of the chromosomal arc? |
| `fasta` | Fasta file to use to create the karyotype. |
| `lengths` | If no sequence file is provided, use a named numeric vector to provide them. |

## Value

The output filename.

---

| `circos_make` | *Write a simple makefile for circos.* |
|---|---|

---

## Description

I regenerate all my circos pictures with make(1). This is my makefile.

## Usage

```
circos_make(cfg, target = "", circos = "circos", verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `cfg` | Configuration from circos_prefix(). |
| `target` | Default make target. |
| `circos` | Location of circos. I have a copy in home/bin/circos and use that sometimes. |

## Value

a kitten

---

circos_plus_minus    *Write tiles of bacterial ontology groups using the categories from microbesonline.org.*

---

### Description

This function tries to save me from writing out ontology definitions and likely making mistakes. It uses the start/ends from the gff annotation along with the 1 letter GO-like categories from microbesonline.org. It then writes two data files circos/data/bob_plus_go.txt, circos/data/bob_minus_go.txt along with two configuration files circos/conf/bob_minus_go.conf and circos/conf/bob_plus_go.conf and finally adds an include to circos/bob.conf

### Usage

```
circos_plus_minus(
  cfg,
  outer = 1,
  width = 0.08,
  thickness = 95,
  spacing = 0,
  padding = 1,
  margin = 0,
  plus_orientation = "out",
  minus_orientation = "in",
  layers = 1,
  layers_overflow = "hide",
  acol = "orange",
  bcol = "reds-9-seq",
  ccol = "yellow",
  dcol = "vlpurple",
  ecol = "vlgreen",
  fcol = "dpblue",
  gcol = "vlgreen",
  hcol = "vlpblue",
  icol = "vvdpgreen",
  jcol = "dpred",
  kcol = "orange",
  lcol = "vvlorange",
  mcol = "dpgreen",
  ncol = "vvlpblue",
  ocol = "vvlgreen",
  pcol = "vvdpred",
  qcol = "ylgn-3-seq",
  rcol = "vlgrey",
  scol = "grey",
  tcol = "vlpurple",
  ucol = "greens-3-seq",
```

```
    vcol = "vlred",
    wcol = "vvdppurple",
    xcol = "black",
    ycol = "lred",
    zcol = "vlpblue"
)
```

## Arguments

| | |
|---|---|
| cfg | Result from circos_prefix(). |
| outer | Floating point radius of the circle into which to place the plus-strand data. |
| width | Radial width of each tile. |
| thickness | How wide to make the bars. |
| spacing | Radial distance between outer, inner, and inner to whatever follows. |
| padding | How much space between them. |
| margin | Margin between elements. |
| plus_orientation | |
| | Orientation of the plus pieces. |
| minus_orientation | |
| | Orientation of the minus pieces. |
| layers | How many layers to use |
| layers_overflow | |
| | How to handle too many layers. |
| acol | A color: RNA processing and modification. |
| bcol | B color: Chromatin structure and dynamics. |
| ccol | C color: Energy production conversion. |
| dcol | D color: Cell cycle control, mitosis and meiosis. |
| ecol | E color: Amino acid transport metabolism. |
| fcol | F color: Nucleotide transport and metabolism. |
| gcol | G color: Carbohydrate transport and metabolism. |
| hcol | H color: Coenzyme transport and metabolism. |
| icol | I color: Lipid transport and metabolism. |
| jcol | J color: Translation, ribosome structure and biogenesis. |
| kcol | K color: Transcription. |
| lcol | L color: Replication, recombination, and repair. |
| mcol | M color: Cell wall/membrane biogenesis. |
| ncol | N color: Cell motility |
| ocol | O color: Posttranslational modification, protein turnover, chaperones. |
| pcol | P color: Inorganic ion transport and metabolism. |
| qcol | Q color: Secondary metabolite biosynthesis, transport, and catabolism. |

| rcol | R color: General function prediction only. |
|---|---|
| scol | S color: Function unknown. |
| tcol | T color: Signal transduction mechanisms. |
| ucol | U color: Intracellular trafficking(sp?) and secretion. |
| vcol | V color: Defense mechanisms. |
| wcol | W color: Extracellular structures. |
| xcol | X color: Not in COG. |
| ycol | Y color: Nuclear structure. |
| zcol | Z color: Cytoskeleton. |

## Value

Radius after adding the plus/minus information and the spacing between them.

---

circos_prefix  *Write the beginning of a circos configuration file.*

---

## Description

A few parameters need to be set when starting circos. This sets some of them and gets ready for plot stanzas.

## Usage

```
circos_prefix(
  annotation,
  name = "mgas",
  basedir = "circos",
  chr_column = "seqnames",
  cog_column = "COGFun",
  start_column = "start",
  stop_column = "end",
  strand_column = "strand",
  id_column = NULL,
  cog_map = NULL,
  radius = 1800,
  chr_units = 1000,
  band_url = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| annotation | Annotation data frame. |
| name | Name of the map, called with 'make name'. |
| basedir | Base directory for writing the data. |
| chr_column | Name of the column containing the chromosome names in the annotations. |
| cog_column | Name of the column containing the COG groups in the annotations. |
| start_column | Name of the column containing the starts in the annotations. |
| stop_column | Name of the column containing the stops in the annotations. |
| strand_column | Name of the column containing the strand information. |
| id_column | Where do the gene IDs live? NULL means rownames. |
| cog_map | Not yet used, but used to provide an alternate map of groups/colors. |
| radius | Size of the image. |
| chr_units | How often to print chromosome in 'prefix' units. |
| band_url | Place to imagemap link. |
| ... | Extra arguments passed to the tick/karyotype makers. |

## Details

In its current implementation, this really assumes that there will be no highlight stanzas and at most 1 link stanza. chromosomes. A minimal amount of logic and data organization will address these weaknesses.

## Value

The master configuration file name.

---

| circos_suffix | *Write the end of a circos master configuration.* |
|---|---|

---

## Description

circos configuration files need an ending. This writes it.

## Usage

```
circos_suffix(cfg)
```

## Arguments

| | |
|---|---|
| cfg | Result from circos_prefix() |

## Value

The filename of the configuration.

---

circos_ticks *Create the ticks for a circos plot.*

---

### Description

This function writes ticks for circos. This has lots of options, the defaults are all taken from the circos example documentation for a bacterial genome.

### Usage

```
circos_ticks(
  name = "default",
  conf_dir = "circos/conf",
  show_ticks = "yes",
  show_tick_labels = "yes",
  show_grid = "no",
  skip_first_label = "yes",
  skip_last_label = "no",
  tick_separation = 2,
  min_label_distance = 0,
  label_separation = 5,
  label_offset = 5,
  label_size = 8,
  multiplier = 0.001,
  main_color = "black",
  main_thickness = 3,
  main_size = 20,
  first_size = 10,
  first_spacing = 1,
  first_color = "black",
  first_show_label = "no",
  first_label_size = 12,
  second_size = 15,
  second_spacing = 5,
  second_color = "black",
  second_show_label = "yes",
  second_label_size = 16,
  third_size = 18,
  third_spacing = 10,
  third_color = "black",
  third_show_label = "yes",
  third_label_size = 16,
  fourth_spacing = 100,
  fourth_color = "black",
  fourth_show_label = "yes",
  suffix = " kb",
  fourth_label_size = 36,
```

```
    include_first_label = TRUE,
    include_second_label = TRUE,
    include_third_label = TRUE,
    include_fourth_label = TRUE,
    ...
)
```

## Arguments

| | |
|---|---|
| name | Name of the configuration file to which to add the ideogram. |
| conf_dir | Where does the configuration live. |
| show_ticks | Show them or not. |
| show_tick_labels | Show the tick labels, or do not. |
| show_grid | Print a grid behind. |
| skip_first_label | Like a clock. |
| skip_last_label | Ditto. |
| tick_separation | Top-level separation between tick marks. |
| min_label_distance | distance to the edge of the plot for labels. |
| label_separation | radial distance between labels. |
| label_offset | The offset for the labels. |
| label_size | Top-level label size. |
| multiplier | When writing the position, by what factor to lower the numbers? |
| main_color | Color for top-level labels? |
| main_thickness | Top-level thickness of lines etc. |
| main_size | Top-level size of text. |
| first_size | Second level size of text. |
| first_spacing | Second level spacing of ticks. |
| first_color | Second-level text color. |
| first_show_label | Show a label for the second level ticks? |
| first_label_size | Text size for second level labels? |
| second_size | Size of ticks for the third level. |
| second_spacing | third-level spacing |
| second_color | Text color for the third level. |
| second_show_label | Give them a label? |

second_label_size
                    And a size.

third_size          Now for the size of the almost-largest ticks

third_spacing       How far apart?

third_color         and their color

third_show_label
                    give a label?

third_label_size
                    and a size.

fourth_spacing      The largest ticks!

fourth_color        The largest color.

fourth_show_label
                    Provide a label?

suffix              String for printing chromosome distances.

fourth_label_size
                    They are big!

include_first_label
                    Provide the smallest labels?

include_second_label
                    Second smallest labels?

include_third_label
                    Second biggest labels?

include_fourth_label
                    Largest labels?

...                 Extra arguments from circos_prefix().

## Value

The file to which the ideogram configuration was written.

---

circos_tile                 *Write tiles of arbitrary categorical point data in circos.*

---

## Description

This function tries to make the writing circos tiles easier. Like circos_plus_minus() and circos_hist() it works in 3 stages, It writes out a data file using cfgout as a basename and the data from df in the circos histogram format into circos/data/bob_tile.txt It then writes out a configuration plot stanza in circos/conf/bob_tile.conf and finally adds an include to circos/bob.conf

**Usage**

```
circos_tile(
  cfg,
  df,
  colname = "logFC",
  basename = "",
  colors = NULL,
  thickness = 80,
  padding = 1,
  margin = 0,
  stroke_thickness = 0,
  orientation = "out",
  outer = 0.9,
  width = 0.08,
  spacing = 0
)
```

**Arguments**

| | |
|---|---|
| cfg | Result from circos_prefix(). |
| df | Dataframe with starts/ends and the floating point information. |
| colname | Name of the column with the data of interest. chromosome) |
| basename | Used to make unique filenames for the data/conf files. |
| colors | Colors of the data. |
| thickness | How thick to make the tiles in radial units. |
| padding | Space between tiles. |
| margin | How much space between other rings and the tiles? |
| stroke_thickness | |
| | Size of the tile outlines. |
| orientation | Facing in or out. |
| outer | Floating point radius of the circle into which to place the categorical data. |
| width | Width of each tile. |
| spacing | Radial distance between outer, inner, and inner to whatever follows. |

**Value**

Radius after adding the histogram and the spacing.

| clear_session | *Clear an R session, this is probably unwise given what I have read about R.* |
|---|---|

### Description

Clear an R session, this is probably unwise given what I have read about R.

### Usage

```
clear_session(keepers = NULL, depth = 10)
```

### Arguments

| | |
|---|---|
| keepers | List of namespaces to leave alone (unimplemented). |
| depth | Cheesy forloop of attempts to remove packages stops after this many tries. |

### Value

A spring-fresh R session, hopefully.

| cleavage_histogram | *Make a histogram of how many peptides are expected at every integer dalton from a given start to end size for a given enzyme digestion.* |
|---|---|

### Description

This is very similar to plot_cleaved() above, but tries to be a little bit smarter.

### Usage

```
cleavage_histogram(
  pep_sequences,
  enzyme = "trypsin",
  start = 600,
  end = 1500,
  color = "black"
)
```

### Arguments

| | |
|---|---|
| pep_sequences | Protein sequences as per plot_cleaved(). |
| enzyme | Compatible enzyme name from cleaver. |
| start | Print histogram from here |
| end | to here. |
| color | Make the bars this color. |

**Value**

List containing the plot and size distribution.

---

cluster_trees *Take clusterprofile group data and print it on a tree as per topGO.*

---

**Description**

TopGO's ontology trees can be very illustrative. This function shoe-horns clusterProfiler data into the format expected by topGO and uses it to make those trees.

**Usage**

```
cluster_trees(
  de_genes,
  cpdata,
  goid_map = "id2go.map",
  go_db = NULL,
  score_limit = 0.2,
  overwrite = FALSE,
  selector = "topDiffGenes",
  pval_column = "adj.P.Val"
)
```

**Arguments**

| | |
|---|---|
| de_genes | List of genes deemed 'interesting'. |
| cpdata | Data from simple_clusterprofiler(). |
| goid_map | Mapping file of IDs to GO ontologies. |
| go_db | Dataframe of mappings used to build goid_map. |
| score_limit | Scoring limit above which to ignore genes. |
| overwrite | Overwrite an existing goid mapping file? |
| selector | Name of a function for applying scores to the trees. |
| pval_column | Name of the column in the GO table from which to extract scores. |

**Value**

plots! Trees! oh my!

**See Also**

**Ramigo** `showSigOfNodes`

**Examples**

```
## Not run:
 cluster_data <- simple_clusterprofiler(genes, stuff)
 ctrees <- cluster_trees(genes, cluster_data)

## End(Not run)
```

---

combine_de_tables          *Combine portions of deseq/limma/edger table output.*

---

**Description**

This hopefully makes it easy to compare the outputs from limma/DESeq2/EdgeR on a table-by-table basis.

**Usage**

```
combine_de_tables(
  apr,
  extra_annot = NULL,
  excel = NULL,
  sig_excel = NULL,
  abundant_excel = NULL,
  excel_title = "Table SXXX: Combined Differential Expression of YYY",
  keepers = "all",
  excludes = NULL,
  adjp = TRUE,
  include_limma = TRUE,
  include_deseq = TRUE,
  include_edger = TRUE,
  include_ebseq = TRUE,
  include_basic = TRUE,
  rownames = TRUE,
  add_plots = TRUE,
  loess = FALSE,
  plot_dim = 6,
  compare_plots = TRUE,
  padj_type = "ihw",
  lfc_cutoff = 1,
  p_cutoff = 0.05,
  de_types = c("limma", "deseq", "edger"),
  ...
)
```

## Arguments

| | |
|---|---|
| `apr` | Output from all_pairwise(). |
| `extra_annot` | Add some annotation information? |
| `excel` | Filename for the excel workbook, or null if not printed. |
| `sig_excel` | Filename for writing significant tables. |
| `abundant_excel` | Filename for writing abundance tables. |
| `excel_title` | Title for the excel sheet(s). If it has the string 'YYY', that will be replaced by the contrast name. |
| `keepers` | List of reformatted table names to explicitly keep certain contrasts in specific orders and orientations. |
| `excludes` | List of columns and patterns to use for excluding genes. |
| `adjp` | Perhaps you do not want the adjusted p-values for plotting? |
| `include_limma` | Include limma analyses in the table? |
| `include_deseq` | Include deseq analyses in the table? |
| `include_edger` | Include edger analyses in the table? |
| `include_ebseq` | Include ebseq analyses in the table? |
| `include_basic` | Include my stupid basic logFC tables? |
| `rownames` | Add rownames to the xlsx printed table? |
| `add_plots` | Add plots to the end of the sheets with expression values? |
| `loess` | Add time intensive loess estimation to plots? |
| `plot_dim` | Number of inches squared for the plot if added. |
| `compare_plots` | Add some plots comparing the results. |
| `padj_type` | Add a consistent p adjustment of this type. |
| `...` | Arguments passed to significance and abundance tables. |

## Value

Table combining limma/edger/deseq outputs.

## See Also

[all_pairwise](#)

## Examples

```
## Not run:
 expt <- create_expt(metadata="some_metadata.xlsx", gene_info=funkytown)
 big_result <- all_pairwise(expt, model_batch=FALSE)
 pretty <- combine_de_tables(big_result, table='t12_vs_t0')
 pretty <- combine_de_tables(big_result, table='t12_vs_t0',
                             keepers=list("avsb" = c("a","b")))
 pretty <- combine_de_tables(big_result, table='t12_vs_t0',
                             keepers=list("avsb" = c("a","b")),
                             excludes=list("description" = c("sno","rRNA")))

## End(Not run)
```

---

combine_expts                  *Take two expressionsets and smoosh them together.*

---

### Description

Because of the extra sugar I added to expressionSets, the combine() function needs a little help when combining expts. Notably, the information from tximport needs some help.

### Usage

```
combine_expts(
  expt1,
  expt2,
  condition = "condition",
  batch = "batch",
  merge_meta = TRUE
)
```

### Arguments

| | |
|---|---|
| expt1 | First expt object. |
| expt2 | Second expt object. |
| condition | Column with which to reset the conditions. |
| batch | Column with which to reset the batches. |
| merge_meta | Merge the metadata when they mismatch? This should perhaps default to TRUE. |

### Value

Larger expt.

### Examples

```
 ## Not run:
  ## I am trying to get rid of all my dontrun sections, but I don't have two
  ## expressionsets to combine.
  expt1 <- create_expt(first_meta)
  expt2 <- create_expt(second_meta)
  combined <- combine_expts(expt1, expt2, merge_meta=TRUE)

## End(Not run)
```

combine_extracted_plots

*Gather data required to make MA/Volcano plots for pairwise comparisons.*

### Description

It should be possible to significantly simplify the arguments passed to this function, but I have thus far focused only on getting it to work with the newly split-apart combine_de_tables() functions.

### Usage

```
combine_extracted_plots(
  name,
  combined,
  denominator,
  numerator,
  plot_inputs,
  include_basic = TRUE,
  include_deseq = TRUE,
  include_edger = TRUE,
  include_limma = TRUE,
  include_ebseq = FALSE,
  loess = FALSE,
  logfc = 1,
  p = 0.05,
  do_inverse = FALSE,
  found_table = NULL
)
```

### Arguments

| | |
|---|---|
| name | Name of the table to plot. |
| combined | Modified pairwise result, containing the various DE methods. |
| denominator | Name of the denominator coefficient. |
| numerator | Name of the numerator coefficient. |
| plot_inputs | The individual outputs from limma etc. |
| include_basic | Add basic data? |
| include_deseq | Add deseq data? |
| include_edger | Add edger data? |
| include_limma | Add limma data? |
| include_ebseq | Add ebseq data? |
| loess | Add a loess estimation? |
| do_inverse | Flip the numerator/denominator? |
| found_table | The table name actually used. |

combine_single_de_table

*Given a limma, edger, and deseq table, combine them into one.*

### Description

This combines the outputs from the various differential expression tools and formalizes some column names to make them a little more consistent.

### Usage

```
combine_single_de_table(
  li = NULL,
  ed = NULL,
  eb = NULL,
  de = NULL,
  ba = NULL,
  table_name = "",
  final_table_names = c(),
  annot_df = NULL,
  do_inverse = FALSE,
  adjp = TRUE,
  padj_type = "fdr",
  include_deseq = TRUE,
  include_edger = TRUE,
  include_ebseq = TRUE,
  include_limma = TRUE,
  include_basic = TRUE,
  lfc_cutoff = 1,
  p_cutoff = 0.05,
  excludes = NULL,
  sheet_count = 0
)
```

### Arguments

| | |
|---|---|
| li | Limma output table. |
| ed | Edger output table. |
| eb | EBSeq output table |
| de | DESeq2 output table. |
| ba | Basic output table. |
| table_name | Name of the table to merge. |
| final_table_names | |
| | Vector of the final table names. |
| annot_df | Add some annotation information? |

| | |
|---|---|
| `do_inverse` | Invert the fold changes? |
| `adjp` | Use adjusted p-values? |
| `padj_type` | Add this consistent p-adjustment. |
| `include_deseq` | Include tables from deseq? |
| `include_edger` | Include tables from edger? |
| `include_ebseq` | Include tables from ebseq? |
| `include_limma` | Include tables from limma? |
| `include_basic` | Include the basic table? |
| `lfc_cutoff` | Preferred logfoldchange cutoff. |
| `p_cutoff` | Preferred pvalue cutoff. |
| `excludes` | Set of genes to exclude from the output. |
| `sheet_count` | What sheet is being written? |

## Value

List containing a) Dataframe containing the merged limma/edger/deseq/basic tables, and b) A summary of how many genes were observed as up/down by output table.

## See Also

**data.table openxlsx**

---

| | |
|---|---|
| compare_de_results | *Compare the results of separate all_pairwise() invocations.* |

---

## Description

Where compare_led_tables looks for changes between limma and friends, this function looks for differences/similarities across the models/surrogates/etc across invocations of limma/deseq/edger.

## Usage

```
compare_de_results(
  first,
  second,
  cor_method = "pearson",
  try_methods = c("limma", "deseq", "edger")
)
```

## Arguments

| | |
|---|---|
| `first` | One invocation of combine_de_tables to examine. |
| `second` | A second invocation of combine_de_tables to examine. |
| `cor_method` | Method to use for cor.test(). |
| `try_methods` | List of methods to attempt comparing. |

## Details

Tested in 29de_shared.R

## Value

A list of compared columns, tables, and methods.

## Examples

```
## Not run:
 first <- all_pairwise(expt, model_batch=FALSE, excel="first.xlsx")
 second <- all_pairwise(expt, model_batch="svaseq", excel="second.xlsx")
 comparison <- compare_de_results(first$combined, second$combined)

## End(Not run)
```

---

compare_go_searches        *Compare the results from different ontology tools*

---

## Description

Combine the results from goseq, cluster profiler, topgo, and gostats; poke at them with a stick and
see what happens. The general idea is to pull the p-value data from each tool and contrast that to
the set of all possibile ontologies. This allows one to do a correlation coefficient between them. In
addition, take the 1-pvalue for each ontology for each tool. Thus for strong p-values the score will
be near 1 and so we can sum the scores for all the tools. Since topgo has 4 tools, the total possible
is 7 if everything has a p-value equal to 0.

## Usage

```
compare_go_searches(goseq = NULL, cluster = NULL, topgo = NULL, gostats = NULL)
```

## Arguments

| | |
|---|---|
| goseq | The goseq result from simple_goseq() |
| cluster | The result from simple_clusterprofiler() |
| topgo | Guess |
| gostats | Yep, ditto |

## Value

a summary of the similarities of ontology searches

## See Also

**goseq clusterProfiler topGO goStats**

---

compare_logfc_plots *Compare logFC values from limma and friends*

---

### Description

There are some peculiar discrepencies among these tools, what is up with that?

### Usage

```
compare_logfc_plots(combined_tables)
```

### Arguments

combined_tables

> The combined tables from limma et al.

### Details

Invoked by combine_de_tables() in order to compare the results.

### Value

Some plots

### See Also

[plot_linear_scatter](#)

### Examples

```
## Not run:
 limma_vs_deseq_vs_edger <- compare_logfc_plots(combined)
 ## Get a list of plots of logFC by contrast of LvD, LvE, DvE
 ## It provides comparisons against the basic analysis, but who cares about that.

## End(Not run)
```

compare_significant_contrasts

> *Implement a cleaner version of 'subset_significants' from analyses with Maria Adelaida.*

## Description

This should provide nice venn diagrams and some statistics to compare 2 or 3 contrasts in a differential expression analysis.

## Usage

```
compare_significant_contrasts(
  sig_tables,
  compare_by = "deseq",
  weights = FALSE,
  contrasts = c(1, 2, 3)
)
```

## Arguments

| | |
|---|---|
| sig_tables | Set of significance tables to poke at. |
| compare_by | Use which program for the comparisons? |
| weights | When printing venn diagrams, weight them? |
| contrasts | List of contrasts to compare. |

compare_surrogate_estimates

> *Perform a comparison of the surrogate estimators demonstrated by Jeff Leek.*

## Description

This is entirely derivative, but seeks to provide similar estimates for one's own actual data and catch corner cases not taken into account in that document (for example if the estimators don't converge on a surrogate variable). This will attempt each of the surrogate estimators described by Leek: pca, sva supervised, sva unsupervised, ruv supervised, ruv residuals, ruv empirical. Upon completion it will perform the same limma expression analysis and plot the ranked t statistics as well as a correlation plot making use of the extracted estimators against condition/batch/whatever else. Finally, it does the same ranking plot against a linear fitting Leek performed and returns the whole pile of information as a list.

## Usage

```
compare_surrogate_estimates(
  expt,
  extra_factors = NULL,
  filter_it = TRUE,
  filter_type = TRUE,
  do_catplots = FALSE,
  surrogates = "be",
  ...
)
```

## Arguments

| | |
|---|---|
| expt | Experiment containing a design and other information. |
| extra_factors | Character list of extra factors which may be included in the final plot of the data. |
| filter_it | Most of the time these surrogate methods get mad if there are 0s in the data. Filter it? |
| filter_type | Type of filter to use when filtering the input data. |
| do_catplots | Include the catplots? They don't make a lot of sense yet, so probably no. |
| surrogates | Use 'be' or 'leek' surrogate estimates, or choose a number. |
| ... | Extra arguments when filtering. |

## Value

List of the results.

---

| concatenate_runs | *Sum the reads/gene for multiple sequencing runs of a single condition/batch.* |
|---|---|

---

## Description

On occasion we have multiple technical replicates of a sequencing run. This can use a column in the experimental design to identify those replicates and sum the counts into a single column in the count tables.

## Usage

```
concatenate_runs(expt, column = "replicate")
```

## Arguments

| | |
|---|---|
| expt | Experiment class containing the requisite metadata and count tables. |
| column | Column of the design matrix used to specify which samples are replicates. |

## Details

Untested as of 2016-12-01, but used in a couple of projects where sequencing runs got repeated.

## Value

Expt with the concatenated counts, new design matrix, batches, conditions, etc.

## See Also

**Biobase** `exprs` `fData` `pData`

## Examples

```
## Not run:
 compressed <- concatenate_runs(expt)

## End(Not run)
```

---

convert_counts                    *Perform a cpm/rpkm/whatever transformation of a count table.*

---

## Description

I should probably tell it to also handle a simple df/vector/list of gene lengths, but I haven't. cp_seq_m is a cpm conversion of the data followed by a rp-ish conversion which normalizes by the number of the given oligo. By default this oligo is 'TA' because it was used for tnseq which should be normalized by the number of possible transposition sites by mariner. It could, however, be used to normalize by the number of methionines, for example – if one wanted to do such a thing.

## Usage

```
convert_counts(count_table, method = "raw", ...)
```

## Arguments

| | |
|---|---|
| method | Type of conversion to perform: edgecpm/cpm/rpkm/cp_seq_m. |
| ... | Options I might pass from other functions are dropped into arglist, used by rpkm (gene lengths) and divide_seq (genome, pattern to match, and annotation type). |
| data | Matrix of count data. |

## Value

Dataframe of cpm/rpkm/whatever(counts)

## See Also

**edgeR Biobase** `cpm`

## Examples

```
## Not run:
 converted_table = convert_counts(count_table, method='cbcbcpm')

## End(Not run)
```

---

convert_gsc_ids                *Use AnnotationDbi to translate geneIDs from type x to type y.*

---

## Description

This is intended to convert all the IDs in a geneSet from one ID type to another and giving back the
geneSet with the new IDs.

## Usage

```
convert_gsc_ids(
  gsc,
  orgdb = "org.Hs.eg.db",
  from_type = NULL,
  to_type = "ENTREZID"
)
```

## Arguments

| | |
|---|---|
| gsc | geneSetCollection with IDs of a type one wishes to change. |
| orgdb | Annotation object containing the various IDs. |
| from_type | Name of the ID which your gsc is using. This can probably be automagically detected... |
| to_type | Name of the ID you wish to use. |

## Details

One caveat: this will collapse redundant IDs via unique().

## Value

Fresh gene set collection replete with new names.

---

| cordist | *Similarity measure which combines elements from Pearson correlation and Euclidean distance.* |
| --- | --- |

---

## Description

Here is Keith's summary: Where the cor returns the Pearson correlation matrix for the input matrix, and the dist function returns the Euclidean distance matrix for the input matrix. The LHS of the equation is simply the sign of the correlation function, which serves to preserve the sign of the interaction. The RHS combines the Pearson correlation and the log inverse Euclidean distance with equal weights. The result is a number in the range from -1 to 1 where values close to -1 indicate a strong negative correlation and values close to 1 indicate a strong positive corelation. While the Pearson correlation and Euclidean distance each contribute equally in the above equation, one could also assign tuning parameters to each of the metrics to allow for unequal contributions.

## Usage

```
cordist(
  data,
  cor_method = "pearson",
  dist_method = "euclidean",
  cor_weight = 0.5,
  ...
)
```

## Arguments

| | |
| --- | --- |
| data | Matrix of data |
| cor_method | Which correlation method to use? |
| dist_method | Which distance method to use? |
| cor_weight | 0-1 weight of the correlation, the distance weight will be 1-cor_weight. |
| ... | extra arguments for cor/dist |

## Value

Matrix of the correlation-modified distances of the original matrix.

## Author(s)

Keigth Hughitt

---

correlate_de_tables     *See how similar are results from limma/deseq/edger/ebseq.*

---

### Description

limma, DEseq2, and EdgeR all make somewhat different assumptions. and choices about what makes a meaningful set of differentially. expressed genes. This seeks to provide a quick and dirty metric describing the degree to which they (dis)agree.

### Usage

```
correlate_de_tables(results, annot_df = NULL, extra_contrasts = NULL)
```

### Arguments

results        Data from do_pairwise()

annot_df       Include annotation data?

### Details

Invoked by all_pairwise().

### Value

Heatmap showing how similar they are along with some correlations betwee the three players.

### See Also

[limma_pairwise](limma_pairwise) [edger_pairwise](edger_pairwise) [deseq2_pairwise](deseq2_pairwise)

### Examples

```
## Not run:
 l = limma_pairwise(expt)
 d = deseq_pairwise(expt)
 e = edger_pairwise(expt)
 fun = compare_led_tables(limma=l, deseq=d, edger=e)

## End(Not run)
```

---

count_expt_snps *Gather snp information for an expt*

---

### Description

This function attempts to gather a set of variant positions using an extant expressionset. This therefore seeks to keep the sample metadata consistent with the original data. In its current iteration, it therefore makes some potentially bad assumptions about the naming conventions for its input files. It furthermore assumes inputs from the variant calling methods in cyoa.

### Usage

```
count_expt_snps(
  expt,
  type = "counts",
  annot_column = "bcftable",
  tolower = TRUE,
  snp_column = "diff_count"
)
```

### Arguments

| | |
|---|---|
| expt | an expressionset from which to extract information. |
| type | Use counts / samples or ratios? |
| annot_column | Column in the metadata for getting the table of bcftools calls. |
| tolower | Lowercase stuff like 'HPGL'? |
| snp_column | Which column of the parsed bcf table contains our interesting material? |

### Value

A new expt object

### Examples

```
 ## Not run:
 expt <- create_expt(metadata, gene_information)
 snp_expt <- count_expt_snps(expt)
 ## This assumes that the metadata has a column named 'bcftable' with one file per
 ## cell.  These files in turn should have a column named 'diff_count' which will
 ## be the source of the numbers found when doing exprs(snp_expt).

 ## End(Not run)
```

---

count_nmer                              *Count n-mers in a given data set using Biostrings*

---

### Description

This just calls PDict() and vcountPDict() on a sequence database given a pattern and number of mismatches. This may be used by divide_seq() normalization.

### Usage

```
count_nmer(genome, pattern = "ATG", mismatch = 0)
```

### Arguments

genome          Sequence database, genome in this case.

pattern         Count off this string.

mismatch        How many mismatches are acceptable?

### Value

Set of counts by sequence.

---

counts_from_surrogates
                            *A single place to extract count tables from a set of surrogate variables.*

---

### Description

Given an initial set of counts and a series of surrogates, what would the resulting count table look like? Hopefully this function answers that question.

### Usage

```
counts_from_surrogates(
  data,
  adjust = NULL,
  design = NULL,
  method = "ruv",
  cond_column = "condition",
  matrix_scale = "linear",
  return_scale = "linear",
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | Original count table, may be an expt/expressionset or df/matrix. |
| `adjust` | Surrogates with which to adjust the data. |
| `design` | Experimental design if it is not included in the expressionset. |
| `method` | Which methodology to follow, ideally these agree but that seems untrue. |
| `cond_column` | design column containing the condition data. |
| `matrix_scale` | Was the input for the surrogate estimator on a log or linear scale? |
| `return_scale` | Does one want the output linear or log? |
| `...` | Arguments passed to downstream functions. |

## Value

A data frame of adjusted counts.

## See Also

**sva RUVSeq**

---

| `cp_options` | *Set up appropriate option sets for clusterProfiler* |
|---|---|

---

## Description

This hard-sets some defaults for orgdb/kegg databases when using clusterProfiler.

## Usage

```
cp_options(species)
```

## Arguments

| | |
|---|---|
| `species` | Currently it only works for humans and fruit flies. |

---

create_expt                 *Wrap bioconductor's expressionset to include some extra information.*

---

### Description

The primary innovation of this function is that it will check the metadata for columns containing filenames for the count tables, thus hopefully making the collation and care of metadata/counts easier. For example, I have some data which has been mapped against multiple species. I can use this function and just change the file_column argument to pick up each species' tables.

### Usage

```
create_expt(
  metadata = NULL,
  gene_info = NULL,
  count_dataframe = NULL,
  sanitize_rownames = FALSE,
  sample_colors = NULL,
  title = NULL,
  notes = NULL,
  countdir = NULL,
  include_type = "all",
  include_gff = NULL,
  file_column = "file",
  id_column = NULL,
  savefile = NULL,
  low_files = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| metadata | Comma separated file (or excel) describing the samples with information like condition, batch, count_filename, etc. |
| gene_info | Annotation information describing the rows of the data set, this often comes from a call to import.gff() or biomart or organismdbi. |
| count_dataframe | |
| | If one does not wish to read the count tables from the filesystem, they may instead be fed as a data frame here. |
| sanitize_rownames | |
| | Clean up weirdly written gene IDs? |
| sample_colors | List of colors by condition, if not provided it will generate its own colors using colorBrewer. |
| title | Provide a title for the expt? |
| notes | Additional notes? |

| countdir | Directory containing count tables. |
|---|---|
| include_type | I have usually assumed that all gff annotations should be used, but that is not always true, this allows one to limit to a specific annotation type. |
| include_gff | Gff file to help in sorting which features to keep. |
| file_column | Column to use in a gene information dataframe for |
| id_column | Column which contains the sample IDs. |
| savefile | Rdata filename prefix for saving the data of the resulting expt. |
| low_files | Explicitly lowercase the filenames when searching the filesystem? |
| ... | More parameters are fun! |

**Value**

experiment an expressionset

**See Also**

**Biobase** `pData` `fData` `exprs` `read_counts_expt`

**Examples**

```
load(file=system.file("cdm_expt.rda", package="hpgltools"))
head(cdm_counts)
head(cdm_metadata)
## The gff file has differently labeled locus tags than the count tables
## The naming standard changed since this experiment was performed and I
## downloaded a new gff file.
rownames(gas_gff_annot) <- make.names(gsub(pattern="(Spy)_", replacement="\\1",
                                          x=gas_gff_annot[["locus_tag"]]), unique=TRUE)
mgas_expt <- create_expt(metadata=cdm_metadata, gene_info=gas_gff_annot,
                        count_dataframe=cdm_counts)
head(pData(mgas_expt))
## An example using count tables referenced in the metadata.
 sb_annot <- load_trinotate_annotations(
   trinotate=system.file("sb/trinotate_head.csv.xz", package="hpgltools"))
sb_annot <- as.data.frame(sb_annot)
  rownames(sb_annot) <- make.names(sb_annot[["transcript_id"]], unique=TRUE)
 sb_annot[["rownames"]] <- NULL
 untarred <- utils::untar(tarfile=system.file("sb/preprocessing.tar.xz",
                         package="hpgltools"))
 sb_expt <- create_expt(metadata="preprocessing/kept_samples.xlsx",
                       gene_info=sb_annot)
dim(exprs(sb_expt))
dim(fData(sb_expt))
pData(sb_expt)
## There are lots of other ways to use this, for example:
## Not run:
 new_experiment <- create_expt(metadata="some_csv_file.csv", gene_info=gene_df)
  ## Remember that this depends on an existing data structure of gene annotations.
  meta <- extract_metadata("some_supplementary_materials_xls_file_I_downloaded.xls")
 another_expt <- create_expt(metadata=meta, gene_info=annotations, count_dataframe=df_I_downloaded)
```

```
## End(Not run)
```

---

de_venn                    *Create venn diagrams describing how well deseq/limma/edger agree.*

---

### Description

The sets of genes provided by limma and friends would ideally always agree, but they do not. Use
this to see out how much the (dis)agree.

### Usage

```
de_venn(table, adjp = FALSE, p = 0.05, lfc = 0, ...)
```

### Arguments

| | |
|---|---|
| table | Which table to query? |
| adjp | Use adjusted p-values |
| p | p-value cutoff, I forget what for right now. |
| lfc | What fold-change cutoff to include? |
| ... | More arguments are passed to arglist. |

### Value

A list of venn plots

### See Also

**venneuler Vennerable**

### Examples

```
## Not run:
 bunchovenns <- de_venn(pairwise_result)

## End(Not run)
```

---

default_norm                    *Perform a default normalization of some data*

---

### Description

This just calls normalize expt with the most common arguments except log2 transformation, but that may be appended with 'transform=log2', so I don't feel bad. Indeed, it will allow you to overwrite any arguments if you wish. In our work, the most common normalization is: quantile(cpm(low-filter(data))).

### Usage

```
default_norm(expt, ...)
```

### Arguments

| | |
|---|---|
| expt | An expressionset containing expt object |
| ... | More options to pass to normalize_expt() |

### Value

The normalized expt

### See Also

[normalize_expt](#)

---

default_proper                  *Invoke PROPER and replace its default data set with data of interest.*

---

### Description

Recent reviewers of Najib's grants have taken an increased interest in knowing the statistical power of the various experiments. He queried Dr. Corrada-Bravo who suggested PROPER. I spent some time looking through it and, with some revervations, modified its workflow to (at least in theory) be able to examine any dataset. The workflow in question is particularly odd and warrants further discussion/analysis. This function invokes PROPER exactly as it was performed in their paper.

**Usage**

```
default_proper(
  de_tables,
  p = 0.05,
  experiment = "cheung",
  nsims = 20,
  reps = c(3, 5, 7, 10),
  de_method = "edger",
  alpha_type = "fdr",
  alpha = 0.1,
  stratify = "expr",
  target = "lfc",
  filter = "none",
  delta = 0.5
)
```

**Arguments**

| | |
|---|---|
| de_tables | A set of differential expression results, presumably from EdgeR or DESeq2. |
| p | Cutoff |
| experiment | The default data set in PROPER is entitled 'cheung'. |
| nsims | Number of simulations to perform. |
| reps | Simulate these number of experimental replicates. |
| de_method | There are a couple choices here for tools which are pretty old, my version of this only accepts deseq or edger. |
| alpha_type | I assume p-adjust type. |
| alpha | Accepted fdr rate. |
| stratify | There are a few options here, I don't fully understand them. |
| target | Cutoff. |
| filter | Apply a filter? |
| delta | Not epsilon! (E.g. I forget what this does). |

**Value**

List containing the various results and plots from proper.

---

deparse_go_value          *Extract more easily readable information from a GOTERM datum.*

---

**Description**

The output from the GOTERM/GO.db functions is inconsistent, to put it nicely. This attempts to extract from that heterogeneous datatype something easily readable. Example: Synonym() might return any of the following: NA, NULL, "NA", "NULL", c("NA",NA,"GO:00001"), "GO:00002", c("Some text",NA, NULL, "GO:00003") This function will boil that down to 'not found', '', 'GO:00004', or "GO:0001, some text, GO:00004"

## Usage

```
deparse_go_value(value)
```

## Arguments

value          Result of try(as.character(somefunction(GOTERM[id])), silent=TRUE). some-
               function would be 'Synonym' 'Secondary' 'Ontology', etc...

## Value

something more sane (hopefully).

## See Also

**GO.db**

## Examples

```
## Not run:
 ## goterms = GOTERM[ids]
 ## sane_goterms = deparse_go_value(goterms)

## End(Not run)
```

---

deseq_pairwise          *deseq_pairwise() Because I can't be trusted to remember '2'.*

---

## Description

This calls deseq2_pairwise(...) because I am determined to forget typing deseq2.

## Usage

```
deseq_pairwise(...)
```

## Arguments

...            I like cats.

## Value

stuff deseq2_pairwise results.

## See Also

[deseq2_pairwise](#)

---

deseq_try_sv                         *Given a set of surrogate variables from sva and friends, try adding*
                                     *them to a DESeqDataSet.*

---

### Description

Sometimes sva returns a set of surrogate variable estimates which lead to models which are in-
valid according to DESeq2. This function will try before buying and tell the user if the sva model
additions are valid according to DESeq.

### Usage

```
deseq_try_sv(data, summarized, svs, num_sv = NULL)
```

### Arguments

| | |
|---|---|
| data | DESeqDataSet to test out. |
| summarized | Existing DESeq metadata to append svs. |
| svs | Surrogates from sva and friends to test out. |
| num_sv | Optionally, provide the number of SVs, primarily used if recursing in the hunt for a valid number of surrogates. |

### Value

DESeqDataSet with at least some of the SVs appended to the model.

---

deseq2_pairwise                      *Set up model matrices contrasts and do pairwise comparisons of all*
                                     *conditions using DESeq2.*

---

### Description

Invoking DESeq2 is confusing, this should help.

### Usage

```
deseq2_pairwise(
  input = NULL,
  conditions = NULL,
  batches = NULL,
  model_cond = TRUE,
  model_batch = TRUE,
  model_intercept = FALSE,
  alt_model = NULL,
  extra_contrasts = NULL,
```

```
    annot_df = NULL,
    force = FALSE,
    deseq_method = "long",
    ...
)
```

## Arguments

| | |
|---|---|
| `input` | Dataframe/vector or expt class containing data, normalization state, etc. |
| `conditions` | Factor of conditions in the experiment. |
| `batches` | Factor of batches in the experiment. |
| `model_cond` | Is condition in the experimental model? |
| `model_batch` | Is batch in the experimental model? |
| `model_intercept` | |
| | Use an intercept model? |
| `alt_model` | Provide an arbitrary model here. |
| `extra_contrasts` | |
| | Provide extra contrasts here. |
| `annot_df` | Include some annotation information in the results? |
| `force` | Force deseq to accept data which likely violates its assumptions. |
| `deseq_method` | The DESeq2 manual shows a few ways to invoke it, I make 2 of them available here. |
| `...` | Triple dots! Options are passed to arglist. |

## Details

Like the other _pairwise() functions, this attempts to perform all pairwise contrasts in the provided data set. The details are of course slightly different when using DESeq2. Thus, this uses the function choose_binom_dataset() to try to ensure that the incoming data is appropriate for DESeq2 (if one normalized the data, it will attempt to revert to raw counts, for example). It continues on to extract the conditions and batches in the data, choose an appropriate experimental model, and run the DESeq analyses as described in the manual. It defaults to using an experimental batch factor, but will accept a string like 'sva' instead, in which case it will use sva to estimate the surrogates, and append them to the experimental design. The deseq_method parameter may be used to apply different DESeq2 code paths as outlined in the manual. If you want to play with non-standard data, the force argument will round the data and shoe-horn it into DESeq2.

## Value

List including the following information: run = the return from calling DESeq() denominators = list of denominators in the contrasts numerators = list of the numerators in the contrasts conditions = the list of conditions in the experiment coefficients = list of coefficients making the contrasts all_tables = list of DE tables

## See Also

**DESeq2 Biobase stats**

## Examples

```
## Not run:
 pretend = deseq2_pairwise(data, conditions, batches)

## End(Not run)
```

---

disjunct_pvalues        *Test for infected/control/beads – a placebo effect?*

---

### Description

This was a function I copied out of Keith/Hector/Laura/Cecilia's paper in which they sought to discriminate the effect of inert beads on macrophages vs. the effect of parasites. The simpler way of expressing it is: take the worst p-value observed for the pair of contrasts, infected/uninfected and beads/uninfected.

### Usage

```
disjunct_pvalues(contrast_fit, cellmeans_fit, conj_contrasts, disj_contrast)
```

### Arguments

| | |
|---|---|
| contrast_fit | Result of lmFit. |
| cellmeans_fit | Result of a cellmeans fit. |
| conj_contrasts | Result from the makeContrasts of the first set. |
| disj_contrast | Result of the makeContrasts of the second set. |

### Details

The goal is therefore to find responses different than beads The null hypothesis is (H0): (infected == uninfected) | (infected == beads) The alt hypothesis is (HA): (infected != uninfected) & (infected != beads)

---

divide_seq        *Express a data frame of counts as reads per pattern per million.*

---

### Description

This uses a sequence pattern rather than length to normalize sequence. It is essentially fancy pants rpkm.

### Usage

```
divide_seq(counts, ...)
```

## Arguments

| | |
|---|---|
| counts | Read count matrix. |
| ... | Options I might pass from other functions are dropped into arglist. |

## Value

The RPseqM counts

## See Also

**edgeR Rsamtools** `FaFile` `rpkm`

## Examples

```
## Not run:
 cptam <- divide_seq(cont_table, fasta="mgas_5005.fasta.xz", gff="mgas_5005.gff.xz")

## End(Not run)
```

---

| do_pairwise | *Generalize pairwise comparisons* |
|---|---|

---

## Description

I want to multithread my pairwise comparisons, this is the first step in doing so.

## Usage

```
do_pairwise(type, ...)
```

## Arguments

| | |
|---|---|
| type | Which type of pairwise comparison to perform |
| ... | Set of arguments intended for limma_pairwise(), edger_pairwise(), and friends. |

## Details

Used to make parallel operations easier.

## Value

Result from limma/deseq/edger/basic

## See Also

`limma_pairwise` `edger_pairwise` `deseq_pairwise` `basic_pairwise`

---

do_topgo                            *An attempt to make topgo invocations a bit more standard.*

---

**Description**

My function 'simple_topgo()' was excessively long and a morass of copy/pasted fragments. This attempts to simplify that and converge on a single piece of code for all the methodologies provided by topgo.

**Usage**

```
do_topgo(
  type,
  go_map = NULL,
  fisher_genes = NULL,
  ks_genes = NULL,
  selector = "topDiffGenes",
  sigforall = TRUE,
  numchar = 300,
  pval_column = "adj.P.Val",
  overwrite = FALSE,
  cutoff = 0.05,
  densities = FALSE,
  pval_plots = TRUE
)
```

**Arguments**

| | |
|---|---|
| type | Type of topgo search to perform: fisher, KS, EL, or weight. |
| go_map | Mappings of gene and GO IDs. |
| fisher_genes | List of genes used for fisher analyses. |
| ks_genes | List of genes used for KS analyses. |
| selector | Function to use when selecting genes. |
| sigforall | Provide significance metrics for all ontologies observed, not only the ones deemed statistically significant. |
| numchar | A limit on characters printed when printing topgo tables (used?) |
| pval_column | Column from which to extract DE p-values. |
| overwrite | Overwrite an existing gene ID/GO mapping? |
| cutoff | Define 'significant'? |
| densities | Perform gene density plots by ontology? |
| pval_plots | Print p-values plots as per clusterProfiler? |

**Value**

A list of results from the various tests in topGO.

---

download_gbk *A genbank accession downloader scurrilously stolen from ape.*

---

### Description

This takes and downloads genbank accessions.

### Usage

```
download_gbk(accessions = "AE009949", write = TRUE)
```

### Arguments

| | |
|---|---|
| accessions | An accession – actually a set of them. |
| write | Write the files? Otherwise return a list of the strings |

### Details

Tested in test_40ann_biomartgenbank.R In this function I stole the same functionality from the ape package and set a few defaults so that it hopefully fails less often.

### Value

A list containing the number of files downloaded and the character strings acquired.

### Author(s)

The ape authors with some modifications by atb.

### See Also

**ape**

### Examples

```
written <- download_gbk(accessions="AE009949")
written$written_file
```

---

```
download_microbesonline_files
```
                    *Download the various file formats from microbesoline.*

---

### Description

Microbesonline provides an interesting set of file formats to download. Each format proves useful under one condition or another, ergo this defaults to iterating through them all and getting every file.

### Usage

```
download_microbesonline_files(id = "160490", type = NULL)
```

### Arguments

| | |
|---|---|
| `id` | Species ID to query. |
| `type` | File type(s) to download, if left null it will grab the genbank, tab, protein fasta, transcript fasta, and genome. |

### Value

List describing the files downloaded and their locations.

---

```
download_uniprot_proteome
```
                    *Download the txt uniprot data for a given accession/species.*

---

### Description

Uniprot is an astonishing resource, but man is it a pain to use. Hopefully this function will help. It takes either a uniprot accession, taxonomy ID, or species name and does its best to find the appropriate uniprot data. This is therefore primarily used by load_uniprot_annotations().

### Usage

```
download_uniprot_proteome(
  accession = NULL,
  species = NULL,
  taxonomy = NULL,
  all = FALSE,
  first = FALSE
)
```

## Arguments

| | |
|---|---|
| `accession` | Which accession to grab? |
| `species` | Or perhaps species? |
| `taxonomy` | Query for a specific taxonomy ID rather than species/accession? |
| `all` | If there are more than 1 hit, grab them all? |
| `first` | Or perhaps just grab the first hit? |

## Value

A filename/accession tuple.

## Examples

```
uniprot_sc_downloaded <- download_uniprot_proteome(species="Saccharomyces cerevisiae S288c")
uniprot_sc_downloaded$filename
uniprot_sc_downloaded$species
```

---

| | |
|---|---|
| ebseq_few | *Invoke EBMultiTest() when we do not have too many conditions to deal with.* |

---

## Description

Starting at approximately 5 conditions, ebseq becomes too unwieldy to use effectively. But, its results until then are pretty neat.

## Usage

```
ebseq_few(
  data,
  conditions,
  patterns = NULL,
  ng_vector = NULL,
  rounds = 10,
  target_fdr = 0.05,
  norm = "median"
)
```

## Arguments

| | |
|---|---|
| `data` | Expressionset/matrix |
| `conditions` | Factor of conditions in the data to compare. |
| `patterns` | Set of patterns as described in the ebseq documentation to query. |
| `ng_vector` | Passed along to ebmultitest(). |
| `rounds` | Passed to ebseq. |
| `target_fdr` | Passed to ebseq. |
| `norm` | Normalization method to apply to the data. |

---

ebseq_pairwise                    *Set up model matrices contrasts and do pairwise comparisons of all*
                                  *conditions using EBSeq.*

---

### Description

Invoking EBSeq is confusing, this should help.

### Usage

```
ebseq_pairwise(
  input = NULL,
  patterns = NULL,
  conditions = NULL,
  batches = NULL,
  model_cond = NULL,
  model_intercept = NULL,
  alt_model = NULL,
  model_batch = NULL,
  ng_vector = NULL,
  rounds = 10,
  target_fdr = 0.05,
  method = "pairwise_subset",
  norm = "median",
  force = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| input | Dataframe/vector or expt class containing data, normalization state, etc. |
| patterns | Set of expression patterns to query. |
| conditions | Not currently used, but passed from all_pairwise() |
| batches | Not currently used, but passed from all_pairwise() |
| model_cond | Not currently used, but passed from all_pairwise() |
| model_intercept | |
| | Not currently used, but passed from all_pairwise() |
| alt_model | Not currently used, but passed from all_pairwise() |
| model_batch | Not currently used, but passed from all_pairwise() |
| ng_vector | I think this is for isoform quantification, but am not yet certain. |
| rounds | Number of iterations for doing the multi-test |
| target_fdr | Definition of 'significant' |

| method | The default ebseq methodology is to create the set of all possible 'patterns' in the data; for data sets which are more than trivially complex, this is not tenable, so this defaults to subsetting the data into pairs of conditions. |
| --- | --- |
| norm | Normalization method to use. |
| force | Force ebseq to accept bad data (notably NA containing stuff from proteomics. |
| ... | Extra arguments currently unused. |

## Value

List containing tables from ebseq, the conditions tested, and the ebseq table of conditions.

## Examples

```
 ## Not run:
  expt <- create_expt(metadata="sample_sheet.xlsx", gene_info=annotations)
  ebseq_de <- ebseq_pairwise(input=expt)

## End(Not run)
```

---

ebseq_pairwise_subset    *Perform pairwise comparisons with ebseq, one at a time.*

---

## Description

This uses the same logic as in the various *_pairwise functions to invoke the 'normal' ebseq pairwise comparison for each pair of conditions in an expressionset. It therefore avoids the strange logic inherent in the ebseq multitest function.

## Usage

```
ebseq_pairwise_subset(
  input,
  ng_vector = NULL,
  rounds = 10,
  target_fdr = 0.05,
  model_batch = FALSE,
  model_cond = TRUE,
  model_intercept = FALSE,
  alt_model = NULL,
  conditions = NULL,
  norm = "median",
  force = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `input` | Expressionset/expt to perform de upon. |
| `ng_vector` | Passed on to ebseq, I forget what this does. |
| `rounds` | Passed on to ebseq, I think it defines how many iterations to perform before return the de estimates |
| `target_fdr` | If we reach this fdr before iterating rounds times, return. |
| `model_batch` | Provided by all_pairwise() I do not think a Bayesian analysis really cares about models, but if one wished to try to add a batch factor, this would be the place to do it. It is currently ignored. |
| `model_cond` | Provided by all_pairwise(), ibid. |
| `model_intercept` | |
| | Ibid. |
| `alt_model` | Ibid. |
| `conditions` | Factor of conditions in the data, used to define the contrasts. |
| `norm` | EBseq normalization method to apply to the data. |
| `force` | Flag used to force inappropriate data into the various methods. |
| `...` | Extra arguments passed downstream, noably to choose_model() |

## Value

A pairwise comparison of the various conditions in the data.

---

ebseq_size_factors            *Choose the ebseq normalization method to apply to the data.*

---

## Description

EBSeq provides three normaliation methods. Median, Quantile, and Rank. Choose among them here.

## Usage

```
ebseq_size_factors(data_mtrx, norm = NULL)
```

## Arguments

| | |
|---|---|
| `data_mtrx` | This is exprs(expressionset) |
| `norm` | The method to pass along. |

## Value

a new matrix using the ebseq specific method of choice.

ebseq_two                    *The primary function used in my EBSeq implementation.*

## Description

Most of the time, my invocation of ebseq will fall into this function.

## Usage

```
ebseq_two(
  pair_data,
  conditions,
  numerator = 2,
  denominator = 1,
  ng_vector = NULL,
  rounds = 10,
  target_fdr = 0.05,
  norm = "median",
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| pair_data | Matrix containing the samples comprising two experimental factors of interest. |
| conditions | Factor of conditions in the data. |
| numerator | Which factor has the numerator in the data. |
| denominator | Which factor has the denominator in the data. |
| ng_vector | Passed to ebseq. |
| rounds | Passed to ebseq. |
| target_fdr | Passed to ebseq. |
| norm | Normalization method of ebseq to apply. |
| force | Force inappropriate data into ebseq? |

## Value

EBSeq result table with some extra formatting.

| edger_pairwise | *Set up a model matrix and set of contrasts to do pairwise comparisons using EdgeR.* |
|---|---|

## Description

This function performs the set of possible pairwise comparisons using EdgeR.

## Usage

```
edger_pairwise(
  input = NULL,
  conditions = NULL,
  batches = NULL,
  model_cond = TRUE,
  model_batch = TRUE,
  model_intercept = FALSE,
  alt_model = NULL,
  extra_contrasts = NULL,
  annot_df = NULL,
  force = FALSE,
  edger_method = "long",
  ...
)
```

## Arguments

| | |
|---|---|
| input | Dataframe/vector or expt class containing data, normalization state, etc. |
| conditions | Factor of conditions in the experiment. |
| batches | Factor of batches in the experiment. |
| model_cond | Include condition in the experimental model? |
| model_batch | Include batch in the model? In most cases this is a good thing(tm). |
| model_intercept | |
| | Use an intercept containing model? |
| alt_model | Alternate experimental model to use? |
| extra_contrasts | |
| | Add some extra contrasts to add to the list of pairwise contrasts. This can be pretty neat, lets say one has conditions A,B,C,D,E and wants to do (C/B)/A and (E/D)/A or (E/D)/(C/B) then use this with a string like: "c_vs_b_ctrla = (C-B)-A, e_vs_d_ctrla = (E-D)-A, de_vs_cb = (E-D)-(C-B)," |
| annot_df | Annotation information to the data tables? |
| force | Force edgeR to accept inputs which it should not have to deal with. |
| edger_method | I found a couple/few ways of doing edger in the manual, choose with this. |
| ... | The elipsis parameter is fed to write_edger() at the end. |

**Details**

Like the other _pairwise() functions, this attempts to perform all pairwise contrasts in the provided data set. The details are of course slightly different when using EdgeR. Thus, this uses the function choose_binom_dataset() to try to ensure that the incoming data is appropriate for EdgeR (if one normalized the data, it will attempt to revert to raw counts, for example). It continues on to extract the conditions and batches in the data, choose an appropriate experimental model, and run the EdgeR analyses as described in the manual. It defaults to using an experimental batch factor, but will accept a string like 'sva' instead, in which case it will use sva to estimate the surrogates, and append them to the experimental design. The edger_method parameter may be used to apply different EdgeR code paths as outlined in the manual. If you want to play with non-standard data, the force argument will round the data and shoe-horn it into EdgeR.

**Value**

List including the following information: contrasts = The string representation of the contrasts performed. lrt = A list of the results from calling glmLRT(), one for each contrast. contrast_list = The list of each call to makeContrasts() I do this to avoid running into the limit on # of contrasts addressable by topTags() all_tables = a list of tables for the contrasts performed.

**See Also**

**edgeR**

**Examples**

```
## Not run:
 expt <- create_expt(metadata="metadata.xlsx", gene_info=annotations)
 pretend <- edger_pairwise(expt, model_batch="sva")

## End(Not run)
```

---

exclude_genes_expt          *Exclude some genes given a pattern match*

---

**Description**

Because I am too lazy to remember that expressionsets use matrix subsets for gene and sample. Also those methods lead to shenanigans when I want to know what happened to the data over the course of the subset.

**Usage**

```
exclude_genes_expt(
  expt,
  column = "txtype",
  method = "remove",
  ids = NULL,
```

```
    patterns = c("snRNA", "tRNA", "rRNA"),
    ...
)
```

## Arguments

| | |
|---|---|
| `expt` | Expressionset containing expt object. |
| `column` | fData column to use for subsetting. |
| `method` | Either remove explicit rows, or keep them. |
| `ids` | Specific IDs to exclude. |
| `patterns` | Character list of patterns to remove/keep |
| `...` | Extra arguments are passed to arglist, currently unused. |

## Value

A smaller expt

## See Also

[create_expt](create_expt)

## Examples

```
 ## Not run:
  all_expt <- create_expt(metadata)
  ## This assumes a column in the metadata named 'txtype' containing the
  ## information telling us what type of transcript each gene is.
  no_ribosomes <- exclude_genes_expt(all_expt, column="txtype",
                                     patterns=c("snRNA", "tRNA", "rRNA"))
  i_hate_these_genes <- exclude_genes_expt(all_expt, ids=c("gene1", "gene2"))
  only_ribosomes <- exclude_genes_expt(all_expt, method="keep")

## End(Not run)
```

---

expt                         *An expt is an ExpressionSet superclass with a shorter name.*

---

## Description

It is also a simple list so that one may summarize it more simply, provides colors and some slots to make one's life easier. It is created via the function create_expt() which perhaps should be changed.

## Usage

```
expt(...)
```

## Arguments

...                 Parameters for create_expt()

## Details

Another important caveat: expressionSets and their methods are all S4; but I did not want to write S4 methods, so I made my expt a S3 class. As a result, in order to make use of exprs, notes, pData, fData, and friends, I made use of setMethod() to set up calls for the expressionSet portion of the expt objects.

## Slots

`title` Title for the expressionSet.

`notes` Notes for the expressionSet (redundant with S4 notes()).

`design` Copy of the experimental metadata (redundant with pData()).

`annotation` Gene annotations (redundant with fData()).

`gff_file` filename of a gff file which feeds this data.

`state` What is the state of the data vis a vis normalization, conversion, etc.

`conditions` Usually the condition column from pData.

`batches` Usually the batch column from pData.

`libsize` Library sizes of the data in its current state.

`colors` Chosen colors for plotting the data.

`tximport` Data provided by tximport() to create the exprs() data.

---

extract_abundant_genes

> *Extract the sets of genes which are significantly more abundant than the rest.*

---

## Description

Given the output of something_pairwise(), pull out the genes for each contrast which are the most/least abundant. This is in contrast to extract_significant_genes(). That function seeks out the most changed, statistically significant genes.

## Usage

```
extract_abundant_genes(
  pairwise,
  according_to = "all",
  n = 200,
  z = NULL,
  unique = FALSE,
  least = FALSE,
  excel = "excel/abundant_genes.xlsx",
  ...
)
```

## Arguments

| | |
|---|---|
| pairwise | Output from _pairwise()(). |
| according_to | What tool(s) define 'most?' One may use deseq, edger, limma, basic, all. |
| n | How many genes to pull? |
| z | Instead take the distribution of abundances and pull those past the given z score. |
| unique | One might want the subset of unique genes in the top-n which are unique in the set of available conditions. This will attempt to provide that. |
| least | Instead of the most abundant, do the least. |
| excel | Excel file to write. |
| ... | Arguments passed into arglist. |

## Value

The set of most/least abundant genes by contrast/tool.

## See Also

**openxlsx**

---

extract_coefficient_scatter

*Perform a coefficient scatter plot of a limma/deseq/edger/basic table.*

---

## Description

Plot the gene abundances for two coefficients in a differential expression comparison. By default, genes past 1.5 z scores from the mean are colored red/green.

## Usage

```
extract_coefficient_scatter(
  output,
  toptable = NULL,
  type = "limma",
  x = 1,
  y = 2,
  z = 1.5,
  p = NULL,
  lfc = NULL,
  n = NULL,
  loess = FALSE,
  alpha = 0.4,
  color_low = "#DD0000",
  z_lines = FALSE,
  color_high = "#7B9F35",
  ...
)
```

## Arguments

| | |
|---|---|
| `output` | Result from the de_ family of functions, all_pairwise, or combine_de_tables(). |
| `toptable` | Chosen table to query for abundances. |
| `type` | Query limma, deseq, edger, or basic outputs. |
| `x` | The x-axis column to use, either a number of name. |
| `y` | The y-axis column to use. |
| `z` | Define the range of genes to color (FIXME: extend this to p-value and fold-change). |
| `p` | Set a p-value cutoff for coloring the scatter plot (currently not supported). |
| `lfc` | Set a fold-change cutoff for coloring points in the scatter plot (currently not supported.) |
| `n` | Set a top-n fold-change for coloring the points in the scatter plot (this should work, actually). |
| `loess` | Add a loess estimation (This is slow.) |
| `alpha` | How see-through to make the dots. |
| `color_low` | Color for the genes less than the mean. |
| `z_lines` | Add lines to show the z-score demarcations. |
| `color_high` | Color for the genes greater than the mean. |
| `...` | More arguments are passed to arglist. |

## See Also

**ggplot2** `plot_linear_scatter`

## Examples

```
## Not run:
 expt <- create_expt(metadata="some_metadata.xlsx", gene_info=annotations)
 pairwise_output <- all_pairwise(expt)
 scatter_plot <- extract_coefficient_scatter(pairwise_output,
                                              type="deseq", x="uninfected", y="infected")

## End(Not run)
```

---

| extract_de_plots | *Make a MA plot of some limma output with pretty colors and shapes.* |
|---|---|

---

## Description

Yay pretty colors and shapes! This function should be reworked following my rewrite of combine_de_tables(). It is certainly possible to make the logic here much simpler now.

**Usage**

```
extract_de_plots(
  pairwise,
  type = "edger",
  table = NULL,
  logfc = 1,
  p_type = "adj",
  p = 0.05,
  invert = FALSE,
  ...
)
```

**Arguments**

| | |
|---|---|
| pairwise | The result from all_pairwise(), which should be changed to handle other invocations too. |
| type | Type of table to use: deseq, edger, limma, basic. |
| table | Result from edger to use, left alone it chooses the first. |
| logfc | What logFC to use for the MA plot horizontal lines. |
| p_type | Adjusted or raw pvalues? |
| p | Cutoff to define 'significant' by p-value. |
| invert | Invert the plot? |
| ... | Extra arguments are passed to arglist. |

**Value**

a plot!

**See Also**

[plot_ma_de](plot_ma_de)

**Examples**

```
## Not run:
 prettyplot <- edger_ma(all_aprwise) ## [sic, I'm witty! and can speel]

## End(Not run)
```

---

extract_go *Extract a set of geneID to GOID mappings from a suitable data source.*

---

### Description

Like extract_lengths above, this is primarily intended to read gene ID and GO ID mappings from a OrgDb/OrganismDbi object.

### Usage

```
extract_go(db, metadf = NULL, keytype = "ENTREZID")
```

### Arguments

| | |
|---|---|
| db | Data source containing mapping information. |
| metadf | Data frame containing extant information. |
| keytype | Keytype used for querying |

### Value

Dataframe of 2 columns: geneID and goID.

### See Also

**AnnotationDbi**

---

extract_keepers_all *When no set of 'keeper' contrasts is specified, grab them all.*

---

### Description

This has a couple of cousin functions, extract_keepers_list and _single. These handle extracting one or more contrasts out of the various tables produced by all_pairwise().

### Usage

```
extract_keepers_all(
  apr,
  extracted,
  keepers,
  table_names,
  all_coefficients,
  limma,
  edger,
  ebseq,
```

```
    deseq,
    basic,
    adjp,
    annot_df,
    include_deseq,
    include_edger,
    include_ebseq,
    include_limma,
    include_basic,
    excludes,
    padj_type,
    loess = FALSE,
    lfc_cutoff = 1,
    p_cutoff = 0.05
)
```

## Arguments

| | |
|---|---|
| `apr` | Result from all_pairwise(). |
| `extracted` | Table of extracted data. |
| `keepers` | In this case, one may assume either NULL or 'all'. |
| `table_names` | The set of tables produced by all_pairwise(). |
| `all_coefficients` | |
| | The set of all experimental conditions in the experimental metadata. |
| `limma` | The limma data from all_pairwise(). |
| `edger` | The edger data from all_pairwise(). |
| `ebseq` | The ebseq data from all_pairwise(). |
| `deseq` | The deseq data from all_pairwise(). |
| `basic` | The basic data from all_pairwise(). |
| `adjp` | Pull out the adjusted p-values from the data? |
| `annot_df` | What annotations should be added to the table? |
| `include_deseq` | Whether or not to include the deseq data. |
| `include_edger` | Whether or not to include the edger data. |
| `include_ebseq` | Whether or not to include the ebseq data. |
| `include_limma` | Whether or not to include the limma data. |
| `include_basic` | Whether or not to include the basic data. |
| `excludes` | Set of genes to exclude. |
| `padj_type` | Choose a specific p adjustment. |
| `loess` | Include a loess estimator in the plots? |

---

extract_keepers_lst *When a list of 'keeper' contrasts is specified, extract it from the data.*

---

### Description

This is the most interesting of the extract_keeper functions. It must check that the numerators and denominators match the desired contrast and flip the signs in the logFCs when appropriate.

### Usage

```
extract_keepers_lst(
  extracted,
  keepers,
  table_names,
  all_coefficients,
  limma,
  edger,
  ebseq,
  deseq,
  basic,
  adjp,
  annot_df,
  include_deseq,
  include_edger,
  include_ebseq,
  include_limma,
  include_basic,
  excludes,
  padj_type,
  loess = FALSE,
  lfc_cutoff = 1,
  p_cutoff = 0.05
)
```

### Arguments

| | |
|---|---|
| extracted | Tables extracted from the all_pairwise data. |
| keepers | In this case, one may assume either NULL or 'all'. |
| table_names | The set of tables produced by all_pairwise(). |
| all_coefficients | |
| | The set of all experimental conditions in the experimental metadata. |
| limma | The limma data from all_pairwise(). |
| edger | The edger data from all_pairwise(). |
| ebseq | The ebseq data from all_pairwise(). |
| deseq | The deseq data from all_pairwise(). |

| basic | The basic data from all_pairwise(). |
|---|---|
| adjp | Pull out the adjusted p-values from the data? |
| annot_df | What annotations should be added to the table? |
| include_deseq | Whether or not to include the deseq data. |
| include_edger | Whether or not to include the edger data. |
| include_ebseq | Whether or not to include the ebseq data. |
| include_limma | Whether or not to include the limma data. |
| include_basic | Whether or not to include the basic data. |
| excludes | Set of genes to exclude. |
| padj_type | Choose a specific p adjustment. |
| loess | Add a loess to plots? |
| apr | Result from all_pairwise() |

---

extract_keepers_single

*When a single 'keeper' contrast is specified, find and extract it.*

---

### Description

When a single 'keeper' contrast is specified, find and extract it.

### Usage

```
extract_keepers_single(
  apr,
  extracted,
  keepers,
  table_names,
  all_coefficients,
  limma,
  edger,
  ebseq,
  deseq,
  basic,
  adjp,
  annot_df,
  include_deseq,
  include_edger,
  include_ebseq,
  include_limma,
  include_basic,
  excludes,
  padj_type,
```

```
    loess = FALSE,
    lfc_cutoff = 1,
    p_cutoff = 0.05
)
```

## Arguments

| | |
|---|---|
| `apr` | Data from all_pairwise(). |
| `extracted` | Tables extracted in combine_de_tables(). |
| `keepers` | In this case, one may assume either NULL or 'all'. |
| `table_names` | The set of tables produced by all_pairwise(). |
| `all_coefficients` | |
| | The set of all experimental conditions in the experimental metadata. |
| `limma` | The limma data from all_pairwise(). |
| `edger` | The edger data from all_pairwise(). |
| `ebseq` | The ebseq data from all_pairwise(). |
| `deseq` | The deseq data from all_pairwise(). |
| `basic` | The basic data from all_pairwise(). |
| `adjp` | Pull out the adjusted p-values from the data? |
| `annot_df` | What annotations should be added to the table? |
| `include_deseq` | Whether or not to include the deseq data. |
| `include_edger` | Whether or not to include the edger data. |
| `include_ebseq` | Whether or not to include the ebseq data. |
| `include_limma` | Whether or not to include the limma data. |
| `include_basic` | Whether or not to include the basic data. |
| `excludes` | Set of genes to exclude. |
| `padj_type` | Choose a specific p adjustment. |
| `loess` | Add a loess to plots? |

---

| `extract_lengths` | *Take gene/exon lengths from a suitable data source (gff/TxDb/OrganismDbi)* |
|---|---|

---

## Description

Primarily goseq, but also other tools on occasion require a set of gene IDs and lengths. This function is resposible for pulling that data from either a gff, or TxDb/OrganismDbi.

**Usage**

```
extract_lengths(
  db = NULL,
  gene_list = NULL,
  type = "GenomicFeatures::transcripts",
  id = "TXID",
  possible_types = c("GenomicFeatures::genes", "GenomicFeatures::cds",
    "GenomicFeatures::transcripts"),
  ...
)
```

**Arguments**

| | |
|---|---|
| `db` | Object containing data, if it is a string then a filename is assumed to a gff file. |
| `gene_list` | Set of genes to query. |
| `type` | Function name used for extracting data from TxDb objects. |
| `id` | Column from the resulting data structure to extract gene IDs. |
| `possible_types` | Character list of types I have previously used. |
| `...` | More arguments are passed to arglist. |

**Value**

Dataframe containing 2 columns: ID, length

**See Also**

**GenomicFeatures**

---

| | |
|---|---|
| `extract_mayu_pps_fdr` | *Read output from mayu to get the IP/PP number corresponding to a given FDR value.* |

---

**Description**

Read output from mayu to get the IP/PP number corresponding to a given FDR value.

**Usage**

```
extract_mayu_pps_fdr(file, fdr = 0.01)
```

**Arguments**

| | |
|---|---|
| `file` | Mayu output file. |
| `fdr` | Chosen fdr value to acquire. |

**Value**

List of two elements: the full mayu table sorted by fdr and the number corresponding to the chosen fdr value.

---

extract_metadata                *Pull metadata from a table (xlsx/xls/csv/whatever)*

---

**Description**

I find that when I acquire metadata from a paper or collaborator, annoyingly often there are many special characters or other shenanigans in the column names. This function performs some simple sanitizations. In addition, if I give it a filename it calls my generic 'read_metadata()' function before sanitizing.

**Usage**

```
extract_metadata(metadata, id_column = "sampleid", ...)
```

**Arguments**

| | |
|---|---|
| metadata | file or df of metadata |
| id_column | Column in the metadat containing the sample names. |
| ... | Arguments to pass to the child functions (read_csv etc). |

**Value**

Metadata dataframe hopefully cleaned up to not be obnoxious.

**Examples**

```
## Not run:
 sanitized <- extract_metadata("some_random_supplemental.xls")
 saniclean <- extract_metadata(some_goofy_df)

## End(Not run)
```

extract_msraw_data          *Read a bunch of mzXML files to acquire their metadata.*

### Description

I have had difficulties getting the full set of correct parameters for a DDA/DIA experiment. After some poking, I eventually found most of these required prameters in the mzXML raw files. Ergo, this function uses them. 20190310: I had forgotten about the mzR library. I think much (all?) of this is redundant with respect to it and perhaps should be removed in deference to the more complete and fast implementation included in mzR.

### Usage

```
extract_msraw_data(
  metadata,
  write_windows = TRUE,
  id_column = "sampleid",
  file_column = "raw_file",
  allow_window_overlap = FALSE,
  start_add = 0,
  format = "mzXML",
  parallel = TRUE,
  savefile = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| metadata | Data frame describing the samples, including the mzXML filenames. |
| write_windows | Write out SWATH window frames. |
| id_column | What column in the sample sheet provides the ID for the samples? |
| file_column | Which column in the sample sheet provides the filenames? |
| allow_window_overlap | |
| | What it says on the tin, some tools do not like DIA windows to overlap, if TRUE, this will make sure each annotated window starts at the end of the previous window if they overlap. |
| start_add | Another strategy is to just add a static amount to each window. |
| format | Currently this handles mzXML or mzML files. |
| parallel | Perform operations using an R foreach cluster? |
| savefile | If not null, save the resulting data structure to an rda file. |
| ... | Extra arguments, presumably color palettes and column names and stuff like that. |

### Value

List of data extracted from every sample in the MS run (DIA or DDA).

---

extract_mzML_scans *Parse a mzML file and return the relevant data.*

---

### Description

This does the actual work for extract_scan_data(). This levers mzR to provide the data and goes a step further to pull out the windows acquired in the MS/MS scan and print them in formats acceptable to TPP/OpenMS (eg. with and without headers).

### Usage

```
extract_mzML_scans(
  file,
  id = NULL,
  write_acquisitions = TRUE,
  allow_window_overlap = FALSE,
  start_add = 0
)
```

### Arguments

| | |
|---|---|
| file | Input mzML file to parse. |
| id | Chosen ID for the given file. |
| write_acquisitions | |
| | Write acquisition windows. |
| allow_window_overlap | |
| | Some downstream tools cannot deal with overlapping windows. Toggle that here. |
| start_add | Other downstream tools appear to expect some padding at the beginning of each window. Add that here. |

### Value

The list of metadata, scan data, etc from the mzXML file.

---

extract_mzXML_scans *Parse a mzXML file and return the relevant data.*

---

### Description

This does the actual work for extract_scan_data(). When I wrote this function, I had forgotten about the mzR library; with that in mind, this seems to give a bit more information and be a bit faster than my short tests with mzR (note however that my tests were to compare mzR parsing mzML files vs. this function with mzXML, which is a classic apples to oranges).

## Usage

```
extract_mzXML_scans(
  file,
  id = NULL,
  write_acquisitions = TRUE,
  allow_window_overlap = FALSE,
  start_add = 0
)
```

## Arguments

| | |
|---|---|
| `file` | Input mzXML file to parse. |
| `id` | Chosen ID for the given file. |
| `write_acquisitions` | |
| | Write acquisition windows. |
| `allow_window_overlap` | |
| | Some downstream tools cannot deal with overlapping windows. Toggle that here. |
| `start_add` | Other downstream tools appear to expect some padding at the beginning of each window. Add that here. |

## Details

This goes a step further to pull out the windows acquired in the MS/MS scan and print them in formats acceptable to TPP/OpenMS (eg. with and without headers).

## Value

The list of metadata, scan data, etc from the mzXML file.

---

```
extract_peprophet_data
```
                              *Get some data from a peptideprophet run.*

---

## Description

I am not sure what if any parameters this should have, but it seeks to extract the useful data from a peptide prophet run. In the situation in which I wish to use it, the input command was: > xinteract -dDECOY_ -OARPpd -Nfdr_library.xml comet_result.pep.xml Eg. It is a peptideprophet result provided by TPP. I want to read the resulting xml table and turn it into a data.table so that I can plot some metrics from it.

## Usage

```
extract_peprophet_data(pepxml, decoy_string = "DECOY_", ...)
```

**Arguments**

pepxml          The file resulting from the xinteract invocation.

decoy_string    What prefix do decoys have in the data.

...             Catch extra arguments passed here, currently unused.

**Value**

data table of all the information I saw fit to extract The columns are: * protein: The name of the matching sequence (DECOYs allowed here) * decoy: TRUE/FALSE, is this one of our decoys? * peptide: The sequence of the matching spectrum. * start_scan: The scan in which this peptide was observed * end scan: Ibid * index This seems to just increment * precursor_neutral_mass: Calculated mass of this fragment assuming no isotope shenanigans (yeah, looking at you C13). * assumed_charge: The expected charge state of this peptide. * retention_time_sec: The time at which this peptide eluted during the run. * peptide_prev_aa: The amino acid before the match. * peptide_next_aa: and the following amino acid. * num_tot_proteins: The number of matches not counting decoys. * num_matched_ions: How many ions for this peptide matched? * tot_num_ions: How many theoretical ions are in this fragment? * matched_ion_ratio: num_matched_ions / tot_num_ions, bigger is better! * cal_neutral_pep_mass: This is redundant with precursor_neutral_mass, but recalculated by peptideProphet, so if there is a discrepency we should yell at someone! * massdiff How far off is the observed mass vs. the calculated? (also redundant with massd later) * num_tol_term: The number of peptide termini which are consistent with the cleavage (hopefully 2), but potentially 1 or even 0 if digestion was bad. (redundant with ntt later) * num_missed_cleavages: How many cleavages must have failed in order for this to be a good match? * num_matched_peptides: Number of alternate possible peptide matches. * xcorr: cross correlation of the experimental and theoretical spectra (this is supposedly only used by sequest, but I seem to have it here...) * deltacn: The normalized difference between the xcorr values for the best hit and next best hit. Thus higher numbers suggest better matches. * deltacnstar: Apparently 'important for things like phospho-searches containing homologous top-scoring peptides when analyzed by peptideprophet...' – the comet release notes. * spscore: The raw value of preliminary score from the sequest algorithm. * sprank: The rank of the match in a preliminary score. 1 is good. * expect: E-value of the given peptide hit. Thus how many identifications one expect to observe by chance, lower is therefore better * prophet_probability: The peptide prophet probability score, higher is better. * fval: 0.6(the dot function + 0.4(the delta dot function) - (the dot bias penalty function) – which is to say... well I dunno, but it is supposed to provide information about how similar this match is to other potential matches, so I presume higher means the match is more ambiguous. * ntt: Redundant with num_tol_term above, but this time from peptide prophet. * nmc: Redundant with num_missed_cleavages, except it coalesces them. * massd: Redundant with massdiff * isomassd: The mass difference, but taking into account stupid C13. * RT: Retention time * RT_score: The score of the retention time! * modified_peptides: A string describing modifications in the found peptide * variable_mods: A comma separated list of the variable modifications observed. * static_mods: A comma separated list of the static modifications observed.

---

extract_pyprophet_data

*Read a bunch of scored swath outputs from pyprophet to acquire their metrics.*

---

## Description

This function is mostly cribbed from the other extract_ functions in this file. With it, I hope to be able to provide some metrics of a set of openswath runs, thus potentially opening the door to being able to objectively compare the same run with different options and/or different runs.

## Usage

```
extract_pyprophet_data(
  metadata,
  pyprophet_column = "diascored",
  savefile = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| metadata | Data frame describing the samples, including the mzXML filenames. |
| pyprophet_column | |
| | Which column from the metadata provides the requisite filenames? |
| savefile | If not null, save the data from this to the given filename. |
| ... | Extra arguments, presumably color palettes and column names and stuff like that. |

## Details

Likely columns generated by exporting OpenMS data via pyprophet include: transition_group_id: Incrementing ID of the transition in the MS(.pqp) library used for matching (I am pretty sure). decoy: Is this match of a decoy peptide? run_id: This is a bizarre encoding of the run, OpenMS/pyprophet re-encodes the run ID from the filename to a large signed integer. filename: Which raw mzXML file provides this particular intensity value? rt: Retention time in seconds for the matching peak group. assay_rt: The expected retention time after normalization with the iRT. (how does the iRT change this value?) delta_rt: The difference between rt and assay_rt irt: (As described in the abstract of Claudia Escher's 2012 paper: "Here we present iRT, an empirically derived dimensionless peptide-specific value that allows for highly accurate RT prediction. The iRT of a peptide is a fixed number relative to a standard set of reference iRT-peptides that can be transferred across laboratories and chromatographic systems.") assay_irt: The iRT observed in the actual chromatographic run. delta_irt: The difference. I am seeing that all the delta iRTs are in the -4000 range for our actual experiment; since this is in seconds, does that mean that it is ok as long as they stay in a similar range? id: unique long signed integer for the peak group. sequence: The sequence of the matched peptide fullunimodpeptidename: The sequence, but with unimod formatted modifications included. charge: The assumed charge of the observed peptide. mz: The m/z value of the precursor ion. intensity: The sum of all transition intensities in the peak group. aggr_prec_peak_area: Semi-colon separated list of intensities (peak areas) of the MS traces for this match. aggr_prec_peak_apex: Intensity peak apexes of the MS1 traces. leftwidth: The start of the peak group in seconds. rightwidth: The end of the peak group in seconds. peak_group_rank: When multiple peak groups match, which one is this? d_score: I think this is the score as retured by openMS (higher is better). m_score: I am pretty sure this is the result of a SELECT QVALUE operation in pyprophet. aggr_peak_area: The intensities of this fragment ion separated by semicolons. aggr_peak_apex: The intensities of this

fragment ion separated by semicolons. aggr_fragment_annotation: Annotations of the fragment ion traces by semicolon. proteinname: Name of the matching protein. m_score_protein_run_specific: I am guessing the fdr for the pvalue for this run. mass: Mass of the observed fragment.

## Value

A list of data from each sample in the pyprophet scored DIA run.

---

| extract_scan_data | *Read a mzML/mzXML file and extract from it some important metadata.* |
|---|---|

---

## Description

When working with swath data, it is fundamentally important to know the correct values for a bunch of the input variables. These are not trivial to acquire. This function attempts to make this easier (but slow) by reading the mzXML file and parsing out helpful data.

## Usage

```
extract_scan_data(
  file,
  id = NULL,
  write_acquisitions = TRUE,
  format = "mzXML",
  allow_window_overlap = FALSE,
  start_add = 0
)
```

## Arguments

| | |
|---|---|
| `file` | Filename to read. |
| `id` | An id to give the result. |
| `write_acquisitions` | |
| | If a filename is provided, write a tab separated table of windows. |
| `format` | Either mzXML or mzML. |
| `allow_window_overlap` | |
| | One may choose to foce windows to not overlap. |
| `start_add` | Add a minute to the start of the windows to avoid overlaps? |

## Value

List containing a table of scan and precursor data.

extract_siggenes                    *Alias for extract_significant_genes because I am dumb.*

### Description

Alias for extract_significant_genes because I am dumb.

### Usage

```
extract_siggenes(...)
```

### Arguments

| | |
|---|---|
| ... | The parameters for extract_significant_genes() |

### Value

It should return a reminder for me to remember my function names or change them to something not stupid.

---

extract_significant_genes

                    *Extract the sets of genes which are significantly up/down regulated*
                    *from the combined tables.*

---

### Description

Given the output from combine_de_tables(), extract the genes in which we have the greatest likely interest, either because they have the largest fold changes, lowest p-values, fall outside a z-score, or are at the top/bottom of the ranked list.

### Usage

```
extract_significant_genes(
  combined,
  according_to = "all",
  lfc = 1,
  p = 0.05,
  sig_bar = TRUE,
  z = NULL,
  n = NULL,
  top_percent = NULL,
  ma = TRUE,
  p_type = "adj",
  invert_barplots = FALSE,
```

```
    excel = NULL,
    siglfc_cutoffs = c(0, 1, 2),
    ...
)
```

## Arguments

| | |
|---|---|
| combined | Output from combine_de_tables(). |
| according_to | What tool(s) decide 'significant?' One may use the deseq, edger, limma, basic, meta, or all. |
| lfc | Log fold change to define 'significant'. |
| p | (Adjusted)p-value to define 'significant'. |
| sig_bar | Add bar plots describing various cutoffs of 'significant'? |
| z | Z-score to define 'significant'. |
| n | Take the top/bottom-n genes. |
| top_percent | Use a percentage to get the top-n genes. |
| ma | Add ma plots to the sheets of 'up' genes? |
| p_type | use an adjusted p-value? |
| invert_barplots | |
| | Invert the significance barplots as per Najib's request? |
| excel | Write the results to this excel file, or NULL. |
| siglfc_cutoffs | Set of cutoffs used to define levels of 'significant.' |
| ... | Arguments passed into arglist. |

## Value

The set of up-genes, down-genes, and numbers therein.

## See Also

[combine_de_tables](combine_de_tables)

---

| factor_rsquared | *Collect the r^2 values from a linear model fitting between a singular value decomposition and factor.* |
|---|---|

---

## Description

Collect the r^2 values from a linear model fitting between a singular value decomposition and factor.

## Usage

```
factor_rsquared(datum, fact, type = "factor")
```

## Arguments

| | |
|---|---|
| datum | Result from corpcor::fast.svd. |
| fact | Experimental factor from the original data. |
| type | Make this categorical or continuous with factor/continuous. |

## Value

The r^2 values of the linear model as a percentage.

## See Also

**corpcor** `fast.svd`

---

features_greater_than       *Count the number of features(genes) greater than x in a data set.*

---

## Description

Sometimes I am asked how many genes have >= x counts. Well, here you go.

## Usage

```
features_greater_than(data, cutoff = 1, hard = TRUE, inverse = FALSE)
```

## Arguments

| | |
|---|---|
| data | Dataframe/exprs/matrix/whatever of counts. |
| cutoff | Minimum number of counts. |
| hard | Greater-than is hard, greater-than-equals is not. |
| inverse | when inverted, this provides features less than the cutoff. |

## Details

Untested as of 2016-12-01 but used with Lucia. I think it would be interesting to iterate this function from small to large cutoffs and plot how the number of kept genes decreases.

## Value

A list of two elements, the first comprised of the number of genes greater than the cutoff, the second with the identities of said genes.

## See Also

**Biobase**

## Examples

```
## Not run:
 features <- features_greater_than(expt)
 fewer <- features_greater_than(expt, cutoff=100)

## End(Not run)
```

---

features_in_single_condition

> *I want an easy way to answer the question: what features are in only condition x?*

---

## Description

The answer to this lies in a combination of subset_expt() and features_greater_than().

## Usage

```
features_in_single_condition(
  expt,
  cutoff = 2,
  factor = "condition",
  chosen = NULL
)
```

## Arguments

| | |
|---|---|
| expt | An experiment to query. |
| cutoff | What is the minimum number of counts required to define 'included.' |
| factor | What metadata factor to query? |
| chosen | Either choose a subset or all conditions to query. |

## Value

A set of features.

## Examples

```
## Not run:
 unique_genes

## End(Not run)
```

---

features_less_than          *Do features_greater_than() inverted!*

---

### Description

Do features_greater_than() inverted!

### Usage

```
features_less_than(...)
```

### Arguments

|  |  |
|---|---|
| ... | Arguments passed to features_greather_than() |

### Value

The set of features less than whatever you would have done with features_greater_than().

### See Also

[features_greater_than](#)

---

filter_counts          *Call various count filters.*

---

### Description

This calls the various filtering functions in genefilter along with suggestions made in our lab meetings; defaulting to the threshold based filter suggested by Hector.

### Usage

```
filter_counts(
  count_table,
  method = "cbcb",
  p = 0.01,
  A = 1,
  k = 1,
  cv_min = 0.01,
  cv_max = 1000,
  thresh = 2,
  min_samples = 2,
  ...
)
```

## Arguments

| | |
|---|---|
| `count_table` | Some counts to filter. |
| `method` | Filtering method to apply (cbcb, pofa, kofa, cv right now). |
| `p` | Used by genefilter's pofa(). |
| `A` | Also for pofa(). |
| `k` | Used by genefilter's kofa(). |
| `cv_min` | Used by genefilter's cv(). |
| `cv_max` | Also used by cv(). |
| `thresh` | Minimum threshold across samples for cbcb. |
| `min_samples` | Minimum number of samples for cbcb. |
| `...` | More options might be needed, especially if I fold cv/p/etc into ... |

## Value

Data frame of filtered counts.

## See Also

**genefilter**

## Examples

```
## Not run:
 new <- filter_counts(old)

## End(Not run)
```

---

find_working_mart          *Find a functional biomart instance.*

---

## Description

In my experience, the various biomart mirrors are not varyingly likely to be functional at any given time. In addition, I often find it useful to use an archive instance rather than the most recent ensembl instance. This function therefore iterates over the various mirrors; or if archive=TRUE it will try a series of archive servers from 1, 2, and 3 years ago.

## Usage

```
find_working_mart(
  default_hosts = c("useast.ensembl.org", "uswest.ensembl.org", "www.ensembl.org",
    "asia.ensembl.org"),
  trymart = "ENSEMBL_MART_ENSEMBL",
  archive = FALSE,
  year = NULL,
  month = NULL
)
```

## Arguments

| | |
|---|---|
| `default_hosts` | List of biomart mirrors. |
| `trymart` | Specific mart to query. |
| `archive` | Try an archive server instead of a mirror? If this is a character, it will assume it is a specific archive hostname. |
| `year` | Choose specific year(s) for the archive servers? |
| `month` | Choose specific month(s) for the archive servers? |

## Value

Either a mart instance or NULL if no love was forthcoming.

---

| | |
|---|---|
| `flanking_sequence` | *Extract sequence flanking a set of annotations (generally coding sequences)* |

---

## Description

Given a set of annotations and genome, one might want to get the set of adjacent sequences.

## Usage

```
flanking_sequence(
  bsgenome,
  annotation,
  distance = 200,
  type = "gene",
  prefix = ""
)
```

## Arguments

| | |
|---|---|
| `bsgenome` | Genome sequence |
| `annotation` | Set of annotations |
| `distance` | How far from each annotation is desired? |
| `type` | What type of annotation is desired? |
| `prefix` | Provide a prefix to the names to distinguish them from the existing annotations. |

## Value

A list of sequences before and after each sequence.

gather_eupath_utrs_padding
*Given an eupathdb species lacking UTR boundaries, extract an arbitrary region before/after each gene.*

## Description

This is a very domain-specific function.

## Usage

```
gather_eupath_utrs_padding(
  species_name = "Leishmania major",
  entry = NULL,
  webservice = "tritrypdb",
  padding = 200,
  ...
)
```

## Arguments

| | |
|---|---|
| species_name | Species name for which to query the eupathdb. |
| entry | EuPathDB metadatum entry. |
| webservice | If specified, makes the query faster, I always used tritrypdb.org. |
| padding | Number of nucleotides to gather. |
| ... | Extra arguments for the various EuPathDB functions. |

## Value

Set of padding UTR sequences/coordinates.

gather_genes_orgdb *Use the orgdb instances from clusterProfiler to gather annotation data for GO.*

## Description

Since clusterprofiler no longer builds gomaps, I need to start understanding how to properly get information from orgDBs.

## Usage

```
gather_genes_orgdb(goseq_data, orgdb_go, orgdb_ensembl)
```

## Arguments

| | |
|---|---|
| `goseq_data` | Some data from goseq and friends. |
| `orgdb_go` | The orgDb instance with GO data. |
| `orgdb_ensembl` | The orgDb instance with ensembl data. |

## Value

a go mapping

## See Also

**clusterProfiler**

---

| `gather_masses` | *Use BRAIN to find the peptide mass from a sequence.* |
|---|---|

---

## Description

This rounds the avgMass from BRAIN to deal with isotopes, maybe this should be changed.

## Usage

```
gather_masses(sequence)
```

## Arguments

| | |
|---|---|
| `sequence` | Sequence to count. |

## Value

Rounded average mass.

---

| `gather_ontology_genes` | *Given a set of goseq data from simple_goseq(), make a list of genes represented in each ontology.* |
|---|---|

---

## Description

This function uses the GO2ALLEG data structure to reverse map ontology categories to a list of genes represented. It therefore assumes that the GO2ALLEG.rda data structure has been deposited in pwd(). This in turn may be generated by clusterProfilers buildGOmap() function if it doesn't exist. For some species it may also be auto-generated. With little work this can be made much more generic, and it probably should.

## Usage

```
gather_ontology_genes(
  result,
  ontology = NULL,
  column = "over_represented_pvalue",
  pval = 0.1,
  include_all = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `result` | List of results as generated by simple_*(). |
| `ontology` | Ontology to search (MF/BP/CC). |
| `column` | Which column to use for extracting ontologies? |
| `pval` | Maximum accepted pvalue to include in the list of categories to cross reference. |
| `include_all` | Include all genes in the ontology search? |
| `...` | Extra options without a purpose just yet. |

## Value

Data frame of categories/genes.

## See Also

**goseq clusterProfiler** [`simple_goseq`](#)

## Examples

```
 ## Not run:
  data <- simple_goseq(sig_genes=limma_output, lengths=annotation_df, goids=goids_df)
  genes_in_cats <- gather_genes(data, ont='BP')

## End(Not run)
```

---

| gather_utrs_padding | *Take a BSgenome and data frame of chr/start/end/strand, provide 5' and 3' padded sequence.* |
|---|---|

---

## Description

For some species, we do not have a fully realized set of UTR boundaries, so it can be useful to query some arbitrary and consistent amount of sequence before/after every CDS sequence. This function can provide that information.

## Usage

```
gather_utrs_padding(
  bsgenome,
  annot_df,
  gid = NULL,
  name_column = "gid",
  chr_column = "chromosome",
  start_column = "start",
  end_column = "end",
  strand_column = "strand",
  type_column = "annot_gene_type",
  gene_type = "protein coding",
  padding = 120,
  ...
)
```

## Arguments

| | |
|---|---|
| bsgenome | BSgenome object containing the genome of interest. |
| annot_df | Annotation data frame containing all the entries of interest, this is generally extracted using a function in the load_something_annotations() family (load_orgdb_annotations() being the most likely). |
| gid | Specific GID(s) to query. |
| name_column | Give each gene a name using this column. |
| chr_column | Column name of the chromosome names. |
| start_column | Column name of the start information. |
| end_column | Ibid, end column. |
| strand_column | Ibid, strand. |
| type_column | Subset the annotation data using this column, if not null. |
| gene_type | Subset the annotation data using the type_column with this type. |
| padding | Return this number of nucleotides for each gene. |
| ... | Arguments passed to child functions (I think none currently). |

## Value

Dataframe of UTR, CDS, and UTR+CDS sequences.

gather_utrs_txdb *Get UTR sequences using information provided by TxDb and fiveU-TRsByTranscript*

## Description

For species like Mus musculus, load_orgdb_annotations(Mus.musculus) should return a list including the requisite GRanges for the 5'/3' UTRs.

## Usage

```
gather_utrs_txdb(
  bsgenome,
  fivep_utr = NULL,
  threep_utr = NULL,
  start_column = "start",
  end_column = "end",
  strand_column = "strand",
  chr_column = "seqnames",
  name_column = "group_name",
  ...
)
```

## Arguments

| | |
|---|---|
| bsgenome | A BSGenome instance containing the encoded genome. |
| fivep_utr | Locations of the 5' UTRs. |
| threep_utr | Locations of the 3' UTRs. |
| start_column | What column in the annotation data contains the starts? |
| end_column | Column in the data with the end locations. |
| strand_column | What column in the annotation data contains the sequence strands? |
| chr_column | Column in the df with the chromosome names. |
| name_column | Finally, where are the gene names? |
| ... | Parameters passed to child functions. |

## Value

UTRs!

---

gbk_annotations          *Extract some useful information from a gbk imported as a txDb.*

---

### Description

This function no longer really stands on its own, but is more accessible from load_genbank_annotations().

### Usage

```
gbk_annotations(gbr)
```

### Arguments

gbr                TxDb object to poke at.

### Details

Tested in test_40ann_biomartgenbank.R This function should provide a quick reminder of how to use the AnnotationDbi select function if it does nothing else. It also (hopefully helpfully) returns a granges object containing the essential information one might want for printing out a gff or whatever.

I should revisit this function and improve the generated ranges objects to have better metadata columns via the mcols() function. For examples of some useful tasks one can do here, check out snp.r.

### Value

Granges data

### See Also

**AnnotationDbi GenomeInfoDb GenomicFeatures** select

---

genefilter_cv_counts    *Filter genes from a dataset outside a range of variance.*

---

### Description

This function from genefilter removes genes surpassing a variance cutoff. It is not therefore a low-count filter per se.

### Usage

```
genefilter_cv_counts(count_table, cv_min = 0.01, cv_max = 1000)
```

## Arguments

| | |
|---|---|
| count_table | Input data frame of counts by sample. |
| cv_min | Minimum coefficient of variance. |
| cv_max | Maximum coefficient of variance. |

## Value

Dataframe of counts without the high/low variance genes.

## See Also

**genefilter** [kOverA](#)

## Examples

```
## Not run:
 filtered_table = genefilter_kofa_counts(count_table)

## End(Not run)
```

---

genefilter_kofa_counts

*Filter low-count genes from a data set using genefilter's kOverA().*

---

## Description

This is the most similar to the function suggested by Hector I think.

## Usage

```
genefilter_kofa_counts(count_table, k = 1, A = 1)
```

## Arguments

| | |
|---|---|
| count_table | Input data frame of counts by sample. |
| k | Minimum number of samples to have >A counts. |
| A | Minimum number of counts for each gene's sample in kOverA(). |

## Value

Dataframe of counts without the low-count genes.

## See Also

**genefilter** [kOverA](#)

## Examples

```
## Not run:
 filtered_table = genefilter_kofa_counts(count_table)

## End(Not run)
```

genefilter_pofa_counts

*Filter low-count genes from a data set using genefilter's pOverA().*

## Description

I keep thinking this function is pofa... oh well. Of the various tools in genefilter, this one to me is the most intuitive. Take the ratio of counts/samples and make sure it is >= a score.

## Usage

```
genefilter_pofa_counts(count_table, p = 0.01, A = 100)
```

## Arguments

| count_table | Input data frame of counts by sample. |
| p | Minimum proportion of each gene's counts/sample to be greater than a minimum(A). |
| A | Minimum number of counts in the above proportion. |

## Value

Dataframe of counts without the low-count genes.

## See Also

**genefilter** pOverA

## Examples

```
## Not run:
 filtered_table = genefilter_pofa_counts(count_table)

## End(Not run)
```

---

generate_expt_colors  *Set up default colors for a data structure containing usable metadata*

---

## Description

In theory this function should be useful in any context when one has a blob of metadata and wants to have a set of colors. Since my taste is utterly terrible, I rely entirely upon RColorBrewer, but also allow one to choose his/her own colors.

## Usage

```
generate_expt_colors(
  sample_definitions,
  cond_column = "condition",
  by = "sampleid",
  ...
)
```

## Arguments

sample_definitions

                   Metadata, presumably containing a 'condition' column.

| | |
|---|---|
| cond_column | Which column in the sample data provides the set of 'conditions' used to define the colors? |
| by | Name the factor of colors according to this column. |
| ... | Other arguments like a color palette, etc. |

## Value

Colors!

---

genoplot_chromosome  *Try plotting a chromosome (region)*

---

## Description

genoplotr is cool, I don't yet understand it though

## Usage

```
genoplot_chromosome(
  accession = "AE009949",
  start = NULL,
  end = NULL,
  title = "Genome plot"
)
```

## Arguments

| | |
|---|---|
| accession | An accession to plot, this will download it. |
| start | First segment to plot (doesn't quite work yet). |
| end | Final segment to plot (doesn't quite work yet). |
| title | Put a title on the resulting plot. |

## Value

Hopefully a pretty plot of a genome

## See Also

**genoPlotR**

---

| | |
|---|---|
| get_abundant_genes | *Find the set of most/least abundant genes according to limma and friends following a differential expression analysis.* |

---

## Description

Given a data set provided by limma, deseq, edger, etc; one might want to know what are the most and least abundant genes, much like get_sig_genes() does to find the most significantly different genes for each contrast.

## Usage

```
get_abundant_genes(
  datum,
  type = "limma",
  n = NULL,
  z = NULL,
  unique = FALSE,
  least = FALSE
)
```

## Arguments

| | |
|---|---|
| datum | Output from the _pairwise() functions. |
| type | Extract abundant genes according to what? |
| n | Perhaps take just the top/bottom n genes. |
| z | Or take genes past a given z-score. |
| unique | Unimplemented: take only the genes unique among the conditions surveyed. |
| least | When true, this finds the least abundant rather than most. |

**Value**

List of data frames containing the genes of interest.

**See Also**

**stats limma DESeq2 edgeR**

**Examples**

```
## Not run:
 abundant <- get_abundant_genes(all_pairwise_output, type="deseq", n=100)
 ## Top 100 most abundant genes from deseq
 least <- get_abundant_genes(all_pairwise_output, type="deseq", n=100, least=TRUE)
 ## Top 100 least abundant genes from deseq
 abundant <- get_abundant_genes(all_pairwise_output, type="edger", z=1.5)
 ## Get the genes more than 1.5 standard deviations from the mean.

## End(Not run)
```

---

get_genesizes *Grab gene length/width/size from an annotation database.*

---

**Description**

This function tries to gather an appropriate gene length column from whatever annotation data source is provided.

**Usage**

```
get_genesizes(
  annotation = NULL,
  type = "gff",
  gene_type = "gene",
  type_column = "type",
  key = NULL,
  length_names = NULL,
  ...
)
```

**Arguments**

| | |
|---|---|
| annotation | There are a few likely data sources when getting gene sizes, choose one with this. |
| type | What type of annotation data are we using? |
| gene_type | Annotation type to use (3rd column of a gff file). |
| type_column | Type identifier (10th column of a gff file). |
| key | What column has ID information? |

length_names     Provide some column names which give gene length information?

...              Extra arguments likely for load_annotations()

## Value

Data frame of gene IDs and widths.

## See Also

**rtracklayer** `load_gff_annotations`

## Examples

```
pa_genesizes <- get_genesizes(gff=gff_file)
head(pa_genesizes)
```

---

get_git_commit              *Get the current git commit for hpgltools*

---

## Description

One might reasonably ask about this function: "Why?" I invoke this function at the end of my various knitr documents so that if necessary I can do a > git reset <commit id> and get back to the exact state of my code.

## Usage

```
get_git_commit(gitdir = "~/hpgltools")
```

## Arguments

gitdir           Directory containing the git repository.

---

get_gsvadb_names             *Extract the GeneSets corresponding to the provided name(s).*

---

## Description

Many of the likely GSCs contain far more gene sets than one actually wants to deal with. This will subset them according to a the desired 'requests'.

## Usage

```
get_gsvadb_names(sig_data, requests = NULL)
```

## Arguments

| | |
|---|---|
| `sig_data` | The pile of GeneSets, probably from GSVAdata. |
| `requests` | Character list of sources to keep. |

## Value

Whatever GeneSets remain.

---

| `get_individual_snps` | *Extract the observed snps unique to individual categories in a snp set.* |
|---|---|

---

## Description

The result of get_snp_sets provides sets of snps for all possible categories. This is cool and all, but most of the time we just want the results of a single group in that rather large set (2^number of categories)

## Usage

```
get_individual_snps(retlist)
```

## Arguments

| | |
|---|---|
| `retlist` | The result from get_snp_sets(). |

---

| `get_kegg_genes` | *Extract the set of geneIDs matching pathways for a given species.* |
|---|---|

---

## Description

This uses KEGGREST to extract the mappings for all genes for a species and pathway or 'all'. Because downloading them takes a while, it will save the results to kegg_species.rda. When run interactively, it will give some information regarding the number of genes observed in each pathway.

## Usage

```
get_kegg_genes(
  pathway = "all",
  abbreviation = NULL,
  species = "leishmania major",
  savefile = NULL
)
```

## Arguments

| | |
|---|---|
| pathway | Either a single pathway kegg id or 'all'. |
| abbreviation | Optional 3 letter species kegg id. |
| species | Stringified species name used to extract the 3 letter abbreviation. |
| savefile | Filename to which to save the relevant data. |

## Value

Dataframe of the various kegg data for each pathway, 1 row/gene.

## See Also

**KEGGREST**

## Examples

```
## Not run:
 kegg_info <- get_kegg_genes(species="Canis familiaris")

## End(Not run)
```

---

get_kegg_orgn                    *Search KEGG identifiers for a given species name.*

---

## Description

KEGG identifiers do not always make sense. For example, how am I supposed to remember that Leishmania major is lmj? This takes in a human readable string and finds the KEGG identifiers that match it.

## Usage

```
get_kegg_orgn(species = "Leishmania", short = TRUE)
```

## Arguments

| | |
|---|---|
| species | Search string (Something like 'Homo sapiens'). |
| short | Only pull the orgid? |

## Value

Data frame of possible KEGG identifier codes, genome ID numbers, species, and phylogenetic classifications.

## See Also

**RCurl**

## Examples

```
## Not run:
 fun = get_kegg_orgn('Canis')
 ## >    Tid    orgid    species                phylogeny
 ## > 17 T01007   cfa Canis familiaris (dog) Eukaryotes;Animals;Vertebrates;Mammals

 ## End(Not run)
```

---

get_kegg_sub                 *Provide a set of simple substitutions to convert geneIDs from KEGG->TriTryDB*

---

## Description

This function should provide 2 character lists which, when applied sequentially, will result in a hopefully coherent set of mapped gene IDs matching the TriTypDB/KEGG specifications.

## Usage

```
get_kegg_sub(species = "lma")
```

## Arguments

species          3 letter abbreviation for a given kegg type

## Value

2 character lists containing the patterns and replace arguments for gsub(), order matters!

## See Also

**KEGGREST**

---

get_microbesonline_taxid

*Extract microbesonline taxon IDs without having to click on the weird boxes at the top of the website.*

---

## Description

This should simplify getting material from microbesonline.

## Usage

```
get_microbesonline_taxid(species = "Acyrthosiphon pisum virus")
```

## Arguments

| | |
|---|---|
| species | String to search the set of microbesonline taxa. |

## Value

NULL or 1 or more taxon ids.

## Examples

```
coli_taxids <- get_microbesonline_taxid(species="coli S88")
head(coli_taxids)
```

---

get_msigdb_metadata        *Create a metadata dataframe of msigdb data, this hopefully will be usable to fill the fData slot of a gsva returned expressionset.*

---

## Description

Create a metadata dataframe of msigdb data, this hopefully will be usable to fill the fData slot of a gsva returned expressionset.

## Usage

```
get_msigdb_metadata(
  sig_data = NULL,
  msig_xml = "msigdb_v6.2.xml",
  gsva_result = NULL
)
```

## Arguments

| | |
|---|---|
| sig_data | GeneSetCollection from the broad msigdb. |
| msig_xml | msig XML file downloaded from broad. |
| gsva_result | Some data from GSVA to modify. |

## Value

list containing 2 data frames: all metadata from broad, and the set matching the sig_data GeneSets.

---

get_pairwise_gene_abundances
*A companion function for get_abundant_genes()*

---

## Description

Instead of pulling to top/bottom abundant genes, get all abundances and variances or stderr.

## Usage

```
get_pairwise_gene_abundances(datum, type = "limma", excel = NULL)
```

## Arguments

datum     Output from _pairwise() functions.

type      According to deseq/limma/ed ger/basic?

excel     Print this to an excel file?

## Value

A list containing the expression values and some metrics of variance/error.

## See Also

**limma**

## Examples

```
## Not run:
 abundance_excel <- get_pairwise_gene_abundances(combined, excel="abundances.xlsx")
 ## This should provide a set of abundances after voom by condition.

## End(Not run)
```

---

get_res     *Attempt to get residuals from tsne data*

---

## Description

I strongly suspect that this is not correct, but it is a start.

## Usage

```
get_res(
  svd_result,
  design,
  factors = c("condition", "batch"),
  res_slot = "v",
  var_slot = "d"
)
```

## Arguments

| | |
|---|---|
| svd_result | The set of results from one of the many potential svd-ish methods. |
| design | Experimental design from which to get experimental factors. |
| factors | Set of experimental factors for which to calculate rsquared values. |
| res_slot | Where is the res data in the svd result? |
| var_slot | Where is the var data in the svd result? |

---

get_sig_genes                  *Get a set of up/down differentially expressed genes.*

---

## Description

Take one or more criteria (fold change, rank order, (adj)p-value, z-score from median FC) and use them to extract the set of genes which are defined as 'differentially expressed.' If no criteria are provided, it arbitrarily chooses all genes outside of 1-z.

## Usage

```
get_sig_genes(
  table,
  n = NULL,
  z = NULL,
  lfc = NULL,
  p = NULL,
  column = "logFC",
  fold = "plusminus",
  p_column = "adj.P.Val"
)
```

## Arguments

| | |
|---|---|
| table | Table from limma/edger/deseq. |
| n | Rank-order top/bottom number of genes to take. |
| z | Number of z-scores >/< the median to take. |
| lfc | Fold-change cutoff. |

| p | P-value cutoff. |
|---|---|
| column | Table's column used to distinguish top vs. bottom. |
| fold | Identifier reminding how to get the bottom portion of a fold-change (plusminus says to get the negative of the positive, otherwise 1/positive is taken). This effectively tells me if this is a log fold change or not. |
| p_column | Table's column containing (adjusted or not)p-values. |

## Details

Tested in test_29de_shared.R

## Value

Subset of the up/down genes given the provided criteria.

## See Also

[extract_significant_genes](extract_significant_genes)

## Examples

```
## Not run:
 sig_table <- get_sig_genes(table, lfc=1)

## End(Not run)
```

---

```
get_sig_gsva_categories
```
                    *Attempt to score the results from simple_gsva()*

---

## Description

This function uses a couple of methods to try to get an idea of whether the results from gsva are actually interesting. It does so via the following methods: 1. Use limma on the expressionset returned by simple_gsva(), this might provide an idea of if there are changing signatures among the sample types. 2. Perform a simplified likelihood estimate to get a sense of the significant categories.

## Usage

```
get_sig_gsva_categories(
  gsva_result,
  cutoff = 0.05,
  excel = "excel/gsva_subset.xlsx",
  factor_column = "condition",
  factor = NULL
)
```

## Arguments

| | |
|---|---|
| `gsva_result` | Result from simple_gsva() |
| `cutoff` | Significance cutoff |
| `excel` | Excel file to write the results. |
| `factor_column` | When extracting significance information, use this metadata factor. |
| `factor` | Use this metadata factor as the reference. |

---

| | |
|---|---|
| `get_snp_sets` | *Create all possible sets of variants by sample (types).* |

---

## Description

I like this function. It generates an exhaustive catalog of the snps by chromosome for all the various categories as defined by factor.

## Usage

```
get_snp_sets(
  snp_expt,
  factor = "pathogenstrain",
  limit = 1,
  do_save = FALSE,
  savefile = "variants.rda"
)
```

## Arguments

| | |
|---|---|
| `snp_expt` | The result of count_expt_snps() |
| `factor` | Experimental factor to use for cutting and splicing the data. |
| `limit` | Minimum median number of hits / factor to define a position as a hit. |
| `do_save` | Save the result? |
| `savefile` | Prefix for a savefile if one chooses to save the result. |

## Value

A funky list by chromosome containing: 'medians', the median number of hits / position by sample type; 'possibilities', the; 'intersections', the groupings as detected by Vennerable; 'chr_data', the raw data; 'set_names', a character list of the actual names of the groupings; 'invert_names', the opposite of set_names which is to say the names of groups which do _not_ include samples x,y,z; 'density', a list of snp densities with respect to chromosomes. Note that this last one is approximate as I just calculate with the largest chromosome position number, not the explicit number of nucleotides in the chromosome.

## Examples

```
 ## Not run:
expt <- create_expt(metadata, gene_information)
snp_expt <- count_expt_snps(expt)
snp_sets <- get_snp_sets(snp_expt, factor="condition")
## This assumes a column in the metadata for the expt named 'condition'.

## End(Not run)
```

---

getEdgeWeights            *Plot the ontology DAG.*

---

## Description

This function was stolen from topgo in order to figure out where it was failing.

## Usage

```
getEdgeWeights(graph)
```

## Arguments

graph           Graph from topGO

## Value

Weights!

---

gff2irange            *Extract annotation information from a gff file into an irange object.*

---

## Description

Try to make import.gff a little more robust; I acquire (hopefully) valid gff files from various sources: yeastgenome.org, microbesonline, tritrypdb, ucsc, ncbi. To my eyes, they all look like reasonably good gff3 files, but some of them must be loaded with import.gff2, import.gff3, etc. That is super annoying. Also, I pretty much always just do as.data.frame() when I get something valid from rtracklayer, so this does that for me, I have another function which returns the iranges etc. This function wraps import.gff/import.gff3/import.gff2 calls in try() because sometimes those functions fail in unpredictable ways.

## Usage

```
gff2irange(gff, type = NULL)
```

**Arguments**

| | |
|---|---|
| gff | Gff filename. |
| type | Subset to extract. |

**Details**

This is essentially load_gff_annotations(), but returns data suitable for getSet() This is another place which should be revisited for improvements via mcols(). Check snp.r. for ideas.

**Value**

Iranges! (useful for getSeq().)

**See Also**

**rtracklayer** load_gff_annotations **Biostrings** import.gff

**Examples**

```
gff_file <- system.file("gas.gff", package="hpgltools")
gas_iranges <- gff2irange(gff_file)
colnames(as.data.frame(gas_iranges))
```

---

| ggplotly_url | *Add a little logic to ggplotly to simplify adding clicky link.* |
|---|---|

---

**Description**

There are some other ease of life improvements I have in a few of my plotly invocations which I should add here.

**Usage**

```
ggplotly_url(
  plot,
  filename,
  id_column = "id",
  title = NULL,
  url_data = NULL,
  url_column = "url",
  tooltip = "all"
)
```

## Arguments

| | |
|---|---|
| plot | Plot generated via ggplot2. |
| filename | filename to save the output html plot. |
| id_column | Column containing the gene IDs. |
| title | Provide a title for the generated html file. |
| url_data | Either a glue() string or column of urls. |
| url_column | Name of the column in the ggplot data containing the URLs. |
| tooltip | Passed to ggplotly(). |

## Value

plotly with clicky links.

---

| | |
|---|---|
| ggplt | *Simplify plotly ggplot conversion so that there are no shenanigans.* |

---

## Description

I am a fan of ggplotly, but its conversion to an html file is not perfect. This hopefully will get around the most likely/worst problems.

## Usage

```
ggplt(
  gg,
  filename = "ggplot.html",
  selfcontained = TRUE,
  libdir = NULL,
  background = "white",
  title = class(gg)[[1]],
  knitrOptions = list(),
  ...
)
```

## Arguments

| | |
|---|---|
| gg | Plot from ggplot2. |
| filename | Output filename. |
| selfcontained | htmlwidgets: Return the plot as a self-contained file with images re-encoded base64. |
| libdir | htmlwidgets: Directory into which to put dependencies. |
| background | htmlwidgets: String for the background of the image. |
| title | htmlwidgets: Title of the page! |
| knitrOptions | htmlwidgets: I am not a fan of camelCase, but nonetheless, options from knitr for htmlwidgets. |
| ... | Any remaining elipsis options are passed to ggplotly. |

## Value

The final output filename

---

godef                          *Get a go long-form definition from an id.*

---

## Description

Sometimes it is nice to be able to read the full definition of some GO terms.

## Usage

```
godef(go = "GO:0032432")
```

## Arguments

go                GO ID, this may be a character or list (assuming the elements are goids).

## Value

Some text providing the long definition of each provided GO id.

## See Also

**GOTermsAnnDbBimap**

## Examples

```
## Not run:
 godef("GO:0032432")
 ## > GO:0032432
 ## > "An assembly of actin filaments that are on the same axis but may be
 ## > same or opposite polarities and may be packed with different levels of tightness."

## End(Not run)
```

---

golev                *Get a go level approximation from an ID.*

---

### Description

Sometimes it is useful to know how far up/down the ontology tree a given id resides. This attmepts to answer that question.

### Usage

```
golev(go)
```

### Arguments

go                GO id, this may be a character or list (assuming the elements are goids).

### Value

Set of numbers corresponding to approximate tree positions of the GO ids.

### See Also

**GOTermsAnnDbBimap**

### Examples

```
## Not run:
 golev("GO:0032559")
 ## > 3

## End(Not run)
```

---

golevel              *Get a go level approximation from a set of IDs.*

---

### Description

This just wraps golev() in mapply.

### Usage

```
golevel(go = c("GO:0032559", "GO:0000001"))
```

### Arguments

go                Character list of IDs.

## Value

Set pf approximate levels within the onlogy.

## See Also

**GOTermsAnnDbBimap**

## Examples

```
## Not run:
 golevel(c("GO:0032559", "GO:0000001")
 ## > 3 4

## End(Not run)
```

---

golevel_df                          *Extract a dataframe of golevels using getGOLevel() from clusterPro-
                                     filer.*

---

## Description

This function is way faster than my previous iterative golevel function. That is not to say it is very
fast, so it saves the result to ontlevel.rda for future lookups.

## Usage

```
golevel_df(ont = "MF", savefile = "ontlevel.rda")
```

## Arguments

ont             the ontology to recurse.

savefile        a file to save the results for future lookups.

## Value

golevels a dataframe of goids<->highest level

## See Also

**clusterProfiler**

---

goont                           *Get a go ontology name from an ID.*

---

### Description

Get a go ontology name from an ID.

### Usage

```
goont(go = c("GO:0032432", "GO:0032433"))
```

### Arguments

go                 GO id, this may be a character or list (assuming the elements are goids).

### Value

The set of ontology IDs associated with the GO ids, thus 'MF' or 'BP' or 'CC'.

### See Also

**GOTermsAnnDbBimap**

### Examples

```
## Not run:
 goont(c("GO:0032432", "GO:0032433"))
 ## > GO:0032432 GO:0032433
 ## > "CC" "CC"

 ## End(Not run)
```

---

gosec                           *Get a GO secondary ID from an id.*

---

### Description

Unfortunately, GOTERM's returns for secondary IDs are not consistent, so this function has to have a whole bunch of logic to handle the various outputs.

### Usage

```
gosec(go = "GO:0032432")
```

### Arguments

go                 GO ID, this may be a character or list(assuming the elements, not names, are goids).

## Value

Some text comprising the secondary GO id(s).

## See Also

**GOTermsAnnDbBimap**

## Examples

```
## Not run:
 gosec("GO:0032432")
 ## > GO:0032432
 ## > "GO:0000141" "GO:0030482"

 ## End(Not run)
```

---

goseq_table                    *Enhance the goseq table of gene ontology information.*

---

## Description

While goseq has some nice functionality, the table of outputs it provides is somewhat lacking. This attempts to increase that with some extra helpful data like ontology categories, definitions, etc.

## Usage

```
goseq_table(df, file = NULL)
```

## Arguments

| | |
|---|---|
| df | Dataframe of ontology information. This is intended to be the output from goseq including information like numbers/category, GOids, etc. It requires a column 'category' which contains: GO:000001 and such. |
| file | Csv file to which to write the table. |

## Value

Ontology table with annotation information included.

## See Also

**goseq**

## Examples

```
## Not run:
 annotated_go = goseq_table(go_ids)
 head(annotated_go, n=1)
 ## >        category numDEInCat numInCat over_represented_pvalue
 ## > 571  GO:0006364          9       26            4.655108e-08
 ## >     under_represented_pvalue      qvalue ontology
 ## > 571               1.0000000 6.731286e-05       BP
 ## >                             term
 ## > 571               rRNA processing
 ## >                          synonym
 ## > 571       "35S primary transcript processing, GO:0006365"
 ## >     secondary    definition
 ## > 571   GO:0006365   Any process involved in the conversion of a primary ribosomal
 ##           RNA (rRNA) transcript into one or more mature rRNA molecules.

## End(Not run)
```

---

goseq_trees                    *Make fun trees a la topgo from goseq data.*

---

## Description

This seeks to force goseq data into a format suitable for topGO and then use its tree plotting function
to make it possible to see significantly increased ontology trees.

## Usage

```
goseq_trees(
  goseq,
  goid_map = "id2go.map",
  score_limit = 0.01,
  overwrite = FALSE,
  selector = "topDiffGenes",
  pval_column = "adj.P.Val"
)
```

## Arguments

| | |
|---|---|
| goseq | Data from goseq. |
| goid_map | File to save go id mapping. |
| score_limit | Score limit for the coloring. |
| overwrite | Overwrite the trees? |
| selector | Function for choosing genes. |
| pval_column | Column to acquire pvalues. |

## Value

A plot!

## See Also

**Ramigo**

---

gostats_kegg                         *Use gostats() against kegg pathways.*

---

## Description

This sets up a GSEABase analysis using KEGG pathways rather than gene ontologies. Does this even work? I don't think I have ever tested it yet. oh, it sort of does, maybe if I export it I will rembmer it.

## Usage

```
gostats_kegg(
  organism = "Homo sapiens",
  pathdb = "org.Hs.egPATH",
  godb = "org.Hs.egGO"
)
```

## Arguments

| | |
|---|---|
| organism | The organism used to make the KEGG frame, human readable no taxonomic. |
| pathdb | Name of the pathway database for this organism. |
| godb | Name of the ontology database for this organism. |

## Value

Results from hyperGTest using the KEGG pathways.

## See Also

**AnnotationDbi GSEABase Category**

gostats_trees *Take gostats data and print it on a tree as topGO does.*

## Description

This shoehorns gostats data into a format acceptable by topgo and uses it to print pretty ontology trees showing the over represented ontologies.

## Usage

```
gostats_trees(
  gostats_result,
  goid_map = "id2go.map",
  score_limit = 0.01,
  overwrite = FALSE,
  selector = "topDiffGenes",
  pval_column = "adj.P.Val"
)
```

## Arguments

| | |
|---|---|
| gostats_result | Return from simple_gostats(). |
| goid_map | Mapping of IDs to GO in the Ramigo expected format. |
| score_limit | Maximum score to include as 'significant'. |
| overwrite | Overwrite the goid_map? |
| selector | Function to choose differentially expressed genes in the data. |
| pval_column | Column in the data to be used to extract pvalue scores. |

## Value

plots! Trees! oh my!

## See Also

**topGO gostats**

---

gosyn                          *Get a go synonym from an ID.*

---

### Description

I think I will need to do similar parsing of the output for this function as per gosec() In some cases
this also returns stuff like c("some text", "GO:someID") versus "some other text" versus NULL
versus NA. This function just goes a mapply(gosn, go).

### Usage

```
gosyn(go = "GO:0000001")
```

### Arguments

go                      GO id, this may be a character or list(assuming the elements are goids).

### Value

Some text providing the synonyms for the given id(s).

### See Also

**GOTermsAnnDbBimap**

### Examples

```
## Not run:
 text =  gosyn("GO:0000001")
 text
 ## > GO:000001
 ## > "mitochondrial inheritance"

## End(Not run)
```

---

goterm                         *Get a go term from ID.*

---

### Description

Get a go term from ID.

### Usage

```
goterm(go = "GO:0032559")
```

## Arguments

go                GO id or a list thereof, this may be a character or list(assuming the elements, not names, are goids).

## Value

Some text containing the terms associated with GO id(s).

## See Also

**GOTermsAnnDbBimap**

## Examples

```
## Not run:
 goterm("GO:0032559")
 ## > GO:0032559
 ## > "adenyl ribonucleotide binding"

## End(Not run)
```

---

gotest                       *Test GO ids to see if they are useful.*

---

## Description

This just wraps gotst in mapply.

## Usage

```
gotest(go)
```

## Arguments

go                go IDs as characters.

## Value

Some text

## See Also

**GOTermsAnnDbBimap**

## Examples

```
## Not run:
 gotest("GO:0032559")
 ## > 1
 gotest("GO:0923429034823904")
 ## > 0

## End(Not run)
```

---

graph_metrics                     *Make lots of graphs!*

---

## Description

Plot out a set of metrics describing the state of an experiment including library sizes, # non-zero
genes, heatmaps, boxplots, density plots, pca plots, standard median distance/correlation, and qq
plots.

## Usage

```
graph_metrics(
  expt,
  cormethod = "pearson",
  distmethod = "euclidean",
  title_suffix = NULL,
  qq = FALSE,
  ma = FALSE,
  gene_heat = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| expt | an expt to process |
| cormethod | The correlation test for heatmaps. |
| distmethod | define the distance metric for heatmaps. |
| title_suffix | Text to add to the titles of the plots. |
| qq | Include qq plots? |
| ma | Include pairwise ma plots? |
| gene_heat | Include a heatmap of the gene expression data? |
| ... | Extra parameters optionally fed to the various plots |

**Value**

a loooong list of plots including the following:

1. nonzero = a ggplot2 plot of the non-zero genes vs library size

2. libsize = a ggplot2 bar plot of the library sizes

3. boxplot = a ggplot2 boxplot of the raw data

4. corheat = a recordPlot()ed pairwise correlation heatmap of the raw data

5. smc = a recordPlot()ed view of the standard median pairwise correlation of the raw data

6. disheat = a recordPlot()ed pairwise euclidean distance heatmap of the raw data

7. smd = a recordPlot()ed view of the standard median pairwise distance of the raw data

8. pcaplot = a recordPlot()ed PCA plot of the raw samples

9. pcatable = a table describing the relative contribution of condition/batch of the raw data

10. pcares = a table describing the relative contribution of condition/batch of the raw data

11. pcavar = a table describing the variance of the raw data

12. qq = a recordPlotted() view comparing the quantile/quantiles between the mean of all data and every raw sample

13. density = a ggplot2 view of the density of each raw sample (this is complementary but more fun than a boxplot)

**See Also**

**Biobase ggplot2 grDevices gplots** exprs hpgl_norm plot_nonzero plot_libsize plot_boxplot
plot_corheat plot_sm plot_disheat plot_pca plot_qq_all plot_pairwise_ma

**Examples**

```
## Not run:
 toomany_plots <- graph_metrics(expt)
 toomany_plots$pcaplot
 norm <- normalize_expt(expt, convert="cpm", batch=TRUE, filter_low=TRUE,
                        transform="log2", norm="rle")
 holy_asscrackers <- graph_metrics(norm, qq=TRUE, ma=TRUE)

## End(Not run)
```

---

gsva_likelihoods       *Score the results from gsva().*

---

**Description**

Yeah, this is a bit meta, but the scores from gsva seem a bit meaningless to me, so I decided to look at the distribution of observed scores in some of my data; I quickly realized that they follow a nicely normal distribution. Therefore, I thought to calculate some scores of gsva() using that information.

**Usage**

```
gsva_likelihoods(
  gsva_result,
  score = NULL,
  category = NULL,
  factor = NULL,
  sample = NULL,
  factor_column = "condition",
  method = "mean",
  label_size = NULL,
  col_margin = 6,
  row_margin = 12,
  cutoff = 0.05
)
```

**Arguments**

| | |
|---|---|
| gsva_result | Input result from simple_gsva() |
| score | What type of scoring to perform, against a value, column, row? |
| category | What category to use as baseline? |
| factor | Which experimental factor to compare against? |
| sample | Which sample to compare against? |
| factor_column | When comparing against an experimental factor, which design column to use to find it? |
| method | mean or median when when bringing together values? |

**Details**

The nicest thing in this, I think, is that it provides its scoring metric(s) according to a few different possibilities, including: * the mean of samples found in an experimental factor * All provided scores against the distribution of observed scores as z-scores. * A single score against all scores. * Rows (gene sets) against the set of all gene sets.

**Value**

The scores according to the provided category, factor, sample, or score(s).

---

guess_orgdb_keytype        *Iterate over keytypes looking for matches against a set of IDs.*

---

**Description**

Sometimes, one does not know what the correct keytype is for a given set of IDs. This will hopefully find them.

## Usage

```
guess_orgdb_keytype(ids, orgdb)
```

## Arguments

| | |
|---|---|
| ids | Set of gene IDs to seek. |
| orgdb | Orgdb instance to iterate through. |

## Value

Likely keytype which provides the desired IDs.

## Examples

```
ids <- c("Dm.9", "Dm.2294", "Dm.4971")
keytype_guess <- guess_orgdb_keytype(ids, orgdb)
keytype_guess
```

---

heatmap.3 *a minor change to heatmap.2 makes heatmap.3*

---

## Description

heatmap.2 is the devil.

## Usage

```
heatmap.3(
  x,
  Rowv = TRUE,
  Colv = if (symm) "Rowv" else TRUE,
  distfun = dist,
  hclustfun = fastcluster::hclust,
  dendrogram = c("both", "row", "column", "none"),
  reorderfun = function(d, w) reorder(d, w),
  symm = FALSE,
  scale = c("none", "row", "column"),
  na.rm = TRUE,
  revC = identical(Colv, "Rowv"),
  add.expr,
  breaks,
  symbreaks = min(x < 0, na.rm = TRUE) || scale != "none",
  col = "heat.colors",
  colsep,
  rowsep,
  sepcolor = "white",
  sepwidth = c(0.05, 0.05),
```

```
    cellnote,
    notecex = 1,
    notecol = "cyan",
    na.color = par("bg"),
    trace = c("column", "row", "both", "none"),
    tracecol = "cyan",
    hline = median(breaks),
    vline = median(breaks),
    linecol = tracecol,
    margins = c(5, 5),
    ColSideColors,
    RowSideColors,
    cexRow = 0.2 + 1/log10(nr),
    cexCol = 0.2 + 1/log10(nc),
    labRow = NULL,
    labCol = NULL,
    srtRow = NULL,
    srtCol = NULL,
    adjRow = c(0, NA),
    adjCol = c(NA, 0),
    offsetRow = 0.5,
    offsetCol = 0.5,
    key = TRUE,
    keysize = 1.5,
    density.info = c("histogram", "density", "none"),
    denscol = tracecol,
    symkey = min(x < 0, na.rm = TRUE) || symbreaks,
    densadj = 0.25,
    key.title = NULL,
    key.xlab = NULL,
    key.ylab = NULL,
    key.xtickfun = NULL,
    key.ytickfun = NULL,
    key.par = list(),
    main = NULL,
    xlab = NULL,
    ylab = NULL,
    lmat = NULL,
    lhei = NULL,
    lwid = NULL,
    extrafun = NULL,
    linewidth = 1,
    ...
  )
```

## Arguments

x                 data

| | |
|---|---|
| Rowv | add rows? |
| Colv | add columns? |
| distfun | distance function to use |
| hclustfun | clustering function to use |
| dendrogram | which axes to put trees on |
| reorderfun | reorder the rows/columns? |
| symm | symmetrical? |
| scale | add the scale? |
| na.rm | remove nas from the data? |
| revC | reverse the columns? |
| add.expr | no clue |
| breaks | also no clue |
| symbreaks | still no clue |
| col | colors! |
| colsep | column separator |
| rowsep | row separator |
| sepcolor | color to put between columns/rows |
| sepwidth | how much to separate |
| cellnote | mur? |
| notecex | size of the notes |
| notecol | color of the notes |
| na.color | a parameter call to bg |
| trace | do a trace for rows/columns? |
| tracecol | color of the trace |
| hline | the hline |
| vline | the vline |
| linecol | the line color |
| margins | margins are good |
| ColSideColors | colors for the columns as annotation |
| RowSideColors | colors for the rows as annotation |
| cexRow | row size |
| cexCol | column size |
| labRow | hmmmm |
| labCol | still dont know |
| srtRow | srt the row? |
| srtCol | srt the column? |
| adjRow | adj the row? |

| adjCol | adj the column? |
|---|---|
| offsetRow | how far to place the text from the row |
| offsetCol | how far to place the text from the column |
| key | add a key? |
| keysize | if so, how big? |
| density.info | for the key, what information to add |
| denscol | tracecol hmm ok |
| symkey | I like keys |
| densadj | adj the dens? |
| key.title | title for the key |
| key.xlab | text for the x axis of the key |
| key.ylab | text for the y axis of the key |
| key.xtickfun | add text to the ticks of the key x axis |
| key.ytickfun | add text to the ticks of the key y axis |
| key.par | parameters for the key |
| main | the main title of the plot |
| xlab | main x label |
| ylab | main y label |
| lmat | the lmat |
| lhei | the lhei |
| lwid | the lwid |
| extrafun | I do enjoy me some extra fun |
| linewidth | the width of lines |
| ... | because this function did not already have enough options |

**Value**

a heatmap!

**See Also**

[heatmap.2](heatmap.2)

hpgl_arescore *Implement the arescan function in R*

## Description

This function was taken almost verbatim from AREScore() in SeqTools Available at: https://github.com/lianos/seqtools.git At least on my computer I could not make that implementation work So I rewrapped its apply() calls and am now hoping to extend its logic a little to make it more sensitive and get rid of some of the spurious parameters or at least make them more transparent.

## Usage

```
hpgl_arescore(
  x,
  basal = 1,
  overlapping = 1.5,
  d1.3 = 0.75,
  d4.6 = 0.4,
  d7.9 = 0.2,
  within.AU = 0.3,
  aub.min.length = 10,
  aub.p.to.start = 0.8,
  aub.p.to.end = 0.55
)
```

## Arguments

| | |
|---|---|
| x | DNA/RNA StringSet containing the UTR sequences of interest |
| basal | I dunno. |
| overlapping | default=1.5 |
| d1.3 | default=0.75 These parameter names are so stupid, lets be realistic |
| d4.6 | default=0.4 |
| d7.9 | default=0.2 |
| within.AU | default=0.3 |
| aub.min.length | default=10 |
| aub.p.to.start | default=0.8 |
| aub.p.to.end | default=0.55 |

## Value

a DataFrame of scores

## See Also

**IRanges Biostrings**

## Examples

```
## Not run:
 ## Extract all the genes from my genome, pull a static region 120nt following the stop
 ## and test them for potential ARE sequences.
 ## FIXME: There may be an error in this example, another version I have
 ## handles the +/- strand genes separately, I need to return to this and check
 ## if it is providing the 5' UTR for 1/2 the genome, which would be
 ## unfortunate -- but the logic for testing remains the same.
 are_candidates <- hpgl_arescore(genome)
 utr_genes <- subset(lmajor_annotations, type == 'gene')
 threep <- GenomicRanges::GRanges(seqnames=Rle(utr_genes[,1]),
                                  ranges=IRanges(utr_genes[,3], end=(utr_genes[,3] + 120)),
                                     strand=Rle(utr_genes[,5]),
                                     name=Rle(utr_genes[,10]))
 threep_seqstrings <- Biostrings::getSeq(lm, threep)
 are_test <- hpgltools::hpgl_arescore(x=threep_seqstrings)
 are_genes <- rownames(are_test[ which(are_test$score > 0), ])

## End(Not run)
```

---

hpgl_cor                           *Wrap cor() to include robust correlations.*

---

## Description

Take covRob's robust correlation coefficient and add it to the set of correlations available when one
calls cor(). I should reimplement this using S4.

## Usage

```
hpgl_cor(df, method = "pearson", ...)
```

## Arguments

| | |
|---|---|
| df | Data frame to test. |
| method | Correlation method to use. Includes pearson, spearman, kendal, robust. |
| ... | Other options to pass to stats::cor(). |

## Value

Some fun correlation statistics.

## See Also

**robust** cor cov covRob

## Examples

```
## Not run:
 hpgl_cor(df=df)
 hpgl_cor(df=df, method="robust")

## End(Not run)
```

---

hpgl_dist                    *Because I am not smart enough to remember t()*

---

## Description

It seems to me there should be a function as easy for distances are there is for correlations.

## Usage

```
hpgl_dist(df, method = "euclidean", ...)
```

## Arguments

| | |
|---|---|
| df | data frame from which to calculate distances. |
| method | Which distance calculation to use? |
| ... | Extra arguments for dist. |

---

hpgl_filter_counts    *Filter low-count genes from a data set using cpm data and a threshold.*

---

## Description

This is identical to cbcb_filter_counts except it does not do the somewhat tortured log2CPM() but instead just uses a 4 cpm non-log threshold. It should therefore give basically the same result, but without the shenanigans.

## Usage

```
hpgl_filter_counts(
  count_table,
  threshold = 2,
  min_samples = 2,
  libsize = NULL,
  ...
)
```

**Arguments**

| | |
|---|---|
| `count_table` | Data frame of (pseudo)counts by sample. |
| `threshold` | Lower threshold of counts for each gene. |
| `min_samples` | Minimum number of samples. |
| `libsize` | Table of library sizes. |
| `...` | Arguments passed to cpm and friends. |

**Value**

Dataframe of counts without the low-count genes.

**See Also**

**edgeR**

**Examples**

```
## Not run:
 filtered_table <- cbcb_filter_counts(count_table)

## End(Not run)
```

---

hpgl_GOplot                    *A minor hack of the topGO GOplot function.*

---

**Description**

This allows me to change the line widths from the default.

**Usage**

```
hpgl_GOplot(
  dag,
  sigNodes,
  dag.name = "GO terms",
  edgeTypes = TRUE,
  nodeShape.type = c("box", "circle", "ellipse", "plaintext")[3],
  genNodes = NULL,
  wantedNodes = NULL,
  showEdges = TRUE,
  useFullNames = TRUE,
  oldSigNodes = NULL,
  nodeInfo = NULL,
  maxchars = 30
)
```

## Arguments

| | |
|---|---|
| `dag` | DAG tree of ontologies. |
| `sigNodes` | Set of significant ontologies (with p-values). |
| `dag.name` | Name for the graph. |
| `edgeTypes` | Types of the edges for graphviz. |
| `nodeShape.type` | Shapes on the tree. |
| `genNodes` | Generate the nodes? |
| `wantedNodes` | Subset of the ontologies to plot. |
| `showEdges` | Show the arrows? |
| `useFullNames` | Full names of the ontologies (they can get long). |
| `oldSigNodes` | I dunno. |
| `nodeInfo` | Hmm. |
| `maxchars` | Maximum characters per line inside the shapes. |

## Value

Topgo plot!

## See Also

**topGO**

---

`hpgl_GroupDensity`          *A hack of topGO's groupDensity()*

---

## Description

This just adds a couple wrappers to avoid errors in groupDensity.

## Usage

```
hpgl_GroupDensity(object, whichGO, ranks = TRUE, rm.one = FALSE)
```

## Arguments

| | |
|---|---|
| `object` | TopGO enrichment object. |
| `whichGO` | Individual ontology group to compare against. |
| `ranks` | Rank order the set of ontologies? |
| `rm.one` | Remove pvalue=1 groups? |

## Value

plot of group densities.

---

hpgl_log2cpm             *Converts count matrix to log2 counts-per-million reads.*

---

### Description

Based on the method used by limma as described in the Law et al. (2014) voom paper.

### Usage

```
hpgl_log2cpm(counts, lib.size = NULL)
```

### Arguments

| | |
|---|---|
| counts | Read count matrix. |
| lib.size | Library size. |

### Value

log2-CPM read count matrix.

### See Also

**edgeR**

### Examples

```
## Not run:
 l2cpm <- hpgl_log2cpm(counts)

## End(Not run)
```

---

hpgl_norm                *Normalize a dataframe/expt, express it, and/or transform it*

---

### Description

There are many possible options to this function. Refer to normalize_expt() for a more complete list.

### Usage

```
hpgl_norm(data, ...)
```

### Arguments

| | |
|---|---|
| data | Some data as a df/expt/whatever. |
| ... | I should put all those other options here |

## Value

edgeR's DGEList expression of a count table. This seems to me to be the easiest to deal with.

## See Also

**edgeR DESeq2** `cpm` `rpkm` `hpgl_rpkm` `DESeqDataSetFromMatrix` `estimateSizeFactors` `DGEList` `calcNormFactors`

## Examples

```
## Not run:
 df_raw = hpgl_norm(expt=expt)  ## Only performs low-count filtering
 df_raw = hpgl_norm(df=a_df, design=a_design) ## Same, but using a df
 df_ql2rpkm = hpgl_norm(expt=expt, norm='quant', transform='log2',
                        convert='rpkm')  ## Quantile, log2, rpkm
 count_table = df_ql2rpkm$counts

## End(Not run)
```

---

| hpgl_padjust | *Wrap p.adjust to add IHW adjustments as an option.* |

---

## Description

IHW and apeglm are the two new toys I found, this adds the former as a way to adjust p-values.

## Usage

```
hpgl_padjust(
  data,
  pvalue_column = "pvalue",
  mean_column = "base_mean",
  method = "fdr",
  significance = 0.05,
  type = NULL
)
```

## Arguments

| | |
|---|---|
| data | Column or table containing values to adjust. |
| pvalue_column | Name of the column in a table containing the p-values. |
| mean_column | Name of the column in a table containing the mean count values to weight. |
| method | p adjustment method to apply. |
| significance | Passed to IHW |
| type | Assuming a DE table, what type of DE is this? |

hpgl_qshrink *A hacked copy of Kwame's qsmooth/qstats code.*

## Description

I made a couple small changes to Kwame's qstats() function to make it not fail when on corner-cases. I sent him a diff, but haven't checked to see if it was useful yet.

## Usage

```
hpgl_qshrink(
  data = NULL,
  groups = NULL,
  refType = "mean",
  groupLoc = "mean",
  window = 99,
  groupCol = NULL,
  plot = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Count table to modify |
| groups | Factor of the experimental conditions |
| refType | Method for grouping conditions |
| groupLoc | Method for grouping groups |
| window | Window, for looking! |
| groupCol | Column to define conditions |
| plot | Plot the quantiles? |
| ... | More options |

## Value

New data frame of normalized counts

## See Also

**qsmooth**

## Examples

```
## Not run:
 df <- hpgl_qshrink(data)

## End(Not run)
```

---

hpgl_qstats                    *A hacked copy of Kwame's qsmooth/qstats code.*

---

## Description

I made a couple small changes to Kwame's qstats() function to make it not fail when on corner-cases. I sent him a diff, but haven't checked to see if it was useful yet.

## Usage

```
hpgl_qstats(data, groups, refType = "mean", groupLoc = "mean", window = 99)
```

## Arguments

| | |
|---|---|
| data | Initial count data |
| groups | Experimental conditions as a factor. |
| refType | Method to separate groups, mean or median. |
| groupLoc | I don't remember what this is for. |
| window | Window for basking! |

## Value

Some new data.

## See Also

**matrixStats**

## Examples

```
## Not run:
 qstatted <- hpgl_qstats(data, conditions)

## End(Not run)
```

---

hpgl_rpkm                      *Reads/(kilobase(gene) * million reads)*

---

## Description

Express a data frame of counts as reads per kilobase(gene) per million(library). This function wraps EdgeR's rpkm in an attempt to make sure that the required gene lengths get sent along.

## Usage

```
hpgl_rpkm(count_table, ...)
```

## Arguments

| | |
|---|---|
| `count_table` | Data frame of counts, alternately an edgeR DGEList. |
| `...` | extra options including annotations for defining gene lengths. |

## Value

Data frame of counts expressed as rpkm.

## See Also

**edgeR** `cpm` `rpkm`

## Examples

```
## Not run:
 rpkm_df = hpgl_rpkm(df, annotations=gene_annotations)

## End(Not run)
```

---

hpgl_voom *A slight modification of limma's voom().*

---

## Description

Estimate mean-variance relationship between samples and generate 'observational-level weights' in preparation for linear modeling RNAseq data. This particular implementation was primarily scabbed from cbcbSEQ, but changes the mean-variance plot slightly and attempts to handle corner cases where the sample design is confounded by setting the coefficient to 1 for those samples rather than throwing an unhelpful error. Also, the Elist output gets a 'plot' slot which contains the plot rather than just printing it.

## Usage

```
hpgl_voom(
  dataframe,
  model = NULL,
  libsize = NULL,
  normalize.method = "none",
  span = 0.5,
  stupid = FALSE,
  logged = FALSE,
  converted = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `dataframe` | Dataframe of sample counts which have been normalized and log transformed. |
| `model` | Experimental model defining batches/conditions/etc. |
| `libsize` | Size of the libraries (usually provided by edgeR). |
| `normalize.method` | |
| | Normalization method used in voom(). |
| `span` | The span used in voom(). |
| `stupid` | Cheat when the resulting matrix is not solvable? |
| `logged` | Is the input data is known to be logged? |
| `converted` | Is the input data is known to be cpm converted? |
| `...` | Extra arguments are passed to arglist. |

## Value

EList containing the following information: E = The normalized data weights = The weights of said data design = The resulting design lib.size = The size in pseudocounts of the library plot = A ggplot of the mean/variance trend with a blue loess fit and red trend fit

## See Also

**limma ggplot2**

## Examples

```
## Not run:
 funkytown = hpgl_voom(samples, model)

## End(Not run)
```

---

| | |
|---|---|
| hpgl_voomweighted | *A minor change to limma's voom with quality weights to attempt to address some corner cases.* |

---

## Description

This copies the logic employed in hpgl_voom(). I suspect one should not use it.

## Usage

```
hpgl_voomweighted(
  data,
  fun_model,
  libsize = NULL,
  normalize.method = "none",
  plot = TRUE,
```

```
    span = 0.5,
    var.design = NULL,
    method = "genebygene",
    maxiter = 50,
    tol = 1e-10,
    trace = FALSE,
    replace.weights = TRUE,
    col = NULL,
    ...
)
```

## Arguments

| | |
|---|---|
| data | Some data! |
| fun_model | A model for voom() and arrayWeights() |
| libsize | Library sizes passed to voom(). |
| normalize.method | |
| | Passed to voom() |
| plot | Do the plot of mean variance? |
| span | yes |
| var.design | maybe |
| method | kitty! |
| maxiter | 50 is good |
| tol | I have no tolerance. |
| trace | no trace for you. |
| replace.weights | |
| | Replace the weights? |
| col | yay columns! |
| ... | more arguments! |

## Value

a voom return

## See Also

**limma**

## Examples

```
## Not run:
## No seriously, dont run this, I think it is wiser to use the functions
## provided by limma. But this provides a place to test stuff out.
 voom_result <- hpgl_voomweighted(dataset, model)


## End(Not run)
```

| hpgltools | *hpgltools: a suite of tools to make our analyses easier* |

## Description

This provides a series of helpers for working with sequencing data

## Details

It falls under a few main topics

- Data exploration, look for trends in sequencing data and identify batch effects or skewed distributions.

- Differential expression analyses, use DESeq2/limma/EdgeR in a hopefully robust and flexible fashion.

- Ontology analyses, use goseq/clusterProfiler/topGO/GOStats/gProfiler in hopefully robust ways.

- Perform some simple TnSeq analyses.

To see examples of this in action, check out the vignettes: `browseVignettes(package = 'hpgltools')`

| ihw_adjust | *Make sure the outputs from limma and friends are in a format suitable for IHW.* |

## Description

IHW seems like an excellent way to improve the confidence in the p-values provided by the various DE methods. It expects inputs fairly specific to DESeq2, however, it is trivial to convert other methods to this, ergo this function.

## Usage

```
ihw_adjust(
  de_result,
  pvalue_column = "pvalue",
  type = NULL,
  mean_column = "baseMean",
  significance = 0.05
)
```

## Arguments

| | |
|---|---|
| `de_result` | Table which should have the 2 types of requisite columns: mean value of counts and p-value. |
| `pvalue_column` | Name of the column of p-values. |
| `type` | If specified, this will explicitly perform the calculation for the given type of differential expression analysis: limma, edger, deseq, etc. |
| `mean_column` | Name of the column of mean values. |
| `significance` | IHW uses this parameter, I don't know why. |

## Details

https://bioconductor.org/packages/release/bioc/vignettes/IHW/inst/doc/introduction_to_ihw.html

## Value

weight adjusted p-values.

---

| | |
|---|---|
| import_deseq | *Try to add data to DESeq in a flexible fashion. This currently only handles matrices, htseq data, and tximport data.* |

---

## Description

This will hopefully make adding counts to a DESeq data set easier, as it tries to handle the various arguments with minimal fuss.

## Usage

```
import_deseq(data, column_data, model_string, tximport = NULL)
```

## Arguments

| | |
|---|---|
| `data` | Counts from htseq/mtrx/tximport/etc |
| `column_data` | I think this is the sample names, I forget. |
| `model_string` | Model describing the data by sample names. |
| `tximport` | Where is this data coming from? |

| import_edger | *Import tximport information into edgeR.* |
|---|---|

### Description

This was taken from the tximport manual with minor modifications.

### Usage

```
import_edger(data, conditions, tximport = NULL)
```

### Arguments

| | |
|---|---|
| data | data to be coerced into edgeR. |
| conditions | Set of conditions used to make the DGEList. |
| tximport | Tell this if the data is actually coming from tximport. |

### Value

Hopefully valid DGEList for edgeR.

| impute_expt | *Impute missing values using code from DEP reworked for expression-sets.* |
|---|---|

### Description

impute_expt imputes missing values in a proteomics dataset.

### Usage

```
impute_expt(
  expt,
  filter = TRUE,
  p = 0.5,
 fun = c("bpca", "knn", "QRILC", "MLE", "MinDet", "MinProb", "min", "zero", "mixed",
    "nbavg"),
  ...
)
```

## Arguments

| | |
|---|---|
| expt | An ExpressionSet (well, expt), I think it is assumed that this should have been normalized and filtered for features which have no values across 'most' samples. |
| filter | Use normalize_expt() to filter the data? |
| p | When filtering with pofa, use this p parameter. |
| fun | "bpca", "knn", "QRILC", "MLE", "MinDet", "MinProb", "man", "min", "zero", "mixed" or "nbavg", Function used for data imputation based on [impute](). |
| ... | Additional arguments for imputation functions as depicted in [impute](). |

## Value

An imputed expressionset.

---

init_xlsx            *Initialize an xlsx file with a little bit of logic to make sure there are no annoying downstream errors.*

---

## Description

Initialize an xlsx file with a little bit of logic to make sure there are no annoying downstream errors.

## Usage

```
init_xlsx(excel = "excel/something.xlsx")
```

## Arguments

| | |
|---|---|
| excel | Excel file to create. |

---

intersect_signatures    *Take a result from simple_gsva(), a list of gene IDs, and intersect them.*

---

## Description

Najib is curious about the relationship of genes in sets, the sets, and the genes that comprise those sets. This is pushing gsva towards a oroborous-ish state.

## Usage

```
intersect_signatures(
  gsva_result,
  lst,
  freq_cutoff = 2,
  sig_weights = TRUE,
  gene_weights = TRUE
)
```

## Arguments

| | |
|---|---|
| `gsva_result` | Result from simple_gsva(). |
| `lst` | List of genes of interest. |
| `freq_cutoff` | Minimum number of observations to be counted. |
| `sig_weights` | When making venn diagrams, weight them? |
| `gene_weights` | When venning genes, weight them? |

## Value

List containing some venns, lists, and such.

---

intersect_significant *Find the sets of intersecting significant genes*

---

## Description

Use extract_significant_genes() to find the points of agreement between limma/deseq/edger.

## Usage

```
intersect_significant(
  combined,
  lfc = 1,
  p = 0.05,
  padding_rows = 2,
  z = NULL,
  p_type = "adj",
  selectors = c("limma", "deseq", "edger"),
  order = "inverse",
  excel = "excel/intersect_significant.xlsx",
  ...
)
```

## Arguments

| | |
|---|---|
| `combined` | Result from combine_de_tables(). |
| `lfc` | Define significant via fold-change. |
| `p` | Or p-value. |
| `padding_rows` | How much space to put between groups of data? |
| `z` | Use a z-score filter? |
| `p_type` | Use normal or adjusted p-values. |
| `selectors` | List of methods to intersect. |
| `order` | When set to the default 'inverse', go from the set with the most least intersection to the most. E.g. Start with abc,bc,ac,c,ab,b,a as opposed to a,b,ab,c,ac,bc,abc. |
| `excel` | An optional excel workbook to which to write. |
| `...` | Extra arguments for extract_significant_genes() and friends. |

**Value**

List containing the intersections between the various DE methods for both the up and down sets
of genes. It should also provide some venn diagrams showing the degree of similarity between the
methods.

**Examples**

```
## Not run:
 expt <- create_expt(metadata="some_metadata.xlsx", gene_info=funkytown)
 big_result <- all_pairwise(expt, model_batch=FALSE)
 pretty <- combine_de_tables(big_result, excel="excel/combined_expt.xlsx")
 intersect <- intersect_significant(pretty, excel="excel/intersecting_genes.xlsx")

## End(Not run)
```

---

kegg_vector_to_df            *Convert a potentially non-unique vector from kegg into a normalized*
                             *data frame.*

---

**Description**

This function seeks to reformat data from KEGGREST into something which is rather easier to use.

**Usage**

```
kegg_vector_to_df(vector, final_colname = "first", flatten = TRUE)
```

**Arguments**

vector            Information from KEGGREST

final_colname     Column name for the new information

flatten           Flatten nested data?

**Details**

This could probably benefit from a tidyr-ish revisitation.

**Value**

A normalized data frame of gene IDs to whatever.

limma_pairwise        *Set up a model matrix and set of contrasts for pairwise comparisons*
                      *using voom/limma.*

## Description

Creates the set of all possible contrasts and performs them using voom/limma.

## Usage

```
limma_pairwise(
  input = NULL,
  conditions = NULL,
  batches = NULL,
  model_cond = TRUE,
  model_batch = TRUE,
  model_intercept = FALSE,
  alt_model = NULL,
  extra_contrasts = NULL,
  annot_df = NULL,
  libsize = NULL,
  force = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| input | Dataframe/vector or expt class containing count tables, normalization state, etc. |
| conditions | Factor of conditions in the experiment. |
| batches | Factor of batches in the experiment. |
| model_cond | Include condition in the model? |
| model_batch | Include batch in the model? This is hopefully TRUE. |
| model_intercept | |
| | Perform a cell-means or intercept model? A little more difficult for me to understand. I have tested and get the same answer either way. |
| alt_model | Separate model matrix instead of the normal condition/batch. |
| extra_contrasts | |
| | Some extra contrasts to add to the list. This can be pretty neat, lets say one has conditions A,B,C,D,E and wants to do (C/B)/A and (E/D)/A or (E/D)/(C/B) then use this with a string like: "c_vs_b_ctrla = (C-B)-A, e_vs_d_ctrla = (E-D)-A, de_vs_cb = (E-D)-(C-B)," |
| annot_df | Data frame for annotations. |
| libsize | I've recently figured out that libsize is far more important than I previously realized. Play with it here. |
| force | Force data which may not be appropriate for limma into it? |
| ... | Use the elipsis parameter to feed options to write_limma(). |

**Value**

List including the following information: macb = the mashing together of condition/batch so you can look at it macb_model = The result of calling model.matrix(~0 + macb) macb_fit = The result of calling lmFit(data, macb_model) voom_result = The result from voom() voom_design = The design from voom (redundant from voom_result, but convenient) macb_table = A table of the number of times each condition/batch pairing happens cond_table = A table of the number of times each condition appears (the denominator for the identities) batch_table = How many times each batch appears identities = The list of strings defining each condition by itself all_pairwise = The list of strings defining all the pairwise contrasts contrast_string = The string making up the make-Contrasts() call pairwise_fits = The result from calling contrasts.fit() pairwise_comparisons = The result from eBayes() limma_result = The result from calling write_limma()

**See Also**

**limma Biobase** `write_limma`

**Examples**

```
## Not run:
 pretend <- limma_pairwise(expt)

## End(Not run)
```

---

load_annotations     *Use one of the load_\*_annotations() functions to gather annotation data.*

---

**Description**

We should be able to have an agnostic annotation loader which can take some standard arguments and figure out where to gather data on its own.

**Usage**

```
load_annotations(type = NULL, ...)
```

**Arguments**

| | |
|---|---|
| type | Explicitly state the type of annotation data to load. If not provided, try to figure it out automagically. |
| ... | Arguments passed to the other load_*_annotations(). |

**Value**

Some annotations, hopefully.

## Examples

```
gff_annotations <- load_annotations(type="gff", gff=gff_file)
dim(gff_annotations)
```

---

load_biomart_annotations

*Extract annotation information from biomart.*

---

## Description

Biomart is an amazing resource of information, but using it is a bit annoying. This function hopes to alleviate some common headaches.

## Usage

```
load_biomart_annotations(
  species = "hsapiens",
  overwrite = FALSE,
  do_save = TRUE,
  host = NULL,
  trymart = "ENSEMBL_MART_ENSEMBL",
  archive = TRUE,
 default_hosts = c("useast.ensembl.org", "uswest.ensembl.org", "www.ensembl.org",
    "asia.ensembl.org"),
  year = NULL,
  month = NULL,
  drop_haplotypes = TRUE,
  trydataset = NULL,
  gene_requests = c("ensembl_gene_id", "version", "ensembl_transcript_id",
    "transcript_version", "hgnc_symbol", "description", "gene_biotype"),
  length_requests = c("ensembl_transcript_id", "cds_length", "chromosome_name",
    "strand", "start_position", "end_position"),
  include_lengths = TRUE
)
```

## Arguments

| | |
|---|---|
| species | Choose a species. |
| overwrite | Overwite an existing save file? |
| do_save | Create a savefile of annotations for future runs? |
| host | Ensembl hostname to use. |
| trymart | Biomart has become a circular dependency, this makes me sad, now to list the marts, you need to have a mart loaded. |
| drop_haplotypes | |
| | Some chromosomes have stupid names because they are from non-standard haplotypes and they should go away. Setting this to false stops that. |

| trydataset | Choose the biomart dataset from which to query. |
|---|---|
| gene_requests | Set of columns to query for description-ish annotations. |
| length_requests | |
| | Set of columns to query for location-ish annotations. |
| include_lengths | |
| | Also perform a search on structural elements in the genome? |

### Details

Tested in test_40ann_biomart.R This goes to some lengths to find the relevant tables in biomart. But biomart is incredibly complex and one should carefully inspect the output if it fails to see if there are more appropriate marts, datasets, and columns to download.

### Value

List containing: a data frame of the found annotations, a copy of The mart instance to help with finding problems, the hostname queried, the name of the mart queried, a vector of rows queried, vector of the available attributes, and the ensembl dataset queried.

### See Also

**biomaRt** listDatasets getBM

### Examples

```
## This downloads the hsapiens annotations by default.
hs_biomart_annot <- load_biomart_annotations()
summary(hs_biomart_annot)
dim(hs_biomart_annot$annotation)
```

---

load_biomart_go          *Extract gene ontology information from biomart.*

---

### Description

I perceive that every time I go to acquire annotation data from biomart, they have changed something important and made it more difficult for me to find what I want. I recently found the *.archive.ensembl.org, and so this function uses that to try to keep things predictable, if not consistent.

### Usage

```
load_biomart_go(
  species = "hsapiens",
  overwrite = FALSE,
  do_save = TRUE,
  host = "dec2015.archive.ensembl.org",
  trymart = "ENSEMBL_MART_ENSEMBL",
```

```
    secondtry = "_gene",
    dl_rows = c("ensembl_gene_id", "go_accession"),
    dl_rowsv2 = c("ensembl_gene_id", "go_id")
)
```

## Arguments

| | |
|---|---|
| species | Species to query. |
| overwrite | Overwrite existing savefile? |
| do_save | Create a savefile of the annotations? (if not false, then a filename.) |
| host | Ensembl hostname to use. |
| trymart | Default mart to try, newer marts use a different notation. |
| secondtry | The newer mart name. |
| dl_rows | List of rows from the final biomart object to download. |
| dl_rowsv2 | A second list of potential rows. |

## Details

Tested in test_40ann_biomart.R This function makes a couple of attempts to pick up the correct tables from biomart. It is worth noting that it uses the archive.ensembl host(s) because of changes in table organization after December 2015 as well as an attempt to keep the annotation sets relatively consistent.

## Value

List containing the following: data frame of ontology data, a copy of the biomart instance for further querying, the host queried, the biomart queried, a vector providing the attributes queried, and the ensembl dataset queried.

## See Also

**biomaRt** `listMarts useDataset getBM`

## Examples

```
hs_biomart_ontology <-load_biomart_go()
summary(hs_biomart_ontology)
dim(hs_biomart_ontology$go)
```

---

load_biomart_orthologs

*Use biomart to get orthologs between supported species.*

---

### Description

Biomart's function getLDS is incredibly powerful, but it makes me think very polite people are going to start knocking on my door, and it fails weirdly pretty much always. This function attempts to alleviate some of that frustration.

### Usage

```
load_biomart_orthologs(
  gene_ids = NULL,
  first_species = "hsapiens",
  second_species = "mmusculus",
  host = "dec2016.archive.ensembl.org",
  trymart = "ENSEMBL_MART_ENSEMBL",
  attributes = "ensembl_gene_id"
)
```

### Arguments

| | |
|---|---|
| gene_ids | List of gene IDs to translate. |
| first_species | Linnean species name for one species. |
| second_species | Linnean species name for the second species. |
| host | Ensembl server to query. |
| trymart | Assumed mart name to use. |
| attributes | Key to query |

### Details

Tested in test_40ann_biomart.R As with my other biomart functions, this one grew out of frustrations when attempting to work with the incredibly unforgiving biomart service. It does not attempt to guarantee a useful biomart connection, but will hopefully point out potentially correct marts and attributes to use for a successful query. I can say with confidence that it works well between mice and humans.

### Value

list of 4 elements: The first is the set of all ids, as getLDS seems to always send them all; the second is the subset corresponding to the actual ids of interest, and the 3rd/4th are other, optional ids from other datasets.

### See Also

**biomaRt** getLDS useMart

## Examples

```
mouse_yeast_orthologs <- load_biomart_orthologs(gene_ids=NULL, first_species="mmusculus",
                                                  second_species="scerevisiae")
head(mouse_yeast_orthologs$all_linked_genes)
```

---

load_genbank_annotations

*Given a genbank accession, make a txDb object along with sequences,
etc.*

---

## Description

Let us admit it, sometimes biomart is a pain. It also does not have easily accessible data for microbes. Genbank does!

## Usage

```
load_genbank_annotations(
  accession = "AE009949",
  reread = TRUE,
  savetxdb = FALSE
)
```

## Arguments

| | |
|---|---|
| accession | Accession to download and import |
| reread | Re-read (download) the file from genbank |
| savetxdb | Attempt saving a txdb object? |

## Details

Tested in test_40ann_biomartgenbank.R and test_70expt_spyogenes.R This primarily sets some defaults for the genbankr service in order to facilitate downloading genomes from genbank and dumping them into a local txdb instance.

## Value

List containing a txDb, sequences, and some other stuff which I haven't yet finalized.

## See Also

**genbankr rentrez** [import](import)

## Examples

```
sagalacticae_genbank_annot <- load_genbank_annotations(accession="AE009948")
dim(as.data.frame(sagalacticae_genbank_annot$cds))
```

load_gff_annotations    *Extract annotation information from a gff file into a df*

## Description

Try to make import.gff a little more robust; I acquire (hopefully) valid gff files from various sources:
yeastgenome.org, microbesonline, tritrypdb, ucsc, ncbi. To my eyes, they all look like reasonably
good gff3 files, but some of them must be loaded with import.gff2, import.gff3, etc. That is super
annoying. Also, I pretty much always just do as.data.frame() when I get something valid from
rtracklayer, so this does that for me, I have another function which returns the iranges etc. This
function wraps import.gff/import.gff3/import.gff2 calls in try() because sometimes those functions
fail in unpredictable ways.

## Usage

```
load_gff_annotations(
  gff,
  type = NULL,
  id_col = "ID",
  ret_type = "data.frame",
  second_id_col = "locus_tag",
  try = NULL,
  row.names = NULL
)
```

## Arguments

| | |
|---|---|
| gff | Gff filename. |
| type | Subset the gff file for entries of a specific type. |
| id_col | Column in a successful import containing the IDs of interest. |
| ret_type | Return a data.frame or something else? |
| second_id_col | Second column to check. |
| try | Give your own function call to use for importing. |
| row.names | Choose another column for setting the rownames of the data frame. |

## Value

Dataframe of the annotation information found in the gff file.

## See Also

**rtracklayer GenomicRanges** `import.gff`

## Examples

```
gas_gff_annot <- load_gff_annotations(gff_file)
dim(gas_gff_annot)
```

load_kegg_annotations  *Create a data frame of pathways to gene IDs from KEGGREST*

### Description

This seeks to take the peculiar format from KEGGREST for pathway<->genes and make it easier to deal with.

### Usage

```
load_kegg_annotations(species = "coli", abbreviation = NULL, flatten = TRUE)
```

### Arguments

| | |
|---|---|
| species | String to use to query KEGG abbreviation. |
| abbreviation | If you already know the abbreviation, use it. |
| flatten | Flatten nested tables? |

### Value

dataframe with rows of KEGG gene IDs and columns of NCBI gene IDs and KEGG paths.

### Examples

```
sc_kegg_annot <- load_kegg_annotations(species="cerevisiae")
head(sc_kegg_annot)
```

load_microbesonline_annotations

*Skip the db and download all the text annotations for a given species.*

### Description

The microbesonline publicly available mysqldb is rather more complex than I prefer. This skips that process and just grabs a tsv copy of everything and loads it into a dataframe. I have not yet figured out how to so-easily query microbesonline for species IDs, thus one will have to manually query the database to find species of interest.

### Usage

```
load_microbesonline_annotations(species = NULL, id = NULL)
```

### Arguments

| | |
|---|---|
| species | Microbesonline species. |
| id | Microbesonline ID to query. |

## Details

Tested in test_70expt_spyogenes.R There is so much awesome information in microbesonline, but damn is it annoying to download. This function makes that rather easier, or so I hope at least.

## Value

Dataframe containing the annotation information.

## See Also

**rvest**

## Examples

```
pa14_microbesonline_annot <- load_microbesonline_annotations(species="PA14")
colnames(pa14_microbesonline_annot)
```

---

load_microbesonline_go
*Extract the set of GO categories by microbesonline locus*

---

## Description

The microbesonline is such a fantastic resource, it is a bit of a shame that it is such a pain to query.

## Usage

```
load_microbesonline_go(
  id = NULL,
  species = NULL,
  table_df = NULL,
  id_column = "name",
  data_column = "GO",
  name = NULL
)
```

## Arguments

| | |
|---|---|
| id | Which species to query. |
| species | Microbesonline species. |
| table_df | Pre-existing data frame of annotations containing GO stuff. |
| id_column | This no longer uses MySQL, so which column from the html table to pull? |
| data_column | Similar to above, there are lots of places from which one might extract the data. |
| name | Allowing for non-specific searches by species name. |

**Details**

Tested in test_42ann_microbes.R I am not 100 ontology accessions. At the very least, it does return a large number of them, which is a start.

**Value**

data frame of GO terms from www.microbesonline.org

**Examples**

```
pa14_microbesonline_go <- load_microbesonline_go(species="PA14")
head(pa14_microbesonline_go)
```

---

load_orgdb_annotations

*Load organism annotation data from an orgdb sqlite package.*

---

**Description**

Creates a dataframe gene and transcript information for a given set of gene ids using the AnnotationDbi interface.

**Usage**

```
load_orgdb_annotations(
  orgdb = NULL,
  gene_ids = NULL,
  include_go = FALSE,
  keytype = "ensembl",
  strand_column = "cdsstrand",
  start_column = "cdsstart",
  end_column = "cdsend",
  chromosome_column = "cdschrom",
  type_column = "gene_type",
  name_column = "cdsname",
  fields = NULL,
  sum_exon_widths = FALSE
)
```

**Arguments**

| | |
|---|---|
| orgdb | OrganismDb instance. |
| gene_ids | Search for a specific set of genes? |
| include_go | Ask the Dbi for gene ontology information? |
| keytype | mmm the key type used? |

| | |
|---|---|
| strand_column | There are a few fields I want to gather by default: start, end, strand, chromosome, type, and name; but these do not necessarily have consistent names, use this column for the chromosome strand. |
| start_column | Use this column for the gene start. |
| end_column | Use this column for the gene end. |
| chromosome_column | |
| | Use this column to identify the chromosome. |
| type_column | Use this column to identify the gene type. |
| name_column | Use this column to identify the gene name. |
| fields | Columns included in the output. |
| sum_exon_widths | |
| | Perform a sum of the exons in the data set? |

## Details

Tested in test_45ann_organdb.R This defaults to a few fields which I have found most useful, but the brave or pathological can pass it 'all'.

## Value

Table of geneids, chromosomes, descriptions, strands, types, and lengths.

## See Also

**AnnotationDbi GenomicFeatures BiocGenerics** columns keytypes select exonsBy

## Examples

```
hs_orgdb_annot <- load_orgdb_annotations()
summary(hs_orgdb_annot$genes)
```

---

| load_orgdb_go | *Retrieve GO terms associated with a set of genes.* |
|---|---|

---

## Description

AnnotationDbi provides a reasonably complete set of GO mappings between gene ID and ontologies. This will extract that table for a given set of gene IDs.

## Usage

```
load_orgdb_go(
  orgdb = NULL,
  gene_ids = NULL,
  keytype = "ensembl",
  columns = c("go", "goall", "goid")
)
```

## Arguments

| | |
|---|---|
| orgdb | OrganismDb instance. |
| gene_ids | Identifiers of the genes to retrieve annotations. |
| keytype | The mysterious keytype returns yet again to haunt my dreams. |
| columns | The set of columns to request. |

## Details

Tested in test_45ann_organdb.R This is a nice way to extract GO data primarily because the Orgdb data sets are extremely fast and flexible, thus by changing the keytype argument, one may use a lot of different ID types and still score some useful ontology data.

## Value

Data frame of gene IDs, go terms, and names.

## Author(s)

I think Keith provided the initial implementation of this, but atb messed with it pretty extensively.

## See Also

**AnnotationDbi GO.db magrittr** [select](#)

## Examples

```
drosophila_orgdb_go <- load_orgdb_go(orgdb="org.Dm.eg.db")
head(drosophila_orgdb_go)
```

---

load_trinotate_annotations
                    *Read a csv file from trinotate and make an annotation data frame.*

---

## Description

Trinotate performs some neat sequence searches in order to seek out likely annotations for the trinity contigs. The resulting csv file is encoded in a peculiar fashion, so this function attempts to make it easier to read and put them into a format usable in an expressionset.

## Usage

```
load_trinotate_annotations(trinotate = "reference/trinotate.csv")
```

## Arguments

| | |
|---|---|
| trinotate | CSV of trinotate annotation data. |

**Value**

Dataframe of fun data.

## Examples

```
trinotate <- system.file("sb/trinotate_head.csv.xz", package="hpgltools")
a_few_trinotate <- load_trinotate_annotations(trinotate=trinotate)
dim(a_few_trinotate)
```

---

load_trinotate_go            *Read a csv file from trinotate and extract ontology data from it.*

---

## Description

Trinotate performs some neat sequence searches in order to seek out likely annotations for the trinity
contigs. This function extracts ontology data from it. Keep in mind that this data is primarily from
Blast2GO.

## Usage

```
load_trinotate_go(trinotate = "reference/trinotate.csv")
```

## Arguments

trinotate          CSV of trinotate annotation data.

## Value

List of the extracted GO data, a table of it, length data, and the resulting length table.

## Examples

```
trinotate_go <- load_trinotate_go(trinotate=trinotate)
dim(trinotate_go$go_data)
dim(trinotate_go$go_table)
```

## load_uniprot_annotations

*Read a uniprot text file and extract as much information from it as possible.*

### Description

I spent entirely too long fighting with Uniprot.ws, finally got mad and wrote this.

### Usage

```
load_uniprot_annotations(file = NULL, species = NULL, savefile = TRUE)
```

### Arguments

| | |
|---|---|
| file | Uniprot file to read and parse |
| species | Species name to download/load. |
| savefile | Do a save? |

### Value

Big dataframe of annotation data.

### Examples

```
sc_uniprot_annot <- load_uniprot_annotations(file=uniprot_sc_downloaded$filename)
dim(sc_uniprot_annot)
```

## load_uniprot_go

*Extract ontology information from a uniprot dataframe.*

### Description

Extract ontology information from a uniprot dataframe.

### Usage

```
load_uniprot_go(input)
```

### Arguments

| | |
|---|---|
| input | uniprot filename or dataframe. |

### Value

Ontology dataframe

## Examples

```
sc_uniprot_go <- load_uniprot_go(sc_uniprot_annot)
head(sc_uniprot_go)
```

---

loadme                    *Load a backup rdata file*

---

### Description

I often use R over a sshfs connection, sometimes with significant latency, and I want to be able to save/load my R sessions relatively quickly. Thus this function uses my backup directory to load its R environment.

### Usage

```
loadme(directory = "savefiles", filename = "Rdata.rda.xz")
```

### Arguments

| | |
|---|---|
| directory | Directory containing the RData.rda.xz file. |
| filename | Filename to which to save. |

### Value

a bigger global environment

### See Also

[saveme](#) [load](#) [save](#)

### Examples

```
## Not run:
 loadme()

## End(Not run)
```

---

local_get_value | *Perform a get_value for delimited files*

---

### Description

Keith wrote this as .get_value() but functions which start with . trouble me.

### Usage

```
local_get_value(x, delimiter = ": ")
```

### Arguments

x                Some stuff to split

delimiter        The tritrypdb uses ': ' ergo the default.

### Value

A value!

---

make_exampledata | *Small hack of limma's exampleData() to allow for arbitrary data set sizes.*

---

### Description

exampleData has a set number of genes/samples it creates. This relaxes that restriction.

### Usage

```
make_exampledata(ngenes = 1000, columns = 5)
```

### Arguments

ngenes           How many genes in the fictional data set?

columns          How many samples in this data set?

### Value

Matrix of pretend counts.

### See Also

**limma stats DESeq**

### Examples

```
## Not run:
 pretend = make_exampledata()

## End(Not run)
```

---

make_gsc_from_abundant

*Given a pairwise result, make a gene set collection.*

---

### Description

If I want to play with gsva and friends, then I need GeneSetCollections! Much like make_gsc_from_significant(), this function extract the genes deemed 'abundant' and generates gene sets accordingly.

### Usage

```
make_gsc_from_abundant(
  pairwise,
  according_to = "deseq",
  orgdb = "org.Hs.eg.db",
  researcher_name = "elsayed",
  study_name = "macrophage",
  category_name = "infection",
  phenotype_name = NULL,
  pair_names = "high",
  current_id = "ENSEMBL",
  required_id = "ENTREZID",
  ...
)
```

### Arguments

| | |
|---|---|
| pairwise | A pairwise result, or combined de result, or extracted genes. |
| according_to | When getting significant genes, use this method. |
| orgdb | Annotation dataset. |
| researcher_name | |
| | Prefix of the name for the generated set(s). |
| study_name | Second element in the name of the generated set(s). |
| category_name | Third element in the name of the generated set(s). |
| phenotype_name | Optional phenotype data for the generated set(s). |
| pair_names | The suffix of the generated set(s). |
| current_id | What type of ID is the data currently using? |
| required_id | What type of ID should the use? |
| ... | Extra arguments for extract_abundant_genes(). |

## Value

List containing 3 GSCs, one containing both the highs/lows called 'colored', one of the highs, and one of the lows.

---

| make_gsc_from_ids | *Create a gene set collection from a set of arbitrary IDs.* |
|---|---|

---

## Description

This function attempts to simplify the creation of a gsva compatible GeneSet. Some important caveats when working with gsva, notably the gene IDs we use are not usually compatible with the gene IDs used by gsva, thus the primary logic in this function is intended to bridge these IDs.

## Usage

```
make_gsc_from_ids(
  first_ids,
  second_ids = NULL,
  orgdb = "org.Hs.eg.db",
  researcher_name = "elsayed",
  study_name = "macrophage",
  category_name = "infection",
  phenotype_name = NULL,
  pair_names = "up",
  current_id = "ENSEMBL",
  required_id = "ENTREZID"
)
```

## Arguments

| | |
|---|---|
| first_ids | The required IDs for a single set. |
| second_ids | Potentially null optionally used for a second, presumably contrasting set. |
| orgdb | Orgdb annotation, used to translate IDs to the required type. |
| researcher_name | |
| | Prefix of the name for the generated set(s). |
| study_name | Second element in the name of the generated set(s). |
| category_name | Third element in the name of the generated set(s). |
| phenotype_name | Optional phenotype data for the generated set(s). |
| pair_names | The suffix of the generated set(s). |
| current_id | What type of ID is the data currently using? |
| required_id | What type of ID should the use? |

## Value

Small list comprised of the created gene set collection(s).

make_gsc_from_pairwise

*Given a pairwise result, make a gene set collection.*

## Description

If I want to play with gsva and friends, then I need GeneSetCollections! To that end, this function uses extract_significant_genes() in order to gather sets of genes deemed 'significant'. It then passes these sets to make_gsc_from_ids().

## Usage

```
make_gsc_from_pairwise(
  pairwise,
  according_to = "deseq",
  orgdb = "org.Hs.eg.db",
  pair_names = c("ups", "downs"),
  category_name = "infection",
  phenotype_name = "parasite",
  set_name = "elsayed_macrophage",
  color = TRUE,
  current_id = "ENSEMBL",
  required_id = "ENTREZID",
  ...
)
```

## Arguments

| | |
|---|---|
| pairwise | A pairwise result, or combined de result, or extracted genes. |
| according_to | When getting significant genes, use this method. |
| orgdb | Annotation dataset. |
| pair_names | Describe the contrasts of the GSC: up vs. down, high vs. low, etc. |
| category_name | What category does the GSC describe? |
| phenotype_name | When making color sets, use this phenotype name. |
| set_name | A name for the created gene set. |
| color | Make a colorSet? |
| current_id | Usually we use ensembl IDs, but that does not _need_ to be the case. |
| required_id | GSVA uses entrezids by default. |
| ... | Extra arguments for extract_significant_genes(). |

## Value

List containing 3 GSCs, one containing both the ups/downs called 'colored', one of the ups, and one of the downs.

---

| make_id2gomap | *Make a go mapping from IDs in a format suitable for topGO.* |

---

## Description

When using a non-supported organism, one must write out mappings in the format expected by topgo. This handles that process and gives a summary of the new table.

## Usage

```
make_id2gomap(
  goid_map = "reference/go/id2go.map",
  go_db = NULL,
  overwrite = FALSE
)
```

## Arguments

| | |
|---|---|
| goid_map | TopGO mapping file. |
| go_db | If there is no goid_map, create it with this data frame. |
| overwrite | Rewrite the mapping file? |

## Value

Summary of the new goid table.

## See Also

**topGO**

---

| make_limma_tables | *Writes out the results of a limma search using toptable().* |

---

## Description

However, this will do a couple of things to make one's life easier: 1. Make a list of the output, one element for each comparison of the contrast matrix 2. Write out the toptable() output in separate .csv files and/or sheets in excel 3. Since I have been using qvalues a lot for other stuff, add a column for them.

**Usage**

```
make_limma_tables(
  fit = NULL,
  adjust = "BH",
  n = 0,
  coef = NULL,
  annot_df = NULL,
  intercept = FALSE
)
```

**Arguments**

| | |
|---|---|
| `fit` | Result from lmFit()/eBayes() |
| `adjust` | Pvalue adjustment chosen. |
| `n` | Number of entries to report, 0 says do them all. |
| `coef` | Which coefficients/contrasts to report, NULL says do them all. |
| `annot_df` | Optional data frame including annotation information to include with the tables. |
| `intercept` | Intercept model? |

**Value**

List of data frames comprising the toptable output for each coefficient, I also added a qvalue entry to these toptable() outputs.

**See Also**

**limma qvalue** `write_xlsx` `topTable`

**Examples**

```
## Not run:
 finished_comparison = eBayes(limma_output)
 table = make_limma_tables(finished_comparison, adjust="fdr")

## End(Not run)
```

---

make_pairwise_contrasts

*Run makeContrasts() with all pairwise comparisons.*

---

**Description**

In order to have uniformly consistent pairwise contrasts, I decided to avoid potential human erors(sic) by having a function generate all contrasts.

## Usage

```
make_pairwise_contrasts(
  model,
  conditions,
  do_identities = FALSE,
  do_extras = TRUE,
  do_pairwise = TRUE,
  extra_contrasts = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| model | Describe the conditions/batches/etc in the experiment. |
| conditions | Factor of conditions in the experiment. |
| do_identities | Include all the identity strings? Limma can use this information while edgeR can not. |
| do_pairwise | Include all pairwise strings? This shouldn't need to be set to FALSE, but just in case. |
| extra_contrasts | |
| | Optional string of extra contrasts to include. |
| ... | Extra arguments passed here are caught by arglist. |

## Details

Invoked by the _pairwise() functions.

## Value

List including the following information:

1. all_pairwise_contrasts = the result from makeContrasts(...)
2. identities = the string identifying each condition alone
3. all_pairwise = the string identifying each pairwise comparison alone
4. contrast_string = the string passed to R to call makeContrasts(...)
5. names = the names given to the identities/contrasts

## See Also

**limma** `makeContrasts`

## Examples

```
## Not run:
 pretend <- make_pairwise_contrasts(model, conditions)

## End(Not run)
```

---

make_pombe_expt                    *Create a Schizosaccharomyces cerevisiae expt.*

---

## Description

This just saves some annoying typing if one wishes to make a standard expressionset superclass out
of the publicly available fission data set.

## Usage

```
make_pombe_expt(annotation = TRUE)
```

## Arguments

annotation        Add annotation data?

## Value

Expressionset/expt of fission.

---

make_simplified_contrast_matrix
                              *Create a contrast matrix suitable for MSstats and similar tools.*

---

## Description

I rather like makeContrasts() from limma. I troubled me to have to manually create a contrast matrix
when using MSstats. It turns out it troubled me for good reason because I managed to reverse the
terms and end up with the opposite contrasts of what I intended. Ergo this function.

## Usage

```
make_simplified_contrast_matrix(numerators, denominators)
```

## Arguments

numerators        Character list of conditions which are the numerators of a series of a/b compar-
                  isons.
denominators      Character list of conditions which are the denominators of a series of a/b com-
                  parisons.

## Details

Feed make_simplified_contrast_matrix() a series of numerators and denominators names after the
conditions of interest in an experiment and it returns a contrast matrix in a format acceptable to
MSstats.

## Value

Contrast matrix suitable for use in tools like MSstats.

---

map_kegg_dbs                    *Maps KEGG identifiers to ENSEMBL gene ids.*

---

## Description

Takes a list of KEGG gene identifiers and returns a list of ENSEMBL ids corresponding to those genes.

## Usage

```
map_kegg_dbs(kegg_ids)
```

## Arguments

kegg_ids          List of KEGG identifiers to be mapped.

## Value

Ensembl IDs as a character list.

## See Also

**KEGGREST** `keggGet`

## Examples

```
kegg_df <- load_kegg_annotations(species="coli")
kegg_ids <- head(kegg_df[["kegg_geneid"]])
mapped <- map_kegg_dbs(kegg_ids)
mapped
```

---

map_orgdb_ids                   *Map AnnotationDbi keys from one column to another.*

---

## Description

Given a couple of keytypes, this provides a quick mapping across them. I might have an alternate version of this hiding in the gsva code, which requires ENTREZIDs. In the mean time, this creates a dataframe of the mapped columns for a given set of gene ids using the in a sqlite instance.

## Usage

```
map_orgdb_ids(orgdb, gene_ids = NULL, mapto = "ensembl", keytype = "geneid")
```

## Arguments

| | |
|---|---|
| `orgdb` | OrganismDb instance. |
| `gene_ids` | Gene identifiers for retrieving annotations. |
| `mapto` | Key to map the IDs against. |
| `keytype` | Choose a keytype, this will yell if it doesn't like your choice. |

## Value

a table of gene information

## Author(s)

Keith Hughitt with changes by atb.

## See Also

**AnnotationDbi** `select` `keytypes`

## Examples

```
dm_unigene_to_ensembl <- map_orgdb_ids("org.Dm.eg.db", mapto="ensembl", keytype="unigene")
head(dm_unigene_to_ensembl)
```

---

| mean_by_bioreplicate | *An attempt to address a troubling question when working with DIA data.* |
|---|---|

---

## Description

My biggest concern when treating DIA data in a RNASeqish manner is the fact that if a given peptide is not identified, that is not the same thing as stating that it was not translated. It is somewhat reminiscent of the often mocked and repeated Donald Rumsfeld statement regarding known unknowns vs. unknown unknowns. Thus, in an RNASeq experiment, if one sees a zero, one may assume that transcript was not transcribed, it may be assumed to be a known zero(unknown). In contrast, if the same thing happens in a DIA data set, that represents an unknown unknown. Perhaps it was not translated, and perhaps it was not identified.

## Usage

```
mean_by_bioreplicate(expt, fact = "bioreplicate", fun = "mean")
```

## Arguments

| | |
|---|---|
| `expt` | Starting expressionset to mangle. |
| `fact` | Metadata factor to use when taking the mean of biological replicates. |
| `fun` | Assumed to be mean, but one might want median. |

## Details

This function therefore does the following: 1. Backfill all 0s in the matrix to NA. 2. Performs a mean across all samples which are known technical replicates of the same biological replicate. This mean is performed using na.rm=TRUE. Thus the entries which used to be 0 should no longer affect the result. 3. Recreate the expressionset with the modified set of samples.

## Value

new expressionset

---

| median_by_factor | *Create a data frame of the medians of rows by a given factor in the data.* |
|---|---|

---

## Description

This assumes of course that (like expressionsets) there are separate columns for each replicate of the conditions. This will just iterate through the levels of a factor describing the columns, extract them, calculate the median, and add that as a new column in a separate data frame.

## Usage

```
median_by_factor(data, fact = "condition", fun = "median")
```

## Arguments

| | |
|---|---|
| data | Data frame, presumably of counts. |
| fact | Factor describing the columns in the data. |
| fun | Optionally choose mean or another function. |

## Details

Used in write_expt() as well as a few random collaborations.

## Value

Data frame of the medians.

## See Also

**Biobase matrixStats**

## Examples

```
## Not run:
 compressed = median_by_factor(data, experiment$condition)

## End(Not run)
```

---

model_test                          *Make sure a given experimental factor and design will play together.*

---

### Description

Have you ever wanted to set up a differential expression analysis and after minutes of the computer churning away it errors out with some weird error about rank? Then this is the function for you!

### Usage

```
model_test(design, goal = "condition", factors = NULL, ...)
```

### Arguments

| | |
|---|---|
| design | Dataframe describing the design of the experiment. |
| goal | Experimental factor you actually want to learn about. |
| factors | Experimental factors you rather wish would just go away. |
| ... | I might decide to add more options from other functions. |

### Value

List of booleans telling if the factors + goal will work.

### See Also

[model.matrix qr](model.matrix)

---

my_identifyAUBlocks            *copy/paste the function from SeqTools and figure out where it falls on its ass.*

---

### Description

Yeah, I do not remember what I changed in this function.

### Usage

```
my_identifyAUBlocks(x, min.length = 20, p.to.start = 0.8, p.to.end = 0.55)
```

### Arguments

| | |
|---|---|
| x | Sequence object |
| min.length | I dunno. |
| p.to.start | P to start of course |
| p.to.end | The p to end – wtf who makes names like this? |

## Value

a list of IRanges which contain a bunch of As and Us.

---

| my_isva | *There are some funky scoping problems in isva::DoISVA().* |
|---------|---|

---

## Description

Thus I copy/pasted the function and attempted to address them here.

## Usage

```
my_isva(
  data.m,
  pheno.v,
  cf.m = NULL,
  factor.log = FALSE,
  pvthCF = 0.01,
  th = 0.05,
  ncomp = NULL,
  icamethod = "fastICA"
)
```

## Arguments

| | |
|---|---|
| data.m | Input matrix. |
| pheno.v | Vector of conditions of interest in the data. |
| cf.m | Matrix of confounded conditions in the data. |
| factor.log | I forget. |
| pvthCF | Minimal p-value for considering. |
| th | threshold for inclusion. |
| ncomp | Number of SVA components to estimate. |
| icamethod | Which ICA implementation to use? |

---

| my_runsims | *A version of PROPER:::runsims which is (hopefully) a little more robust.* |
|---|---|

---

## Description

When I was testing PROPER, it fell down mysteriously on a few occasions. The source ended up being in runsims(), ergo this function.

## Usage

```
my_runsims(
  Nreps = c(3, 5, 7, 10),
  Nreps2,
  nsims = 100,
  sim.opts,
  DEmethod = c("edgeR", "DSS", "DESeq", "DESeq2"),
  verbose = TRUE
)
```

## Arguments

| Nreps | Vector of numbers of replicates to simulate. |
|---|---|
| Nreps2 | Second vector of replicates. |
| nsims | How many simulations to perform? |
| sim.opts | Options provided in a list which include information about the expression, numbers of genes, logFC values, etc. |
| DEmethod | I suggest using only either edgeR or DESeq2. |
| verbose | Print some information along the way? |

---

| mymakeContrasts | *A copy of limma::makeContrasts() with special sauce.* |
|---|---|

---

## Description

This is a copy of limma::makeContrasts without the test of make.names() Because I want to be able to use it with interaction models potentially and if a model has first:second, make.names() turns the ':' to a '.' and then the equivalence test fails, causing makeContrasts() to error spuriously (I think).

## Usage

```
mymakeContrasts(..., contrasts = NULL, levels)
```

## Arguments

| | |
|---|---|
| `...` | Conditions used to make the contrasts. |
| `contrasts` | Actual contrast names. |
| `levels` | contrast levels used. |

## Value

Same contrasts as used in makeContrasts, but with unique names.

---

| myretrieveKGML | *A couple functions from KEGGgraph that have broken* |
|---|---|

---

## Description

Some material in KEGGREST is borken.

## Usage

```
myretrieveKGML(
  pathway,
  organism,
  destfile,
  silent = TRUE,
  hostname = "http://www.kegg.jp",
  ...
)
```

## Arguments

| | |
|---|---|
| `pathway` | The path to query. |
| `organism` | Which organism to query? |
| `destfile` | File to which to download. |
| `silent` | Send stdout and stderr to dev null? |
| `hostname` | Host to download from (this is what is broken.) |
| `...` | Arglist! |

---

normalize_counts            *Perform a simple normalization of a count table.*

---

### Description

This provides shortcut interfaces for normalization functions from deseq2/edger and friends.

### Usage

```
normalize_counts(data, design = NULL, method = "raw", ...)
```

### Arguments

| | |
|---|---|
| `data` | Matrix of count data. |
| `design` | Dataframe describing the experimental design. (conditions/batches/etc) |
| `...` | More arguments might be necessary. |
| `norm` | Normalization to perform: 'sf|quant|qsmooth|tmm|upperquartile|tmm|rle' I keep wishy-washing on whether design is a required argument. |

### Value

Dataframe of normalized(counts)

### See Also

**edgeR limma DESeq2**

### Examples

```
## Not run:
 norm_table = normalize_counts(count_table, design=design, norm='qsmooth')

## End(Not run)
```

---

normalize_expt              *Normalize the data of an expt object. Save the original data, and note what was done.*

---

### Description

It is the responsibility of normalize_expt() to perform any arbitrary normalizations desired as well as to ensure that the data integrity is maintained. In order to do this, it writes the actions performed in expt$state and saves the intermediate steps of the normalization in expt$intermediate_counts. Furthermore, it should tell you every step of the normalization process, from count filtering, to normalization, conversion, transformation, and batch correction.

## Usage

```
normalize_expt(
  expt,
  transform = "raw",
  norm = "raw",
  convert = "raw",
  batch = "raw",
  filter = FALSE,
  annotations = NULL,
  fasta = NULL,
  entry_type = "gene",
  use_original = FALSE,
  batch1 = "batch",
  batch2 = NULL,
  batch_step = 4,
  low_to_zero = TRUE,
  thresh = 2,
  min_samples = 2,
  p = 0.01,
  A = 1,
  k = 1,
  cv_min = 0.01,
  cv_max = 1000,
  na_to_zero = FALSE,
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| expt | Original expt. |
| transform | Transformation desired, usually log2. |
| norm | How to normalize the data? (raw, quant, sf, upperquartile, tmm, rle) |
| convert | Conversion to perform? (raw, cpm, rpkm, cp_seq_m) |
| batch | Batch effect removal tool to use? (limma sva fsva ruv etc) |
| filter | Filter out low/undesired features? (cbcb, pofa, kofa, others?) |
| annotations | Used for rpkm – probably not needed as this is in fData now. |
| fasta | Fasta file for cp_seq_m counting of oligos. |
| entry_type | For getting genelengths by feature type (rpkm or cp_seq_m). |
| use_original | Use the backup data in the expt class? |
| batch1 | Experimental factor to extract first. |
| batch2 | Second factor to remove (only with limma's removebatcheffect()). |
| batch_step | From step 1-5, when should batch correction be applied? |
| low_to_zero | When log transforming, change low numbers (< 0) to 0 to avoid NaN? |

| thresh | Used by cbcb_lowfilter(). |
|---|---|
| min_samples | Also used by cbcb_lowfilter(). |
| p | Used by genefilter's pofa(). |
| A | Also used by genefilter's pofa(). |
| k | Used by genefilter's kofa(). |
| cv_min | Used by genefilter's cv(). |
| cv_max | Also used by genefilter's cv(). |
| na_to_zero | Sometimes rpkm gives some NA values for very low numbers. |
| verbose | Print what is happening while the normalization is performed? I am not sure why, but I think they should be 0. |
| ... | more options |

## Value

Expt object with normalized data and the original data saved as 'original_expressionset'

## See Also

**genefilter limma sva edgeR DESeq2**

## Examples

```
## Not run:
 normed <- normalize_expt(exp, transform='log2', norm='rle', convert='cpm',
                          batch='raw', filter='pofa')
 normed_batch <- normalize_expt(exp, transform='log2', norm='rle', convert='cpm',
                                batch='sva', filter='pofa')

## End(Not run)
```

---

| orgdb_from_ah | *Get an orgdb from an AnnotationHub taxonID.* |
|---|---|

---

## Description

Ideally, annotationhub will one day provide a one-stop shopping source for a tremendous wealth of curated annotation databases, sort of like a non-obnoxious biomart. But for the moment, this function is more fragile than I would like.

## Usage

```
orgdb_from_ah(ahid = NULL, title = NULL, species = NULL, type = "OrgDb")
```

## Arguments

| | |
|---|---|
| `ahid` | TaxonID from AnnotationHub |
| `title` | Title for the annotation hub instance |
| `species` | Species to download |
| `type` | Datatype to download |

## Value

An Orgdb instance

## See Also

**AnnotationHub S4Vectors**

## Examples

```
## Not run:
 org <- mytaxIdToOrgDb(species="Leishmania", type="TxDb")

## End(Not run)
```

---

pattern_count_genome      *Find how many times a given pattern occurs in every gene of a genome.*

---

## Description

There are times when knowing how many times a given string appears in a genome/CDS is helpful. This function provides that information and is primarily used by cp_seq_m().

## Usage

```
pattern_count_genome(
  fasta,
  gff = NULL,
  pattern = "TA",
  type = "gene",
  key = NULL
)
```

## Arguments

| | |
|---|---|
| `fasta` | Genome sequence. |
| `gff` | Gff of annotation information from which to acquire CDS (if not provided it will just query the entire genome). |
| `pattern` | What to search for? This was used for tnseq and TA is the mariner insertion point. |
| `type` | Column to use in the gff file. |
| `key` | What type of entry of the gff file to key from? |

## Details

This is once again a place where mcols() usage might improve the overall quality of life.

## Value

Data frame of gene names and number of times the pattern appears/gene.

## See Also

**Biostrings Rsamtools Rsamtools** `FaFile getSeq PDict vcountPDict`

## Examples

```
fasta_file <- system.file("paeruginosa_pa14.fasta", package="hpgltools")
gff_file <- system.file("paeruginosa_pa14.gff", package="hpgltools")
ta_count <- pattern_count_genome(fasta_file, gff_file)
head(ta_count)
```

---

pca_highscores                *Get the highest/lowest scoring genes for every principle component.*

---

## Description

This function uses princomp to acquire a principle component biplot for some data and extracts a dataframe of the top n genes for each component by score.

## Usage

```
pca_highscores(expt, n = 20, cor = TRUE, vs = "means", logged = TRUE)
```

## Arguments

| | |
|---|---|
| expt | Experiment to poke. |
| n | Number of genes to extract. |
| cor | Perform correlations? |
| vs | Do a mean or median when getting ready to perform the pca? |
| logged | Check for the log state of the data and adjust as deemed necessary? |

## Value

a list including the princomp biplot, histogram, and tables of top/bottom n scored genes with their scores by component.

## See Also

**stats** `princomp`

## Examples

```
## Not run:
 information <- pca_highscores(df=df, conditions=cond, batches=bat)
 information$pca_bitplot  ## oo pretty

## End(Not run)
```

---

pca_information          *Gather information about principle components.*

---

## Description

Calculate some information useful for generating PCA plots. pca_information seeks to gather to-
gether interesting information to make principle component analyses easier, including: the results
from (fast.)svd, a table of the r^2 values, a table of the variances in the data, coordinates used to
make a pca plot for an arbitrarily large set of PCs, correlations and fstats between experimental
factors and the PCs, and heatmaps describing these relationships. Finally, it will provide a plot
showing how much of the variance is provided by the top-n genes and (optionally) the set of all
PCA plots with respect to one another. (PCx vs. PCy)

## Usage

```
pca_information(
  expt,
  expt_design = NULL,
  expt_factors = c("condition", "batch"),
  num_components = NULL,
  plot_pcas = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| expt | Data to analyze (usually exprs(somedataset)). |
| expt_design | Dataframe describing the experimental design, containing columns with useful information like the conditions, batches, number of cells, whatever... |
| expt_factors | Character list of experimental conditions to query for R^2 against the fast.svd of the data. |
| num_components | Number of principle components to compare the design factors against. If left null, it will query the same number of components as factors asked for. |
| plot_pcas | Plot the set of PCA plots for every pair of PCs queried. |
| ... | Extra arguments for the pca plotter |

**Value**

a list of fun pca information: svd_u/d/v: The u/d/v parameters from fast.svd rsquared_table: A table of the rsquared values between each factor and principle component pca_variance: A table of the pca variances pca_data: Coordinates for a pca plot pca_cor: A table of the correlations between the factors and principle components anova_fstats: the sum of the residuals with the factor vs without (manually calculated) anova_f: The result from performing anova(withfactor, withoutfactor), the F slot anova_p: The p-value calculated from the anova() call anova_sums: The RSS value from the above anova() call cor_heatmap: A heatmap from recordPlot() describing pca_cor.

**Warning**

This function has gotten too damn big and needs to be split up.

**See Also**

**corpcor stats** `fast.svd`, `lm`

**Examples**

```
## Not run:
 pca_info = pca_information(exprs(some_expt$expressionset), some_design, "all")
 pca_info

## End(Not run)
```

---

| | |
|---|---|
| pct_all_kegg | *Extract the percent differentially expressed genes for all KEGG pathways.* |

---

**Description**

KEGGgraph provides some interesting functionality for mapping KEGGids and examining the pieces. This attempts to use that in order to evaluate how many 'significant' genes are in a given pathway.

**Usage**

```
pct_all_kegg(
  all_ids,
  sig_ids,
  organism = "dme",
  pathways = "all",
  pathdir = "kegg_pathways",
  verbose = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `all_ids` | Set of all gene IDs in a given analysis. |
| `sig_ids` | Set of significant gene IDs. |
| `organism` | KEGG organism identifier. |
| `pathways` | What pathways to look at? |
| `pathdir` | Directory into which to copy downloaded pathway files. |
| `verbose` | Talky talky? |
| `...` | Options I might pass from other functions are dropped into arglist. |

## Value

Dataframe including the filenames, percentages, nodes included, and differential nodes.

## See Also

**KEGGgraph KEGGREST**

---

| | |
|---|---|
| `pct_kegg_diff` | *Extract the percent differentially expressed genes in a given KEGG pathway.* |

---

## Description

KEGGgraph provides some interesting functionality for mapping KEGGids and examining the pieces. This attempts to use that in order to evaluate how many 'significant' genes are in a given pathway.

## Usage

```
pct_kegg_diff(
  all_ids,
  sig_ids,
  pathway = "00500",
  organism = "dme",
  pathdir = "kegg_pathways",
  ...
)
```

## Arguments

| | |
|---|---|
| `all_ids` | Set of all gene IDs in a given analysis. |
| `sig_ids` | Set of significant gene IDs. |
| `pathway` | Numeric pathway identifier. |
| `organism` | KEGG organism identifier. |
| `pathdir` | Directory into which to copy downloaded pathway files. |
| `...` | Options I might pass from other functions are dropped into arglist. |

## Value

Percent genes/pathway deemed significant.

## See Also

**KEGGgraph KEGGREST**

---

please_install                     *Automatic loading and/or installing of packages.*

---

## Description

Load a library, install it first if necessary.

## Usage

```
please_install(lib, update = FALSE)
```

## Arguments

lib             String name of a library to check/install.

update          Update packages?

## Details

This was taken from: http://sbamin.com/2012/11/05/tips-for-working-in-r-automatically-install-missing-package/ and initially provided by Ramzi Temanni.

## Value

0 or 1, whether a package was installed or not.

## See Also

**BiocManager** install install.packages

## Examples

```
## Not run:
 require.auto("ggplot2")

## End(Not run)
```

---

plot_3d_pca                    *Something silly for Najib.*

---

### Description

This will make him very happy, but I remain skeptical.

### Usage

```
plot_3d_pca(pc_result, components = c(1, 2, 3), file = "3dpca.html")
```

### Arguments

| | |
|---|---|
| pc_result | The result from plot_pca() |
| components | List of three axes by component. |
| file | File to write the created plotly object. |

---

plot_batchsv                   *Make a dotplot of known batches vs. SVs.*

---

### Description

This should make a quick df of the factors and surrogates and plot them. Maybe it should be folded into plot_svfactor? Hmm, I think first I will write this and see if it is better.

### Usage

```
plot_batchsv(
  expt,
  svs,
  sv = 1,
  batch_column = "batch",
  factor_type = "factor",
  id_column = "sampleid"
)
```

### Arguments

| | |
|---|---|
| expt | Experiment from which to acquire the design, counts, etc. |
| svs | Set of surrogate variable estimations from sva/svg or batch estimates. |
| sv | Which surrogate variable to show? |
| batch_column | Which experimental design column to use? |
| factor_type | This may be a factor or range, it is intended to plot a scatterplot if it is a range, a dotplot if a factor. |

## Value

Plot of batch vs surrogate variables as per Leek's work.

## See Also

**sva ggplot2**

## Examples

```
## Not run:
 estimate_vs_snps <- plot_batchsv(start, surrogate_estimate, "snpcategory")

## End(Not run)
```

---

plot_bcv                          *Steal edgeR's plotBCV() and make it a ggplot2.*

---

## Description

This was written primarily to understand what that function is doing in edgeR.

## Usage

```
plot_bcv(data)
```

## Arguments

data              A dataframe/expt/exprs with count data

## Value

a plot! of the BCV a la ggplot2.

## See Also

**edgeR** `plotBCV`

## Examples

```
## Not run:
 bcv <- plot_bcv(expt)
 summary(bcv$data)
 bcv$plot

## End(Not run)
```

---

plot_boxplot *Make a ggplot boxplot of a set of samples.*

---

### Description

Boxplots and density plots provide complementary views of data distributions. The general idea is that if the box for one sample is significantly shifted from the others, then it is likely an outlier in the same way a density plot shifted is an outlier.

### Usage

```
plot_boxplot(
  data,
  colors = NULL,
  title = NULL,
  violin = FALSE,
  scale = NULL,
  expt_names = NULL,
  label_chars = 10,
  ...
)
```

### Arguments

| | |
|---|---|
| `data` | Expt or data frame set of samples. |
| `colors` | Color scheme, if not provided will make its own. |
| `title` | A title! |
| `violin` | Print this as a violin rather than a just box/whiskers? |
| `scale` | Whether to log scale the y-axis. |
| `expt_names` | Another version of the sample names for printing. |
| `label_chars` | Maximum number of characters for abbreviating sample names. |
| `...` | More parameters are more fun! |

### Value

Ggplot2 boxplot of the samples. Each boxplot contains the following information: a centered line describing the median value of counts of all genes in the sample, a box around the line describing the inner-quartiles around the median (quartiles 2 and 3 for those who are counting), a vertical line above/below the box which shows 1.5x the inner quartile range (a common metric of the non-outliers), and single dots for each gene which is outside that range. A single dot is transparent.

### See Also

**ggplot2 reshape2** geom_boxplot melt scale_x_discrete

## Examples

```
## Not run:
 a_boxplot <- plot_boxplot(expt)
 a_boxplot  ## ooo pretty boxplot look at the lines

## End(Not run)
```

---

| plot_cleaved | *Plot the average mass and expected intensity of a set of sequences given an enzyme.* |
|---|---|

---

## Description

This uses the cleaver package to generate a plot of expected intensities vs. weight for a list of protein sequences.

## Usage

```
plot_cleaved(pep_sequences, enzyme = "trypsin", start = 600, end = 1500)
```

## Arguments

| | |
|---|---|
| pep_sequences | Set of protein sequences. |
| enzyme | One of the allowed enzymes for cleaver. |
| start | Limit the set of fragments from this point |
| end | to this point. |

## Value

List containing the distribution of weights and the associated plot.

---

| plot_corheat | *Make a heatmap.3 description of the correlation between samples.* |
|---|---|

---

## Description

Given a set of count tables and design, this will calculate the pairwise correlations and plot them as a heatmap. It attempts to standardize the inputs and eventual output.

## Usage

```
plot_corheat(
  expt_data,
  expt_colors = NULL,
  expt_design = NULL,
  method = "pearson",
  expt_names = NULL,
  batch_row = "batch",
  title = NULL,
  label_chars = 10,
  ...
)
```

## Arguments

| | |
|---|---|
| expt_data | Dataframe, expt, or expressionset to work with. |
| expt_colors | Color scheme for the samples, not needed if this is an expt. |
| expt_design | Design matrix describing the experiment, not needed if this is an expt. |
| method | Correlation statistic to use. (pearson, spearman, kendall, robust). |
| expt_names | Alternate names to use for the samples. |
| batch_row | Name of the design row used for 'batch' column colors. |
| title | Title for the plot. |
| label_chars | Limit on the number of label characters. |
| ... | More options are wonderful! |

## Value

Gplots heatmap describing describing how the samples are clustering vis a vis pairwise correlation.

## See Also

**grDevice** `hpgl_cor` `brewer.pal` `recordPlot`

## Examples

```
## Not run:
 corheat_plot <- hpgl_corheat(expt=expt, method="robust")

## End(Not run)
```

---

plot_de_pvals                    *Given a DE table with p-values, plot them.*

---

### Description

Plot a multi-histogram containing (adjusted)p-values.

### Usage

```
plot_de_pvals(
  combined_data,
  type = "limma",
  p_type = "both",
  columns = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| combined_data | Table to extract the values from. |
| type | If provided, extract the type_p and type_adjp columns. |
| p_type | Which type of pvalue to show (adjusted, raw, or all)? |
| columns | Otherwise, extract whatever columns are provided. |
| ... | Arguments passed through to the histogram plotter |

### Details

The assumption of this plot is that the adjustment will significantly decrease the representation of genes in the 'highly significant' range of p-values. However, it is hoped that it will not utterly remove them.

### Value

Multihistogram of the result.

---

plot_density                    *Create a density plot, showing the distribution of each column of data.*

---

### Description

Density plots and boxplots are cousins and provide very similar views of data distributions. Some people like one, some the other. I think they are both colorful and fun!

## Usage

```
plot_density(
  data,
  colors = NULL,
  expt_names = NULL,
  position = "identity",
  direct = TRUE,
  fill = NULL,
  title = NULL,
  scale = NULL,
  colors_by = "condition",
  label_chars = 10,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Expt, expressionset, or data frame. |
| colors | Color scheme to use. |
| expt_names | Names of the samples. |
| position | How to place the lines, either let them overlap (identity), or stack them. |
| direct | Use direct.labels for labeling the plot? |
| fill | Fill the distributions? This might make the plot unreasonably colorful. |
| title | Title for the plot. |
| scale | Plot on the log scale? |
| colors_by | Factor for coloring the lines |
| label_chars | Maximum number of characters in sample names before abbreviation. |
| ... | sometimes extra arguments might come from graph_metrics() |

## Value

ggplot2 density plot!

## See Also

**ggplot2** `geom_density`

## Examples

```
## Not run:
 funkytown <- plot_density(data)

## End(Not run)
```

plot_disheat                 *Make a heatmap.3 of the distances (euclidean by default) between*
                             *samples.*

### Description

Given a set of count tables and design, this will calculate the pairwise distances and plot them as a
heatmap. It attempts to standardize the inputs and eventual output.

### Usage

```
plot_disheat(
  expt_data,
  expt_colors = NULL,
  expt_design = NULL,
  method = "euclidean",
  expt_names = NULL,
  batch_row = "batch",
  title = NULL,
  label_chars = 10,
  ...
)
```

### Arguments

| | |
|---|---|
| expt_data | Dataframe, expt, or expressionset to work with. |
| expt_colors | Color scheme (not needed if an expt is provided). |
| expt_design | Design matrix (not needed if an expt is provided). |
| method | Distance metric to use. |
| expt_names | Alternate names to use for the samples. |
| batch_row | Name of the design row used for 'batch' column colors. |
| title | Title for the plot. |
| label_chars | Limit on the number of label characters. |
| ... | More parameters! |

### Value

a recordPlot() heatmap describing the distance between samples.

### See Also

**RColorBrewer** `brewer.pal` `heatmap.2` `recordPlot`

### Examples

```
## Not run:
 disheat_plot = plot_disheat(expt=expt, method="euclidean")

## End(Not run)
```

---

plot_dist_scatter     *Make a scatter plot between two sets of numbers with a cheesy distance metric and some statistics of the two sets.*

---

### Description

The distance metric should be codified and made more intelligent. Currently it creates a dataframe of distances which are absolute distances from each axis, multiplied by each other, summed by axis, then normalized against the maximum.

### Usage

```
plot_dist_scatter(
  df,
  tooltip_data = NULL,
  gvis_filename = NULL,
  size = 2,
  xlab = NULL,
  ylab = NULL
)
```

### Arguments

| | |
|---|---|
| df | Dataframe likely containing two columns. |
| tooltip_data | Df of tooltip information for gvis graphs. |
| gvis_filename | Filename to write a fancy html graph. |
| size | Size of the dots. |
| xlab | x-axis label. |
| ylab | y-axis label. |

### Value

Ggplot2 scatter plot. This plot provides a "bird's eye" view of two data sets. This plot assumes the two data structures are not correlated, and so it calculates the median/mad of each axis and uses these to calculate a stupid, home-grown distance metric away from both medians. This distance metric is used to color dots which are presumed the therefore be interesting because they are far from 'normal.' This will make a fun clicky googleVis graph if requested.

### See Also

**ggplot2** `plot_gvis_scatter` `geom_point` `plot_linear_scatter`

**Examples**

```
## Not run:
 dist_scatter(lotsofnumbers_intwo_columns, tooltip_data=tooltip_dataframe,
                    gvis_filename="html/fun_scatterplot.html")

## End(Not run)
```

---

  plot_epitrochoid      *Make epitrochoid plots!*

---

**Description**

7, 2, 6, 7 should give a pretty result.

**Usage**

```
plot_epitrochoid(
  radius_a = 7,
  radius_b = 2,
  dist_b = 6,
  revolutions = 7,
  increments = 6480
)
```

**Arguments**

| | |
|---|---|
| radius_a | Radius of the major circle |
| radius_b | And the smaller circle. |
| dist_b | between b and the drawing point. |
| revolutions | How many times to revolve through the spirograph. |
| increments | How many dots to lay down while writing. |

---

  plot_essentiality      *Plot the essentiality of a library as per DeJesus et al.*

---

**Description**

This provides a plot of the essentiality metrics 'zbar' with respect to gene. In my pipeline, I use their stand alone mh_ess and tn_hmm packages. The result files produced are named mh_ess-sequence_prefix-mapping_parameters_gene_tas_m_parameter.csv where sequence_prefix is the base-name() of the input sequence file, mapping_parameters are a string describing the bowtie mapping used, and m_parameter is usually one of 1,2,4,8,16,32 and defines the lower limit of read depth to be considered useful by the mh_ess package. Thus, before using this, one may want to look at the result from tnseq_saturation() to see if there is a most-appropriate m_parameter. I think I should figure out a heuristic to choose the m, but I am not sure what it would be, perhaps the median of the hits summary?

## Usage

```
plot_essentiality(
  file,
  order_by = "posterior_zbar",
  keep_esses = FALSE,
  min_sig = 0.0371,
  max_sig = 0.9902
)
```

## Arguments

| | |
|---|---|
| file | Result from the DeJesus essentiality package. I think this has been effectively replaced by their TRANSIT package. |
| order_by | What column to use when ordering the data? |
| keep_esses | Keep entries in the data which are 'S' meaning insufficient evidence. |
| min_sig | Minimal value below which a gene is deemed non-essential and above which it is uncertain. |
| max_sig | Maximum value above which a gene is deemed essential and below which it is uncertain. |

## Value

A couple of plots

## See Also

**ggplot2**

---

plot_fun_venn                 *A quick wrapper around venneuler to help label stuff*

---

## Description

venneuler makes pretty venn diagrams, but no labels!

## Usage

```
plot_fun_venn(
  ones = c(),
  twos = c(),
  threes = c(),
  fours = c(),
  fives = c(),
  factor = 0.9
)
```

## Arguments

| | |
|---|---|
| `ones` | Character list of singletone categories |
| `twos` | Character list of doubletone categories |
| `threes` | Character list of tripletone categories |
| `fours` | Character list of quad categories |
| `fives` | Character list of quint categories |
| `factor` | Currently unused, but intended to change the radial distance to the label from the center of each circle. |

## Value

Two element list containing the venneuler data and the plot.

## See Also

**venneuler**

---

| `plot_goseq_pval` | *Make a pvalue plot from goseq data.* |
|---|---|

---

## Description

With minor changes, it is possible to push the goseq results into a clusterProfiler-ish pvalue plot. This handles those changes and returns the ggplot results.

## Usage

```
plot_goseq_pval(
  goterms,
  wrapped_width = 30,
  cutoff = 0.1,
  n = 30,
  mincat = 5,
  level = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `goterms` | Some data from goseq! |
| `wrapped_width` | Number of characters before wrapping to help legibility. |
| `cutoff` | Pvalue cutoff for the plot. |
| `n` | How many groups to include? |
| `mincat` | Minimum size of the category for inclusion. |
| `level` | Levels of the ontology tree to use. |
| `...` | Arguments passed from simple_goseq() |

**Value**

Plots!

**See Also**

**goseq clusterProfiler** `goseq` `plot_ontpval`

---

| plot_gostats_pval | *Make a pvalue plot similar to that from clusterprofiler from gostats data.* |
|---|---|

---

**Description**

clusterprofiler provides beautiful plots describing significantly overrepresented categories. This function attempts to expand the repetoire of data available to them to include data from gostats. The pval_plot function upon which this is based now has a bunch of new helpers now that I understand how the ontology trees work better, this should take advantage of that, but currently does not.

**Usage**

```
plot_gostats_pval(
  gs_result,
  wrapped_width = 20,
  cutoff = 0.1,
  n = 30,
  group_minsize = 5
)
```

**Arguments**

| | |
|---|---|
| gs_result | Ontology search results. |
| wrapped_width | Make the text large enough to read. |
| cutoff | What is the maximum pvalue allowed? |
| n | How many groups to include in the plot? |
| group_minsize | Minimum group size before inclusion. |

**Value**

Plots!

**See Also**

**clusterProfiler** `plot_ontpval`

---

plot_gprofiler_pval          *Make a pvalue plot from gprofiler data.*

---

### Description

The p-value plots from clusterProfiler are pretty, this sets the gprofiler data into a format suitable for plotting in that fashion and returns the resulting plots of significant ontologies.

### Usage

```
plot_gprofiler_pval(
  gp_result,
  wrapped_width = 30,
  cutoff = 0.1,
  n = 30,
  group_minsize = 5,
  scorer = "recall",
  ...
)
```

### Arguments

| | |
|---|---|
| gp_result | Some data from gProfiler. |
| wrapped_width | Maximum width of the text names. |
| cutoff | P-value cutoff for the plots. |
| n | Maximum number of ontologies to include. |
| group_minsize | Minimum ontology group size to include. |
| scorer | Which column to use for scoring the data. |
| ... | Options I might pass from other functions are dropped into arglist. |

### Value

List of MF/BP/CC pvalue plots.

### See Also

**topgo clusterProfiler**

---

| | |
|---|---|
| plot_heatmap | *Make a heatmap.3 plot, does the work for plot_disheat and plot_corheat.* |

---

### Description

This does what is says on the tin. Sets the colors for correlation or distance heatmaps, handles the calculation of the relevant metrics, and plots the heatmap.

### Usage

```
plot_heatmap(
  expt_data,
  expt_colors = NULL,
  expt_design = NULL,
  method = "pearson",
  expt_names = NULL,
  type = "correlation",
  batch_row = "batch",
  title = NULL,
  label_chars = 10,
  ...
)
```

### Arguments

| | |
|---|---|
| expt_data | Dataframe, expt, or expressionset to work with. |
| expt_colors | Color scheme for the samples. |
| expt_design | Design matrix describing the experiment vis a vis conditions and batches. |
| method | Distance or correlation metric to use. |
| expt_names | Alternate names to use for the samples. |
| type | Defines the use of correlation, distance, or sample heatmap. |
| batch_row | Name of the design row used for 'batch' column colors. |
| title | Title for the plot. |
| label_chars | Limit on the number of label characters. |
| ... | I like elipses! |

### Value

a recordPlot() heatmap describing the distance between samples.

### See Also

**RColorBrewer** `brewer.pal recordPlot`

plot_heatplus                    *Potential replacement for heatmap.2 based plots.*

## Description

Heatplus is an interesting tool, I have a few examples of using it and intend to include them here.

## Usage

```
plot_heatplus(
  expt,
  type = "correlation",
  method = "pearson",
  annot_columns = "batch",
  annot_rows = "condition",
  cutoff = 1,
  cluster_colors = NULL,
  scale = "none",
  cluster_width = 2,
  cluster_function = NULL,
  heatmap_colors = NULL
)
```

## Arguments

| | |
|---|---|
| expt | Experiment to try plotting. |
| type | What comparison method to use on the data (distance or correlation)? |
| method | What distance/correlation method to perform? |
| annot_columns | Set of columns to include as terminal columns next to the heatmap. |
| annot_rows | Set of columns to include as terminal rows below the heatmap. |
| cutoff | Cutoff used to define color changes in the annotated clustering. |
| cluster_colors | Choose colors for the clustering? |
| scale | Scale the heatmap colors? |
| cluster_width | How much space to include between clustering? |
| cluster_function | |
| | Choose an alternate clustering function than hclust()? |
| heatmap_colors | Choose your own heatmap cluster palette? |

## Value

List containing the returned heatmap along with some parameters used to create it.

---

plot_histogram                    *Make a pretty histogram of something.*

---

### Description

A shortcut to make a ggplot2 histogram which makes an attempt to set reasonable bin widths and set the scale to log if that seems a good idea.

### Usage

```
plot_histogram(
  df,
  binwidth = NULL,
  log = FALSE,
  bins = 500,
  adjust = 1,
  fillcolor = "darkgrey",
  color = "black"
)
```

### Arguments

| | |
|---|---|
| df | Dataframe of lots of pretty numbers. |
| binwidth | Width of the bins for the histogram. |
| log | Replot on the log scale? |
| bins | Number of bins for the histogram. |
| adjust | The prettification parameter in the ggplot2 density. |
| fillcolor | Change the fill colors of the plotted elements? |
| color | Change the color of the lines of the plotted elements? |

### Value

Ggplot histogram.

### See Also

**ggplot2** `geom_histogram` `geom_density`

### Examples

```
## Not run:
 kittytime = plot_histogram(df)

## End(Not run)
```

---

plot_hypotrochoid          *Make hypotrochoid plots!*

---

## Description

3,7,1 should give the classic 7 leaf clover

## Usage

```
plot_hypotrochoid(
  radius_a = 3,
  radius_b = 7,
  dist_b = 1,
  revolutions = 7,
  increments = 6480
)
```

## Arguments

| | |
|---|---|
| radius_a | Radius of the major circle |
| radius_b | And the smaller circle. |
| dist_b | between b and the drawing point. |
| revolutions | How many times to revolve through the spirograph. |
| increments | How many dots to lay down while writing. |

---

plot_intensity_mz          *Plot mzXML peak intensities with respect to m/z.*

---

## Description

I want to have a pretty plot of peak intensities and m/z. The plot provided by this function is interesting, but suffers from some oddities; notably that it does not currently separate the MS1 and MS2 data.

## Usage

```
plot_intensity_mz(
  mzxml_data,
  loess = FALSE,
  alpha = 0.5,
  ms1 = TRUE,
  ms2 = TRUE,
  x_scale = NULL,
  y_scale = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `mzxml_data` | The data structure from extract_mzxml or whatever it is. |
| `loess` | Do a loess smoothing from which to extract a function describing the data? This is terribly slow, and in the data I have examined so far, not very helpful, so it is FALSE by default. |
| `alpha` | Make the plotted dots opaque to this degree. |
| `ms1` | Include MS1 data in the plot? |
| `ms2` | Include MS2 data in the plot? |
| `x_scale` | Plot the x-axis on a non linear scale? |
| `y_scale` | Plot the y-axis on a non linear scale? |
| `...` | Extra arguments for the downstream functions. |

## Value

ggplot2 goodness.

---

| plot_legend | *Scab the legend from a PCA plot and print it alone* |
|---|---|

---

## Description

This way I can have a legend object to move about.

## Usage

```
plot_legend(stuff)
```

## Arguments

| | |
|---|---|
| `stuff` | This can take either a ggplot2 pca plot or some data from which to make one. |

## Value

A legend!

plot_libsize                    *Make a ggplot graph of library sizes.*

### Description

It is often useful to have a quick view of which samples have more/fewer reads. This does that and maintains one's favorite color scheme and tries to make it pretty!

### Usage

```
plot_libsize(
  data,
  condition = NULL,
  colors = NULL,
  text = TRUE,
  order = NULL,
  title = NULL,
  yscale = NULL,
  expt_names = NULL,
  label_chars = 10,
  ...
)
```

### Arguments

| | |
|---|---|
| data | Expt, dataframe, or expressionset of samples. |
| condition | vector of sample condition names. |
| colors | Color scheme if the data is not an expt. |
| text | Add the numeric values inside the top of the bars of the plot? |
| order | Explicitly set the order of samples in the plot? |
| title | Title for the plot. |
| yscale | Whether or not to log10 the y-axis. |
| expt_names | Design column or manually selected names for printing sample names. |
| label_chars | Maximum number of characters before abbreviating sample names. |
| ... | More parameters for your good time! |

### Value

a ggplot2 bar plot of every sample's size

### See Also

**ggplot2** geom_bar geom_text prettyNum scale_y_log10

## Examples

```
## Not run:
 libsize_plot <- plot_libsize(expt=expt)
 libsize_plot  ## ooo pretty bargraph

## End(Not run)
```

---

| plot_libsize_prepost | *Thanks to Sandra Correia for this! This function attempts to represent the change in the number of genes which are well/poorly represented in the data before and after performing a low-count filter.* |

---

## Description

Thanks to Sandra Correia for this! This function attempts to represent the change in the number of genes which are well/poorly represented in the data before and after performing a low-count filter.

## Usage

```
plot_libsize_prepost(expt, low_limit = 2, filter = TRUE, ...)
```

## Arguments

| | |
|---|---|
| expt | Input expressionset. |
| low_limit | A threshold to define 'low-representation.' |
| filter | Method used to low-count filter the data. |
| ... | Extra arbitrary arguments to pass to normalize_expt() |

## Value

Bar plot showing the number of genes below the low_limit before and after filtering the data.

---

| plot_linear_scatter | *Make a scatter plot between two groups with a linear model superimposed and some supporting statistics.* |

---

## Description

Make a scatter plot between two groups with a linear model superimposed and some supporting statistics.

**Usage**

```
plot_linear_scatter(
  df,
  tooltip_data = NULL,
  gvis_filename = NULL,
  cormethod = "pearson",
  size = 2,
  loess = FALSE,
  identity = FALSE,
  gvis_trendline = NULL,
  z_lines = FALSE,
  first = NULL,
  second = NULL,
  base_url = NULL,
  pretty_colors = TRUE,
  xlab = NULL,
  ylab = NULL,
  color_high = NULL,
  color_low = NULL,
  alpha = 0.4,
  ...
)
```

**Arguments**

| | |
|---|---|
| df | Dataframe likely containing two columns. |
| tooltip_data | Df of tooltip information for gvis graphs. |
| gvis_filename | Filename to write a fancy html graph. |
| cormethod | What type of correlation to check? |
| size | Size of the dots on the plot. |
| loess | Add a loess estimation? |
| identity | Add the identity line? |
| gvis_trendline | Add a trendline to the gvis plot? There are a couple possible types, I think linear is the most common. |
| z_lines | Include lines defining the z-score boundaries. |
| first | First column to plot. |
| second | Second column to plot. |
| base_url | Base url to add to the plot. |
| pretty_colors | Colors! |
| xlab | Alternate x-axis label. |
| ylab | Alternate x-axis label. |
| color_high | Chosen color for points significantly above the mean. |
| color_low | Chosen color for points significantly below the mean. |
| alpha | Choose an alpha channel to define how see-through the dots are. |
| ... | Extra args likely used for choosing significant genes. |

**Value**

List including a ggplot2 scatter plot and some histograms. This plot provides a "bird's eye" view
of two data sets. This plot assumes a (potential) linear correlation between the data, so it calculates
the correlation between them. It then calculates and plots a robust linear model of the data using an
'SMDM' estimator (which I don't remember how to describe, just that the document I was reading
said it is good). The median/mad of each axis is calculated and plotted as well. The distance from
the linear model is finally used to color the dots on the plot. Histograms of each axis are plotted
separately and then together under a single cdf to allow tests of distribution similarity. This will
make a fun clicky googleVis graph if requested.

**See Also**

**robust stats ggplot2** `lmRob weights plot_histogram`

**Examples**

```
## Not run:
 plot_linear_scatter(lotsofnumbers_intwo_columns, tooltip_data=tooltip_dataframe,
                     gvis_filename="html/fun_scatterplot.html")

## End(Not run)
```

---

plot_ma_de                *Make a pretty MA plot from one of limma, deseq, edger, or basic.*

---

**Description**

Because I can never remember, the following from wikipedia: "An MA plot is an application of a
Bland-Altman plot for visual representation of two channel DNA microarray gene expression data
which has been transformed onto the M (log ratios) and A (mean average) scale."

**Usage**

```
plot_ma_de(
  table,
  expr_col = "logCPM",
  fc_col = "logFC",
  p_col = "qvalue",
  p = 0.05,
  alpha = 0.4,
  logfc = 1,
  label_numbers = TRUE,
  size = 2,
  tooltip_data = NULL,
  gvis_filename = NULL,
  shapes = TRUE,
  invert = FALSE,
```

```
  label = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `table` | Df of linear-modelling, normalized counts by sample-type, |
| `expr_col` | Column showing the average expression across genes. |
| `fc_col` | Column showing the logFC for each gene. |
| `p_col` | Column containing the relevant p values. |
| `p` | Name of the pvalue column to use for cutoffs. |
| `alpha` | How transparent to make the dots. |
| `logfc` | Fold change cutoff. |
| `label_numbers` | Show how many genes were 'significant', 'up', and 'down'? |
| `size` | How big are the dots? |
| `tooltip_data` | Df of tooltip information for gvis. |
| `gvis_filename` | Filename to write a fancy html graph. |
| `shapes` | Provide different shapes for up/down/etc? |
| `invert` | Invert the ma plot? |
| `label` | Label the top/bottom n logFC values? |
| `...` | More options for you |

## Value

ggplot2 MA scatter plot. This is defined as the rowmeans of the normalized counts by type across all sample types on the x axis, and the log fold change between conditions on the y-axis. Dots are colored depending on if they are 'significant.' This will make a fun clicky googleVis graph if requested.

## See Also

**limma googleVis DESeq2 edgeR** `plot_gvis_ma` `toptable` voom `hpgl_voom` `lmFit` `makeContrasts` `contrasts.fit`

## Examples

```
 ## Not run:
  plot_ma(voomed_data, table, gvis_filename="html/fun_ma_plot.html")
  ## Currently this assumes that a variant of toptable was used which
  ## gives adjusted p-values.  This is not always the case and I should
  ## check for that, but I have not yet.

## End(Not run)
```

plot_multihistogram    *Make a pretty histogram of multiple datasets.*

### Description

If there are multiple data sets, it might be useful to plot them on a histogram together and look at the t.test results between distributions.

### Usage

```
plot_multihistogram(
  data,
  log = FALSE,
  binwidth = NULL,
  bins = NULL,
  colors = NULL
)
```

### Arguments

| | |
|---|---|
| data | Dataframe of lots of pretty numbers, this also accepts lists. |
| log | Plot the data on the log scale? |
| binwidth | Set a static bin width with an unknown # of bins? If neither of these are provided, then bins is set to 500, if both are provided, then bins wins. |
| bins | Set a static # of bins of an unknown width? |
| colors | Change the default colors of the densities? |

### Value

List of the ggplot histogram and some statistics describing the distributions.

### See Also

**ggplot2** `pairwise.t.test` `ddply`

### Examples

```
## Not run:
 kittytime = plot_multihistogram(df)

## End(Not run)
```

---

plot_multiplot                    *Make a grid of plots.*

---

### Description

Make a grid of plots.

### Usage

```
plot_multiplot(plots, file, cols = NULL, layout = NULL)
```

### Arguments

| | |
|---|---|
| plots | List of plots |
| file | File to write to |
| cols | Number of columns in the grid |
| layout | Set the layout specifically |

### Value

a multiplot!

---

plot_mzxml_boxplot             *Make a boxplot out of some of the various data available in the mzxml*
                               *data.*

---

### Description

There are a few data within the mzXML raw data files which are likely candidates for simple
summary via a boxplot/densityplot/whatever. For the moment I am just doing boxplots of a few of
them. Since my metadata extractor dumps a couple of tables, one must choose a desired table and
column from it to plot.

### Usage

```
plot_mzxml_boxplot(
  mzxml_data,
  table = "precursors",
  column = "precursorintensity",
  violin = FALSE,
  names = NULL,
  title = NULL,
  scale = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| mzxml_data | Provide a list of mzxml data, one element for each sample. |
| table | One of precursors or scans |
| column | One of the columns from the table; if 'scans' is chosen, then likely choices include: 'peakscount', 'basepeakmz', 'basepeakintensity'; if 'precursors' is chosen, then the only likely choice for the moment is 'precursorintensity'. |
| violin | Print the samples as violins rather than only box/whiskers? |
| names | Names for the x-axis of the plot. |
| title | Title the plot? |
| scale | Put the data on a specific scale? |
| ... | Further arguments, presumably for colors or some such. |

## Value

Boxplot describing the requested column of data in the set of mzXML files.

---

| | |
|---|---|
| plot_nonzero | *Make a ggplot graph of the number of non-zero genes by sample.* |

---

## Description

This puts the number of genes with > 0 hits on the y-axis and CPM on the x-axis. Made by Ramzi Temanni <temanni at umd dot edu>.

## Usage

```
plot_nonzero(
  data,
  design = NULL,
  colors = NULL,
  plot_labels = NULL,
  expt_names = NULL,
  label_chars = 10,
  plot_legend = FALSE,
  title = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Expt, expressionset, or dataframe. |
| design | Eesign matrix. |
| colors | Color scheme. |

| plot_labels | How do you want to label the graph? 'fancy' will use directlabels() to try to match the labels with the positions without overlapping anything else will just stick them on a 45' offset next to the graphed point. |
| expt_names | Column or character list of preferred sample names. |
| label_chars | How many characters for sample names before abbreviation. |
| plot_legend | Print a legend for this plot? |
| title | Add a title? |
| ... | rawr! |

## Value

a ggplot2 plot of the number of non-zero genes with respect to each library's CPM.

## See Also

**ggplot2** `geom_point geom_dl`

## Examples

```
## Not run:
 nonzero_plot <- plot_nonzero(expt=expt)

## End(Not run)
```

---

| plot_num_siggenes | *Given a DE table with fold changes and p-values, show how 'significant' changes with changing cutoffs.* |

---

## Description

Sometimes one might want to know how many genes are deemed significant while shifting the bars which define significant. This provides that metrics as a set of tables of numbers of significant up/down genes when p-value is held constant, as well as number when fold-change is held constant.

## Usage

```
plot_num_siggenes(
  table,
  methods = c("limma", "edger", "deseq", "ebseq"),
  bins = 100,
  constant_p = 0.05,
  constant_fc = 0
)
```

## Arguments

| | |
|---|---|
| `table` | DE table to examine. |
| `methods` | List of methods to use when plotting. |
| `bins` | Number of incremental changes in p-value/FC to examine. |
| `constant_p` | When plotting changing FC, where should the p-value be held? |
| `constant_fc` | When plotting changing p, where should the FC be held? |

## Value

Plots and dataframes describing the changing definition of 'significant.'

## See Also

**ggplot2**

## Examples

```
## Not run:
 pairwise_result <- all_pairwise(expt)
 crazy_sigplots <- plot_num_siggenes(pairwise_result)

## End(Not run)
```

---

| | |
|---|---|
| plot_ontpval | *Make a pvalue plot from a df of IDs, scores, and p-values.* |

---

## Description

This function seeks to make generating pretty pvalue plots as shown by clusterprofiler easier.

## Usage

```
plot_ontpval(
  df,
  ontology = "MF",
  fontsize = 14,
  numerator = NULL,
  denominator = NULL
)
```

## Arguments

| | |
|---|---|
| `df` | Some data from topgo/goseq/clusterprofiler. |
| `ontology` | Ontology to plot (MF,BP,CC). |
| `fontsize` | Fiddling with the font size may make some plots more readable. |
| `numerator` | Column used for printing a ratio of genes/category. |
| `denominator` | Column used for printing a ratio of genes/category. |

## Value

Ggplot2 plot of pvalues vs. ontology.

## See Also

**goseq ggplot2** `goseq`

---

| plot_pairwise_ma | *Plot all pairwise MA plots in an experiment.* |

---

## Description

Use affy's ma.plot() on every pair of columns in a data set to help diagnose problematic samples.

## Usage

```
plot_pairwise_ma(data, log = NULL, ...)
```

## Arguments

| | |
|---|---|
| data | Expt expressionset or data frame. |
| log | Is the data in log format? |
| ... | Options are good and passed to arglist(). |

## Value

List of affy::maplots

## See Also

**affy** `ma.plot`

## Examples

```
## Not run:
 ma_plots = plot_pairwise_ma(expt=some_expt)

## End(Not run)
```

---

plot_pca                    *Make a PCA plot describing the samples' clustering.*

---

### Description

Make a PCA plot describing the samples' clustering.

### Usage

```
plot_pca(
  data,
  design = NULL,
  plot_colors = NULL,
  plot_title = NULL,
  plot_size = 5,
  plot_alpha = NULL,
  plot_labels = NULL,
  size_column = NULL,
  pc_method = "fast_svd",
  x_pc = 1,
  y_pc = 2,
  num_pc = NULL,
  expt_names = NULL,
  label_chars = 10,
  ...
)
```

### Arguments

| | |
|---|---|
| data | an expt set of samples. |
| design | a design matrix and. |
| plot_colors | a color scheme. |
| plot_title | a title for the plot. |
| plot_size | size for the glyphs on the plot. |
| plot_alpha | Add an alpha channel to the dots? |
| plot_labels | add labels? Also, what type? FALSE, "default", or "fancy". |
| size_column | use an experimental factor to size the glyphs of the plot |
| pc_method | how to extract the components? (svd |
| x_pc | Component to put on the x axis. |
| y_pc | Component to put on the y axis. |
| num_pc | How many components to calculate, default to the number of rows in the meta-data. |
| expt_names | Column or character list of preferred sample names. |
| label_chars | Maximum number of characters before abbreviating sample names. |
| ... | Arguments passed through to the pca implementations and plotter. |

## Value

a list containing the following (this is currently wrong)

1. pca = the result of fast.svd()
2. plot = ggplot2 pca_plot describing the principle component analysis of the samples.
3. table = a table of the PCA plot data
4. res = a table of the PCA res data
5. variance = a table of the PCA plot variance

## See Also

**directlabels** `geom_dl` `plot_pcs`

## Examples

```
## Not run:
 pca_plot <- plot_pca(expt=expt)
 pca_plot

## End(Not run)
```

---

plot_pca_genes                   *Make a PC plot describing the gene' clustering.*

---

## Description

Make a PC plot describing the gene' clustering.

## Usage

```
plot_pca_genes(
  data,
  design = NULL,
  plot_colors = NULL,
  plot_title = NULL,
  plot_size = 2,
  plot_alpha = 0.4,
  plot_labels = FALSE,
  size_column = NULL,
  pc_method = "fast_svd",
  x_pc = 1,
  y_pc = 2,
  label_column = "description",
  num_pc = 2,
  expt_names = NULL,
  label_chars = 10,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | an expt set of samples. |
| `design` | a design matrix and. |
| `plot_colors` | a color scheme. |
| `plot_title` | a title for the plot. |
| `plot_size` | size for the glyphs on the plot. |
| `plot_alpha` | Add an alpha channel to the dots? |
| `plot_labels` | add labels? Also, what type? FALSE, "default", or "fancy". |
| `size_column` | use an experimental factor to size the glyphs of the plot |
| `pc_method` | how to extract the components? (svd |
| `x_pc` | Component to put on the x axis. |
| `y_pc` | Component to put on the y axis. |
| `label_column` | Which metadata column to use for labels. |
| `num_pc` | How many components to calculate, default to the number of rows in the metadata. |
| `expt_names` | Column or character list of preferred sample names. |
| `label_chars` | Maximum number of characters before abbreviating sample names. |
| `...` | Arguments passed through to the pca implementations and plotter. |

## Value

a list containing the following (this is currently wrong)

1. pca = the result of fast.svd()
2. plot = ggplot2 pca_plot describing the principle component analysis of the samples.
3. table = a table of the PCA plot data
4. res = a table of the PCA res data
5. variance = a table of the PCA plot variance

## See Also

**directlabels** `geom_dl` `plot_pcs`

## Examples

```
## Not run:
 pca_plot <- plot_pca(expt=expt)
 pca_plot

## End(Not run)
```

---

plot_pcfactor                    *make a dotplot of some categorised factors and a set of principle components.*

---

### Description

This should make a quick df of the factors and PCs and plot them.

### Usage

```
plot_pcfactor(pc_df, expt, exp_factor = "condition", component = "PC1")
```

### Arguments

| | |
|---|---|
| pc_df | Df of principle components. |
| expt | Expt containing counts, metadata, etc. |
| exp_factor | Experimental factor to compare against. |
| component | Which principal component to compare against? |

### Value

Plot of principle component vs factors in the data

### See Also

**ggplot2**

### Examples

```
## Not run:
 estimate_vs_pcs <- plot_pcfactor(pcs, times)

## End(Not run)
```

---

plot_pcload                      *Print a plot of the top-n most PC loaded genes.*

---

### Description

Sometimes it is nice to know what is happening with the genes which have the greatest effect on a given principal component. This function provides that.

### Usage

```
plot_pcload(expt, genes = 40, desired_pc = 1, which_scores = "high", ...)
```

## Arguments

| | |
|---|---|
| `expt` | Input expressionset. |
| `genes` | How many genes to observe? |
| `desired_pc` | Which component to examine? |
| `which_scores` | Perhaps one wishes to see the least-important genes, if so set this to low. |
| `...` | Extra arguments passed, currently to nothing. |

## Value

List containing an expressionset of the subset and a plot of their expression.

---

| | |
|---|---|
| plot_pcs | *Plot principle components and make them pretty.* |

---

## Description

All the various dimension reduction methods share some of their end-results in common. Most notably a table of putative components which may be plotted against one another so that one may stare at the screen and look for clustering among the samples/genes/whatever. This function attempts to make that process as simple and pretty as possible.

## Usage

```
plot_pcs(
  pca_data,
  first = "PC1",
  second = "PC2",
  variances = NULL,
  design = NULL,
  plot_title = TRUE,
  plot_labels = NULL,
  x_label = NULL,
  y_label = NULL,
  plot_size = 5,
  outlines = TRUE,
  plot_alpha = NULL,
  size_column = NULL,
  rug = TRUE,
  cis = c(0.95, 0.9),
  ...
)
```

## Arguments

| | |
|---|---|
| `pca_data` | Dataframe of principle components PC1 .. PCN with any other arbitrary information. |
| `first` | Principle component PCx to put on the x axis. |
| `second` | Principle component PCy to put on the y axis. |
| `variances` | List of the percent variance explained by each component. |
| `design` | Experimental design with condition batch factors. |
| `plot_title` | Title for the plot. |
| `plot_labels` | Parameter for the labels on the plot. |
| `x_label` | Label for the x-axis. |
| `y_label` | Label for the y-axis. |
| `plot_size` | Size of the dots on the plot |
| `outlines` | Add a black outline to the plotted shapes? |
| `plot_alpha` | Add an alpha channel to the dots? |
| `size_column` | Experimental factor to use for sizing the glyphs |
| `rug` | Include the rugs on the sides of the plot? |
| `cis` | What (if any) confidence intervals to include. |
| `...` | Extra arguments dropped into arglist |

## Value

gplot2 PCA plot

## See Also

**ggplot2** `geom_dl`

## Examples

```
## Not run:
 pca_plot = plot_pcs(pca_data, first="PC2", second="PC4", design=expt$design)

## End(Not run)
```

plot_pct_kept                    *Make a ggplot graph of the percentage/number of reads kept/removed.*

## Description

The function expt_exclude_genes() removes some portion of the original reads. This function will make it possible to see what is left.

## Usage

```
plot_pct_kept(
  data,
  row = "pct_kept",
  condition = NULL,
  colors = NULL,
  names = NULL,
  text = TRUE,
  title = NULL,
  yscale = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Dataframe of the material remaining, usually expt$summary_table |
| row | Row name to plot. |
| condition | vector of sample condition names. |
| colors | Color scheme if the data is not an expt. |
| names | Alternate names for the x-axis. |
| text | Add the numeric values inside the top of the bars of the plot? |
| title | Title for the plot. |
| yscale | Whether or not to log10 the y-axis. |
| ... | More parameters for your good time! |

## Value

a ggplot2 bar plot of every sample's size

## See Also

**ggplot2** geom_bar geom_text prettyNum scale_y_log10

## Examples

```
## Not run:
 kept_plot <- plot_pct_kept(expt_removed)
 kept_plot  ## ooo pretty bargraph

## End(Not run)
```

---

plot_peprophet_data      *Plot some data from the result of extract_peprophet_data()*

---

## Description

extract_peprophet_data() provides a ridiculously large data table of a comet result after processing by RefreshParser and xinteract/peptideProphet. This table has some 37-ish columns and I am not entirely certain which ones are useful as diagnostics of the data. I chose a few and made options to pull some/most of the rest. Lets play!

## Usage

```
plot_peprophet_data(
  table,
  xaxis = "precursor_neutral_mass",
  xscale = NULL,
  yaxis = "num_matched_ions",
  yscale = NULL,
  size_column = "prophet_probability",
  ...
)
```

## Arguments

| | |
|---|---|
| table | Big honking data table from extract_peprophet_data() |
| xaxis | Column to plot on the x-axis |
| xscale | Change the scale of the x-axis? |
| yaxis | guess! |
| yscale | Change the scale of the y-axis? |
| size_column | Use a column for scaling the sizes of dots in the plot? |
| ... | extra options which may be used for plotting. |

## Value

a plot!

---

plot_pyprophet_counts  *Count some aspect(s) of the pyprophet data and plot them.*

---

### Description

This function is mostly redundant with the plot_mzxml_boxplot above. Unfortunately, the two data types are subtly different enough that I felt it not worth while to generalize the functions.

### Usage

```
plot_pyprophet_counts(
  pyprophet_data,
  type = "count",
  keep_real = TRUE,
  keep_decoys = TRUE,
  expt_names = NULL,
  label_chars = 10,
  title = NULL,
  scale = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| pyprophet_data | List containing the pyprophet results. |
| type | What to count/plot? |
| keep_real | Do we keep the real data when plotting the data? (perhaps we only want the decoys) |
| keep_decoys | Do we keep the decoys when plotting the data? |
| expt_names | Names for the x-axis of the plot. |
| label_chars | Maximum number of characters before abbreviating sample names. |
| title | Title the plot? |
| scale | Put the data on a specific scale? |
| ... | Further arguments, presumably for colors or some such. |

### Value

Boxplot describing the desired column from the data.

```
plot_pyprophet_distribution
```
                              *Make a boxplot out of some of the various data available in the*
                              *pyprophet data.*

## Description

This function is mostly redundant with the plot_mzxml_boxplot above. Unfortunately, the two data
types are subtly different enough that I felt it not worth while to generalize the functions.

## Usage

```
plot_pyprophet_distribution(
  pyprophet_data,
  column = "delta_rt",
  keep_real = TRUE,
  keep_decoys = TRUE,
  expt_names = NULL,
  label_chars = 10,
  title = NULL,
  scale = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| pyprophet_data | List containing the pyprophet results. |
| column | What column of the pyprophet scored data to plot? |
| keep_real | Do we keep the real data when plotting the data? (perhaps we only want the decoys) |
| keep_decoys | Do we keep the decoys when plotting the data? |
| expt_names | Names for the x-axis of the plot. |
| label_chars | Maximum number of characters before abbreviating sample names. |
| title | Title the plot? |
| scale | Put the data on a specific scale? |
| ... | Further arguments, presumably for colors or some such. |

## Value

Boxplot describing the desired column from the data.

---

plot_pyprophet_points *Plot some data from the result of extract_pyprophet_data()*

---

## Description

extract_pyprophet_data() provides a ridiculously large data table of a scored openswath data after processing by pyprophet.

## Usage

```
plot_pyprophet_points(
  pyprophet_data,
  xaxis = "mass",
  xscale = NULL,
  sample = NULL,
  yaxis = "leftwidth",
  yscale = NULL,
  alpha = 0.4,
  color_by = "sample",
  legend = TRUE,
  size_column = "mscore",
  rug = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| pyprophet_data | List of pyprophet data, one element for each sample, taken from extract_peprophet_data() |
| xaxis | Column to plot on the x-axis |
| xscale | Change the scale of the x-axis? |
| sample | Which sample(s) to include? |
| yaxis | guess! |
| yscale | Change the scale of the y-axis? |
| alpha | How see-through to make the dots? |
| color_by | Change the colors of the points either by sample or condition? |
| legend | Include a legend of samples? |
| size_column | Use a column for scaling the sizes of dots in the plot? |
| rug | Add a distribution rug to the axes? |
| ... | extra options which may be used for plotting. |

## Value

a plot!

---

plot_pyprophet_protein

*Read data from pyprophet and plot columns from it.*

---

## Description

More proteomics diagnostics! Now that I am looking more closely, I think this should be folded into plot_pyprophet_distribution().

## Usage

```
plot_pyprophet_protein(
  pyprophet_data,
  column = "intensity",
  keep_real = TRUE,
  keep_decoys = FALSE,
  expt_names = NULL,
  label_chars = 10,
  protein = NULL,
  title = NULL,
  scale = NULL,
  legend = NULL,
  order_by = "condition",
  show_all = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| pyprophet_data | Data from extract_pyprophet_data() |
| column | Chosen column to plot. |
| keep_real | FIXME: This should be changed to something like 'data_type' here and in plot_pyprophet_distribution. |
| keep_decoys | Do we keep the decoys when plotting the data? |
| expt_names | Names for the x-axis of the plot. |
| label_chars | Maximum number of characters before abbreviating sample names. |
| protein | chosen protein(s) to plot. |
| title | Title the plot? |
| scale | Put the data on a specific scale? |
| legend | Include the legend? |
| order_by | Reorder the samples by some factor, presumably condition. |
| show_all | Skip samples for which no observations were made. |
| ... | Further arguments, presumably for colors or some such. |

**Value**

Boxplot describing the desired column from the data.

---

| plot_pyprophet_xy | *Invoked plot_pyprophet_counts() twice, once for the x-axis, and once for the y.* |
|---|---|

---

**Description**

Then plot the result, hopefully adding some new insights into the state of the post-pyprophet results. By default, this puts the number of identifications (number of rows) on the x-axis for each sample, and the sum of intensities on the y. Currently missing is the ability to change this from sum to mean/median/etc. That should trivially be possible via the addition of arguments for the various functions of interest.

**Usage**

```
plot_pyprophet_xy(
  pyprophet_data,
  keep_real = TRUE,
  size = 6,
  label_size = 4,
  keep_decoys = TRUE,
  expt_names = NULL,
  label_chars = 10,
  x_type = "count",
  y_type = "intensity",
  title = NULL,
  scale = NULL,
  ...
)
```

**Arguments**

| | |
|---|---|
| pyprophet_data | List of pyprophet matrices by sample. |
| keep_real | Use the real identifications (as opposed to the decoys)? |
| size | Size of the glyphs used in the plot. |
| label_size | Set the label sizes. |
| keep_decoys | Use the decoy identifications (vs. the real)? |
| expt_names | Manually change the labels to some other column than sample. |
| label_chars | Maximum number of characters in the label before shortening. |
| x_type | Column in the data to put on the x-axis. |
| y_type | Column in the data to put on the y-axis. |
| title | Plot title. |

| scale | Put the data onto the log scale? |
|---|---|
| ... | Extra arguments passed along. |

---

| plot_qq_all | *Quantile/quantile comparison of the mean of all samples vs. each sample.* |
|---|---|

---

## Description

This allows one to visualize all individual data columns against the mean of all columns of data in order to see if any one is significantly different than the cloud.

## Usage

```
plot_qq_all(data, labels = "short", ...)
```

## Arguments

| data | Expressionset, expt, or dataframe of samples. |
|---|---|
| labels | What kind of labels to print? |
| ... | Arguments passed presumably from graph_metrics(). |

## Value

List containing: logs = a recordPlot() of the pairwise log qq plots. ratios = a recordPlot() of the pairwise ratio qq plots. means = a table of the median values of all the summaries of the qq plots.

## See Also

**Biobase**

---

| plot_rmats | *Given some psi and tpm data from rMATS, make a pretty plot!* |
|---|---|

---

## Description

This should take either a dataframe or filename for the psi data from rMATS. This was mostly copy/pasted from plot_suppa().

## Usage

```
plot_rmats(
  se = NULL,
  a5ss = NULL,
  a3ss = NULL,
  mxe = NULL,
  ri = NULL,
  sig_threshold = 0.05,
  dpsi_threshold = 0.7,
  label_type = NULL,
  alpha = 0.7
)
```

## Arguments

| | |
|---|---|
| se | Table of skipped exon data from rmats. |
| a5ss | Table of alternate 5p exons. |
| a3ss | Table of alternate 3p exons. |
| mxe | Table of alternate exons. |
| ri | Table of retained introns. |
| sig_threshold | Use this significance threshold. |
| dpsi_threshold | Use a delta threshold. |
| label_type | Choose a type of event to label. |
| alpha | How see-through should the points be in the plot? |

## Value

List containing the plot and some of the requisite data.

## See Also

[plot_suppa](#)

## Examples

```
 ## Not run:
 rmats_plot <- plot_rmats(se_table, a5_table, a3_table)

## End(Not run)
```

| plot_rpm | *Make relatively pretty bar plots of coverage in a genome.* |
|---|---|

### Description

This was written for ribosome profiling coverage / gene. It should however, work for any data with little or no modification, it was also written when I was first learning R and when I look at it now I see a few obvious places which can use improvement.

### Usage

```
plot_rpm(
  input,
  workdir = "images",
  output = "01.svg",
  name = "LmjF.01.0010",
  start = 1000,
  end = 2000,
  strand = 1,
  padding = 100
)
```

### Arguments

| | |
|---|---|
| input | Coverage / position filename. |
| workdir | Where to put the resulting images. |
| output | Output image filename. |
| name | Gene name to print at the bottom of the plot. |
| start | Relative to 0, where is the gene's start codon. |
| end | Relative to 0, where is the gene's stop codon. |
| strand | Is this on the + or - strand? (+1/-1) |
| padding | How much space to provide on the sides? |

### Value

coverage plot surrounging the ORF of interest

### See Also

**ggplot2**

plot_sample_cvheatmap  *An experiment to see if I can visualize the genes with the highest variance.*

### Description

An experiment to see if I can visualize the genes with the highest variance.

### Usage

```
plot_sample_cvheatmap(
  expt,
  fun = "mean",
  fact = "condition",
  row_label = NA,
  title = NULL,
  Rowv = TRUE,
  Colv = TRUE,
  label_chars = 10,
  dendrogram = "column",
  min_delta = 0.5,
  x_factor = 1,
  y_factor = 2,
  min_cvsd = NULL,
  cv_min = 1,
  cv_max = Inf,
  remove_equal = TRUE
)
```

### Arguments

| | |
|---|---|
| expt | ExpressionSet |
| fun | mean or median |
| fact | Which factor to slice/dice the data? |
| row_label | Label the rows? |
| title | Title for the plot |
| Rowv | Row vs (yeah I forgot what this does.) |
| Colv | Col vs |
| label_chars | Maximum number of characters in the sample IDs. |
| dendrogram | Make a tree of the samples? |
| min_delta | Minimum delta value for filtering |
| x_factor | When plotting two factors against each other, which is x? |
| y_factor | When plotting two factors against each other, which is y? |

| | |
|---|---|
| cv_min | Minimum cv to examine (I think this should be slightly lower) |
| cv_max | Maximum cV to examine (I think this should be limited to ~ 0.7?) |
| remove_equal | Filter uninteresting genes. |

---

| | |
|---|---|
| plot_sample_heatmap | *Make a heatmap.3 description of the similarity of the genes among samples.* |

---

## Description

Sometimes you just want to see how the genes of an experiment are related to each other. This can handle that. These heatmap functions should probably be replaced with neatmaps or heatplus or whatever it is, as the annotation dataframes in them are pretty awesome.

## Usage

```
plot_sample_heatmap(
  data,
  colors = NULL,
  design = NULL,
  expt_names = NULL,
  dendrogram = "column",
  row_label = NA,
  title = NULL,
  Rowv = TRUE,
  Colv = TRUE,
  label_chars = 10,
  filter = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Expt/expressionset/dataframe set of samples. |
| colors | Color scheme of the samples (not needed if input is an expt). |
| design | Design matrix describing the experiment (gotten for free if an expt). |
| expt_names | Alternate samples names. |
| dendrogram | Where to put dendrograms? |
| row_label | Passed through to heatmap.2. |
| title | Title of the plot! |
| Rowv | Reorder the rows by expression? |
| Colv | Reorder the columns by expression? |
| label_chars | Maximum number of characters before abbreviating sample names. |
| filter | Filter the data before performing this plot? |
| ... | More parameters for a good time! |

## Value

a recordPlot() heatmap describing the samples.

## See Also

**RColorBrewer** `brewer.pal recordPlot`

---

| plot_scatter | *Make a pretty scatter plot between two sets of numbers.* |

---

## Description

This function tries to supplement a normal scatterplot with some information describing the relationship between the columns of data plotted.

## Usage

```
plot_scatter(
  df,
  tooltip_data = NULL,
  color = "black",
  xlab = NULL,
  ylab = NULL,
  alpha = 0.6,
  gvis_filename = NULL,
  size = 2
)
```

## Arguments

| | |
|---|---|
| df | Dataframe likely containing two columns. |
| tooltip_data | Df of tooltip information for gvis. |
| color | Color of the dots on the graph. |
| xlab | Alternate x-axis label. |
| ylab | Alternate x-axis label. |
| alpha | Define how see-through the dots are. |
| gvis_filename | Filename to write a fancy html graph. |
| size | Size of the dots on the graph. |

## Value

Ggplot2 scatter plot.

## See Also

**ggplot2 googleVis** `plot_gvis_scatter geom_point plot_linear_scatter`

## Examples

```
## Not run:
 plot_scatter(lotsofnumbers_intwo_columns, tooltip_data=tooltip_dataframe,
              gvis_filename="html/fun_scatterplot.html")

## End(Not run)
```

---

plot_significant_bar     *Make a bar plot of the numbers of significant genes by contrast. These*
                         *plots are quite difficult to describe.*

---

## Description

Make a bar plot of the numbers of significant genes by contrast. These plots are quite difficult to describe.

## Usage

```
plot_significant_bar(
  ups,
  downs,
  maximum = NULL,
  text = TRUE,
  color_list = c("lightcyan", "lightskyblue", "dodgerblue", "plum1", "orchid",
    "purple4"),
  color_names = c("a_up_inner", "b_up_middle", "c_up_outer", "a_down_inner",
    "b_down_middle", "c_down_outer")
)
```

## Arguments

| | |
|---|---|
| ups | Set of up-regulated genes. |
| downs | Set of down-regulated genes. |
| maximum | Maximum/minimum number of genes to display. |
| text | Add text at the ends of the bars describing the number of genes >/< 0 fc. |
| color_list | Set of colors to use for the bars. |
| color_names | Categories associated with aforementioned colors. |

## Value

weird significance bar plots

## See Also

**ggplot2** extract_significant_genes

---

| plot_single_qq | *Perform a qqplot between two columns of a matrix.* |

---

## Description

Given two columns of data, how well do the distributions match one another? The answer to that question may be visualized through a qq plot!

## Usage

```
plot_single_qq(data, x = 1, y = 2, labels = TRUE)
```

## Arguments

| | |
|---|---|
| data | Data frame/expt/expressionset. |
| x | First column to compare. |
| y | Second column to compare. |
| labels | Include the lables? |

## Value

a list of the logs, ratios, and mean between the plots as ggplots.

## See Also

**Biobase**

---

| plot_sm | *Make an R plot of the standard median correlation or distance among samples.* |

---

## Description

This was written by a mix of Kwame Okrah <kokrah at gmail dot com>, Laura Dillon <dillonl at umd dot edu>, and Hector Corrada Bravo <hcorrada at umd dot edu> I reimplemented it using ggplot2 and tried to make it a little more flexible. The general idea is to take the pairwise correlations/distances of the samples, then take the medians, and plot them. This version of the plot is no longer actually a dotplot, but a point plot, but who is counting?

## Usage

```
plot_sm(
  data,
  colors = NULL,
  method = "pearson",
  plot_legend = FALSE,
  expt_names = NULL,
  label_chars = 10,
  title = NULL,
  dot_size = 5,
  ...
)
```

## Arguments

| | |
|---|---|
| data | Expt, expressionset, or data frame. |
| colors | Color scheme if data is not an expt. |
| method | Correlation or distance method to use. |
| plot_legend | Include a legend on the side? |
| expt_names | Use pretty names for the samples? |
| label_chars | Maximum number of characters before abbreviating sample names. |
| title | Title for the graph. |
| dot_size | How large should the glyphs be? |
| ... | More parameters to make you happy! |

## Value

ggplot of the standard median something among the samples. This will also write to an open device. The resulting plot measures the median correlation of each sample among its peers. It notes 1.5* the interquartile range among the samples and makes a horizontal line at that correlation coefficient. Any sample which falls below this line is considered for removal because it is much less similar to all of its peers.

## See Also

**matrixStats grDevices** `hpgl_cor` `rowMedians` `quantile` `diff` `recordPlot`

## Examples

```
## Not run:
 smc_plot = hpgl_smc(expt=expt)

## End(Not run)
```

---

plot_spirograph                *Make spirographs!*

---

## Description

Taken (with modifications) from: http://menugget.blogspot.com/2012/12/spirograph-with-r.html#more
A positive value for 'B' will result in a epitrochoid, while a negative value will result in a hypotrochoid.

## Usage

```
plot_spirograph(
  radius_a = 1,
  radius_b = -4,
  dist_bc = -2,
  revolutions = 158,
  increments = 3160,
  center_a = list(x = 0, y = 0)
)
```

## Arguments

| | |
|---|---|
| radius_a | The radius of the primary circle. |
| radius_b | The radius of the circle travelling around a. |
| dist_bc | A point relative to the center of 'b' which rotates with the turning of 'b'. |
| revolutions | How many revolutions to perform in the plot |
| increments | The number of radial increments to be calculated per revolution |
| center_a | The position of the center of 'a'. |

## Value

something which I don't yet know.

---

plot_suppa                *Given some psi and tpm data, make a pretty plot!*

---

## Description

This should take either a dataframe or filename for the psi data from suppa, along with the same for the average log tpm data (acquired from suppa diffSplice with –save_tpm_events)

## Usage

```
plot_suppa(
  dpsi,
  tpm,
  events = NULL,
  psi = NULL,
  sig_threshold = 0.05,
  label_type = NULL,
  alpha = 0.7
)
```

## Arguments

| | |
|---|---|
| dpsi | Table provided by suppa containing all the metrics. |
| tpm | Table provided by suppa containing all the tpm values. |
| events | List of event types to include. |
| psi | Limit the set of included events by psi value? |
| sig_threshold | Use this significance threshold. |
| label_type | Choose a type of event to label. |
| alpha | How see-through should the points be in the plot? |

## Value

List containing the plot and some of the requisite data.

## See Also

[plot_rmats](plot_rmats)

## Examples

```
  ## Not run:
  suppa_plot <- plot_suppa(dpsi_file, tmp_file)

  ## End(Not run)
```

---

| plot_svfactor | *Make a dotplot of some categorised factors and a set of SVs (for other factors).* |
|---|---|

---

## Description

This should make a quick df of the factors and surrogates and plot them.

## Usage

```
plot_svfactor(
  expt,
  svest,
  sv = 1,
  chosen_factor = "batch",
  factor_type = "factor"
)
```

## Arguments

| | |
|---|---|
| expt | Experiment from which to acquire the design, counts, etc. |
| svest | Set of surrogate variable estimations from sva/svg or batch estimates. |
| sv | Which surrogate to plot? |
| chosen_factor | Factor to compare against. |
| factor_type | This may be a factor or range, it is intended to plot a scatterplot if it is a range, a dotplot if a factor. |

## Value

surrogate variable plot as per Leek's work

## See Also

**ggplot2**

## Examples

```
## Not run:
 estimate_vs_snps <- plot_svfactor(start, surrogate_estimate, "snpcategory")

## End(Not run)
```

---

plot_topgo_densities    *Plot the density of categories vs. the possibilities of all categories.*

---

## Description

This can make a large number of plots.

## Usage

```
plot_topgo_densities(godata, table)
```

## Arguments

| | |
|---|---|
| godata | Result from topgo. |
| table | Table of genes. |

## Value

density plot as per topgo

## See Also

**topGO**

---

plot_topgo_pval            *Make a pvalue plot from topgo data.*

---

## Description

The p-value plots from clusterProfiler are pretty, this sets the topgo data into a format suitable for plotting in that fashion and returns the resulting plots of significant ontologies.

## Usage

```
plot_topgo_pval(
  topgo,
  wrapped_width = 20,
  cutoff = 0.1,
  n = 30,
  type = "fisher",
  ...
)
```

## Arguments

| | |
|---|---|
| topgo | Some data from topgo! |
| wrapped_width | Maximum width of the text names. |
| cutoff | P-value cutoff for the plots. |
| n | Maximum number of ontologies to include. |
| type | Type of score to use. |
| ... | arguments passed through presumably from simple_topgo() |

## Value

List of MF/BP/CC pvalue plots.

## See Also

**topgo clusterProfiler**

---

plot_topn    *Plot the representation of the top-n genes in the total counts / sample.*

---

**Description**

One question we might ask is: how much do the most abundant genes in a samples comprise the entire sample? This plot attempts to provide a visual hint toward answering this question. It does so by rank-ordering all the genes in every sample and dividing their counts by the total number of reads in that sample. It then smooths the points to provide the resulting trend. The steeper the resulting line, the more over-represented these top-n genes are. I suspect, but haven't tried yet, that the inflection point of the resulting curve is also a useful diagnostic in this question.

**Usage**

```
plot_topn(
  data,
  title = NULL,
  num = 100,
  expt_names = NULL,
  plot_labels = "direct",
  label_chars = 10,
  plot_legend = FALSE,
  ...
)
```

**Arguments**

| | |
|---|---|
| data | Dataframe/matrix/whatever for performing topn-plot. |
| title | A title for the plot. |
| num | The N in top-n genes, if null, do them all. |
| expt_names | Column or character list of sample names. |
| plot_labels | Method for labelling the lines. |
| label_chars | Maximum number of characters before abbreviating samples. |
| plot_legend | Add a legend to the plot? |
| ... | Extra arguments, currently unused. |

**Value**

List containing the ggplot2

---

plot_tsne                    *Shortcut to plot_pca(pc_method="tsne")*

---

### Description

Shortcut to plot_pca(pc_method="tsne")

### Usage

```
plot_tsne(...)
```

### Arguments

| | |
|---|---|
| ... | Arguments for plot_pca() |

---

plot_variance_coefficients

*Look at the (biological)coefficient of variation/quartile coefficient of dispersion with respect to an experimental factor.*

---

### Description

I want to look at the (B)CV of some data with respect to condition/batch/whatever. This function should make that possible, with some important caveats. The most appropriate metric is actually the biological coefficient of variation as calculated by DESeq2/EdgeR; but the metrics I am currently taking are the simpler and less appropriate CV(sd/mean) and QCD(q3-q1/q3+q1).

### Usage

```
plot_variance_coefficients(
  data,
  x_axis = "condition",
  colors = NULL,
  title = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| data | Expressionset/epxt to poke at. |
| x_axis | Factor in the experimental design we may use to group the data and calculate the dispersion metrics. |
| colors | Set of colors to use when making the violins |
| title | Optional title to include with the plot. |
| ... | Extra arguments to pass along. |

## Value

List of plots showing the coefficients vs. genes along with the data.

---

plot_volcano_de    *Make a pretty Volcano plot!*

---

## Description

Volcano plots and MA plots provide quick an easy methods to view the set of (in)significantly differentially expressed genes. In the case of a volcano plot, it places the -log10 of the p-value estimate on the y-axis and the fold-change between conditions on the x-axis. Here is a neat snippet from wikipedia: "The concept of volcano plot can be generalized to other applications, where the x-axis is related to a measure of the strength of a statistical signal, and y-axis is related to a measure of the statistical significance of the signal."

## Usage

```
plot_volcano_de(
  table,
  alpha = 0.6,
  color_by = "p",
  color_list = c(`FALSE` = "darkred", `TRUE` = "darkblue"),
  fc_col = "logFC",
  fc_name = "log2 fold change",
  gvis_filename = NULL,
  line_color = "black",
  line_position = "bottom",
  logfc = 1,
  p_col = "adj.P.Val",
  p_name = "-log10 p-value",
  p = 0.05,
  shapes_by_state = TRUE,
  size = 2,
  tooltip_data = NULL,
  label = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| table | Dataframe from limma's toptable which includes log(fold change) and an adjusted p-value. |
| alpha | How transparent to make the dots. |
| color_by | By p-value something else? |
| color_list | List of colors for significance. |

| | |
|---|---|
| `fc_col` | Which column contains the fc data? |
| `fc_name` | Name of the fold-change to put on the plot. |
| `gvis_filename` | Filename to write a fancy html graph. |
| `line_color` | What color for the significance lines? |
| `line_position` | Put the significance lines above or below the dots? |
| `logfc` | Cutoff defining the minimum/maximum fold change for interesting. |
| `p_col` | Which column contains the p-value data? |
| `p_name` | Name of the p-value to put on the plot. |
| `p` | Cutoff defining significant from not. |
| `shapes_by_state` | |
| | Add fun shapes for the various significance states? |
| `size` | How big are the dots? |
| `tooltip_data` | Df of tooltip information for gvis. |
| `label` | Label the top/bottom n logFC values? |
| `...` | I love parameters! |

### Value

Ggplot2 volcano scatter plot. This is defined as the -log10(p-value) with respect to log(fold change). The cutoff values are delineated with lines and mark the boundaries between 'significant' and not. This will make a fun clicky googleVis graph if requested.

### See Also

**limma** `plot_gvis_ma` `toptable` `voom` `hpgl_voom` `lmFit` `makeContrasts` `contrasts.fit`

### Examples

```
## Not run:
 plot_volcano_de(table, gvis_filename="html/fun_ma_plot.html")
 ## Currently this assumes that a variant of toptable was used which
 ## gives adjusted p-values.  This is not always the case and I should
 ## check for that, but I have not yet.

## End(Not run)
```

plotly_pca                 *Plot a PC plot with options suitable for ggplotly.*

## Description

Plot a PC plot with options suitable for ggplotly.

## Usage

```
plotly_pca(
  data,
  design = NULL,
  plot_colors = NULL,
  plot_title = NULL,
  plot_size = 5,
  plot_alpha = NULL,
  plot_labels = NULL,
  size_column = NULL,
  pc_method = "fast_svd",
  x_pc = 1,
  y_pc = 2,
  outlines = FALSE,
  num_pc = NULL,
  expt_names = NULL,
  label_chars = 10,
  tooltip = c("shape", "fill", "sampleid"),
  ...
)
```

## Arguments

| | |
|---|---|
| data | an expt set of samples. |
| design | a design matrix and. |
| plot_colors | a color scheme. |
| plot_title | a title for the plot. |
| plot_size | size for the glyphs on the plot. |
| plot_alpha | Add an alpha channel to the dots? |
| plot_labels | add labels? Also, what type? FALSE, "default", or "fancy". |
| size_column | use an experimental factor to size the glyphs of the plot |
| pc_method | how to extract the components? (svd |
| x_pc | Component to put on the x axis. |
| y_pc | Component to put on the y axis. |
| outlines | Include black outlines around glyphs? |

| num_pc | How many components to calculate, default to the number of rows in the metadata. |
| expt_names | Column or character list of preferred sample names. |
| label_chars | Maximum number of characters before abbreviating sample names. |
| tooltip | Which columns to include in the tooltip. |
| ... | Arguments passed through to the pca implementations and plotter. |

## Value

This passes directly to plot_pca(), so its returns should be applicable along with the result from ggplotly.

---

pp                               *Plot a picture, with hopefully useful options for most(any) format.*

---

## Description

This calls svg/png/postscript/etc according to the filename provided.

## Usage

```
pp(file, image = NULL, width = 9, height = 9, res = 180, ...)
```

## Arguments

| file | Filename to write |
| image | Optionally, add the image you wish to plot and this will both print it to file and screen. |
| width | How wide? |
| height | How high? |
| res | The chosen resolution. |
| ... | Arguments passed to the image plotters. |

## Value

a png/svg/eps/ps/pdf with height=width=9 inches and a high resolution

---

print_ups_downs *Reprint the output from extract_significant_genes().*

---

## Description

I found myself needing to reprint these excel sheets because I added some new information. This shortcuts that process for me.

## Usage

```
print_ups_downs(
  upsdowns,
  wb = NULL,
  excel = "excel/significant_genes.xlsx",
  according = "limma",
  summary_count = 1,
  ma = FALSE
)
```

## Arguments

| | |
|---|---|
| upsdowns | Output from extract_significant_genes(). |
| wb | Workbook object to use for writing, or start a new one. |
| excel | Filename for writing the data. |
| according | Use limma, deseq, or edger for defining 'significant'. |
| summary_count | For spacing sequential tables one after another. |
| ma | Include ma plots? |

## Value

Return from write_xlsx.

## See Also

[combine_de_tables](#)

270 rank_order_scatter

---

random_ontology          *Perform a simple_ontology() on some random data.*

---

### Description

At the very least, the result should be less significant than the actual data!

### Usage

```
random_ontology(input, method = "goseq", n = 200, ...)
```

### Arguments

| | |
|---|---|
| input | Some input data |
| method | goseq, clusterp, topgo, gostats, gprofiler. |
| n | how many 'genes' to analyse? |
| ... | Arguments passed to the method. |

### Value

An ontology result

---

rank_order_scatter          *Plot the rank order of the data in two tables against each other.*

---

### Description

Steve Christensen has some neat plots showing the relationship between two tables. I thought they were cool, so I co-opted the idea in this function.

### Usage

```
rank_order_scatter(
  first,
  second = NULL,
  first_type = "limma",
  second_type = "limma",
  first_table = NULL,
  alpha = 0.5,
  second_table = NULL,
  first_column = "logFC",
  second_column = "logFC",
  first_p_col = "adj.P.Val",
  second_p_col = "adj.P.Val",
  p_limit = 0.05,
```

```
   both_color = "red",
   first_color = "green",
   second_color = "blue",
   no_color = "black"
)
```

## Arguments

| | |
|---|---|
| `first` | First table of values. |
| `second` | Second table of values, if null it will use the first. |
| `first_type` | Assuming this is from all_pairwise(), use this method. |
| `second_type` | Ibid. |
| `first_table` | Again, assuming all_pairwise(), use this to choose the table to extract. |
| `alpha` | How see-through to make the dots? |
| `second_table` | Ibid. |
| `first_column` | What column to use to rank-order from the first table? |
| `second_column` | What column to use to rank-order from the second table? |
| `first_p_col` | Use this column for pretty colors from the first table. |
| `second_p_col` | Use this column for pretty colors from the second table. |
| `p_limit` | A p-value limit for coloring dots. |
| `both_color` | If both columns are 'significant', use this color. |
| `first_color` | If only the first column is 'significant', this color. |
| `second_color` | If the second column is 'significant', this color. |
| `no_color` | If neither column is 'significant', then this color. |

## Value

a list with a plot and a couple summary statistics.

---

| read_counts_expt | *Read a bunch of count tables and create a usable data frame from them.* |
|---|---|

---

## Description

It is worth noting that this function has some logic intended for the elsayed lab's data storage structure. It shouldn't interfere with other usages, but it attempts to take into account different ways the data might be stored.

## Usage

```
read_counts_expt(
  ids,
  files,
  header = FALSE,
  include_summary_rows = FALSE,
  suffix = NULL,
  countdir = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| ids | List of experimental ids. |
| files | List of files to read. |
| header | Whether or not the count tables include a header row. |
| include_summary_rows | |
| | Whether HTSeq summary rows should be included. |
| suffix | Optional suffix to add to the filenames when reading them. |
| countdir | Optional count directory to read from. |
| ... | More options for happy time! |

## Details

Used primarily in create_expt() This is responsible for reading count tables given a list of filenames.
It tries to take into account upper/lowercase filenames and uses data.table to speed things along.

## Value

Data frame of count tables.

## See Also

**data.table** [create_expt](#)

## Examples

```
## Not run:
 count_tables <- hpgl_read_files(as.character(sample_ids), as.character(count_filenames))

## End(Not run)
```

| | |
|---|---|
| read_metadata | *Given a table of meta data, read it in for use by create_expt().* |

## Description

Reads an experimental design in a few different formats in preparation for creating an expt.

## Usage

```
read_metadata(file, ...)
```

## Arguments

| | |
|---|---|
| file | Csv/xls file to read. |
| ... | Arguments for arglist, used by sep, header and similar read_csv/read.table parameters. |

## Value

Df of metadata.

## See Also

**tools openxlsx XLConnect**

| | |
|---|---|
| read_snp_columns | *Read the output from bcfutils into a count-table-esque* |

## Description

Previously, I put all my bcfutils output files into one directory. This function would iterate through every file in that directory and add the contents as columns to this growing data table. Now it works by accepting a list of filenames (presumably kept in the metadata for the experiment) and reading them into the data table. It is worth noting that it can accept either a column name or index – which when you think about it is pretty much always true, but in this context is particularly interesting since I changed the names of all the columns when I rewrote this functionality.

## Usage

```
read_snp_columns(samples, file_lst, column = "diff_count")
```

## Arguments

| | |
|---|---|
| samples | Sample names to read. |
| file_lst | Set of files to read. |
| column | Column from the bcf file to read. |

**Value**

A big honking data table.

---

read_thermo_xlsx          *Parse the difficult thermo fisher xlsx file.*

---

**Description**

The Thermo(TM) workflow has as its default a fascinatingly horrible excel output. This function parses that into a series of data frames.

**Usage**

```
read_thermo_xlsx(xlsx_file, test_row = NULL)
```

**Arguments**

| | |
|---|---|
| xlsx_file | The input xlsx file |
| test_row | A single row in the xlsx file to use for testing, as I have not yet seen two of these accursed files which had the same headers. |

**Value**

List containing the protein names, group data, protein dataframe, and peptide dataframe.

---

recolor_points          *Quick point-recolorizer given an existing plot, df, list of rownames to recolor, and a color.*

---

**Description**

This function should make it easy to color a family of genes in any of the point plots.

**Usage**

```
recolor_points(plot, df, ids, color = "red", ...)
```

**Arguments**

| | |
|---|---|
| plot | Geom_point based plot |
| df | Data frame used to create the plot |
| ids | Set of ids which must be in the rownames of df to recolor |
| color | Chosen color for the new points. |
| ... | Extra arguments are passed to arglist. |

## Value

prettier plot.

---

| renderme | *Add a little logic to rmarkdown::render to date the final outputs as per a request from Najib.* |
|---|---|

---

## Description

Add a little logic to rmarkdown::render to date the final outputs as per a request from Najib.

## Usage

```
renderme(file, format = "html_document")
```

## Arguments

| | |
|---|---|
| file | Rmd file to render. |
| format | Chosen file format. |

## Value

Final filename including the prefix rundate.

---

| replot_varpart_percent | |
|---|---|
| | *A shortcut for replotting the percent plots from variancePartition.* |

---

## Description

In case I wish to look at different numbers of genes from variancePartition and/or different columns to sort from.

## Usage

```
replot_varpart_percent(
  varpart_output,
  n = 30,
  column = NULL,
  decreasing = TRUE
)
```

## Arguments

| | |
|---|---|
| `varpart_output` | List returned by varpart() |
| `n` | How many genes to plot. |
| `column` | The df column to use for sorting. |
| `decreasing` | high->low or vice versa? |

## Value

The percent variance bar plots from variancePartition!

## See Also

**variancePartition** `plotPercentBars`

---

| rex | *Send the R plotter to the computer of your choice!* |
|---|---|

---

## Description

Resets the display and xauthority variables to the new computer I am using so that plot() works.

## Usage

```
rex(display = ":0")
```

## Arguments

| | |
|---|---|
| `display` | DISPLAY variable to use, if NULL it looks in ~/.displays/$(host).last |

## Value

Fresh plotting window to the display of your choice!

s2s_all_filters    *Gather together the various SWATH2stats filters into one place.*

### Description

There are quite a few filters available in SWATH2stats. Reading the documentation, it seems at least possible, if not appropriate, to use them together when filtering DIA data before passing it to MSstats/etc. This function attempts to formalize and simplify that process.

### Usage

```
s2s_all_filters(
  s2s_exp,
  column = "proteinname",
  pep_column = "fullpeptidename",
  fft = 0.7,
  plot = FALSE,
  target_fdr = 0.02,
  upper_fdr = 0.05,
  mscore = 0.01,
  percentage = 0.75,
  remove_decoys = TRUE,
  max_peptides = 15,
  min_peptides = 2,
  do_mscore = TRUE,
  do_freqobs = TRUE,
  do_fdr = TRUE,
  do_proteotypic = TRUE,
  do_peptide = TRUE,
  do_max = TRUE,
  do_min = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| s2s_exp | SWHAT2stats result from the sample_annotation() function. (s2s_exp stands for: SWATH2stats experiment) |
| column | What column in the data contains the protein name? |
| pep_column | What column in the data contains the peptide name (not currently used, but it should be.) |
| fft | Ratio of false negatives to true positives, used by assess_by_fdr() and similar functions. |
| plot | Print plots of the various rates by sample? |
| target_fdr | When invoking mscore4assayfdr, choose an mscore which corresponds to this false discovery date. |

| upper_fdr | Used by filter_mscore_fdr() to choose the minimum threshold of identification confidence. |
| mscore | Mscore cutoff for the mscore filter. |
| percentage | Cutoff for the mscore_freqobs filter. |
| remove_decoys | Get rid of decoys in the final filter, if they were not already removed. |
| max_peptides | A maximum number of peptides filter. |
| min_peptides | A minimum number of peptides filter. |
| do_mscore | Perform the mscore filter? SWATH2stats::filter_mscore() |
| do_freqobs | Perform the mscore_freqobs filter? SWATH2stats::filter_mscore_freqobs() |
| do_fdr | Perform the fdr filter? SWATH2stats::filter_mscore_fdr() |
| do_proteotypic | Perform the proteotypic filter? SWATH2stats::filter_proteotypic_peptides() |
| do_peptide | Perform the single-peptide filter? SWATH2stats::filter_all_peptides() |
| do_max | Perform the maximum peptide filter? SWATH2stats::filter_max_peptides() |
| do_min | Perform the minimum peptide filter? SWATH2stats::filter_min_peptides() |
| ... | Other arguments passed down to the filters. |

## Value

Smaller SWATH2stats data set.

---

samtools_snp_coverage    *Use Rsamtools to read alignments and get snp coverage.*

---

## Description

This is horrifyingly slow. I think I might remove this function.

## Usage

```
samtools_snp_coverage(
  expt,
  type = "counts",
  input_dir = "preprocessing/outputs",
  tolower = TRUE,
  bam_suffix = ".bam",
  annot_column = annot_column
)
```

## Arguments

| expt | Expressionset to analyze |
| type | counts or percent? |
| input_dir | Directory containing the samtools results. |
| tolower | lowercase the sample names? |
| bam_suffix | In case the data came from sam. |
| annot_column | Passed along to count_expt_snps() |

## Value

It is so slow I no longer know if it works.

---

| | |
|---|---|
| sanitize_expt | *Get rid of characters which will mess up contrast making and such before playing with an expt.* |

---

## Description

Get rid of characters which will mess up contrast making and such before playing with an expt.

## Usage

```
sanitize_expt(expt)
```

## Arguments

| | |
|---|---|
| expt | An expt object to clean. |

---

| | |
|---|---|
| saveme | *Make a backup rdata file for future reference* |

---

## Description

I often use R over a sshfs connection, sometimes with significant latency, and I want to be able to save/load my R sessions relatively quickly. Thus this function uses pxz to compress the R session maximally and relatively fast. This assumes you have pxz installed and >= 4 CPUs.

## Usage

```
saveme(
  directory = "savefiles",
  backups = 2,
  cpus = 6,
  filename = "Rdata.rda.xz"
)
```

## Arguments

| | |
|---|---|
| directory | Directory to save the Rdata file. |
| backups | How many revisions? |
| cpus | How many cpus to use for the xz call |
| filename | Choose a filename. |

## Value

Command string used to save the global environment.

## See Also

[save](#) [pipe](#)

## Examples

```
## Not run:
 saveme()

## End(Not run)
```

---

score_mhess *A scoring function for the mh_ess TNSeq method.*

---

## Description

I dunno, I might delete this function, I am not sure if it will ever get use.

## Usage

```
score_mhess(expt, ess_column = "essm1")
```

## Arguments

| | |
|---|---|
| expt | Input expressionset with a metadata column with the ess output files. |
| ess_column | Metadata column containing the mh_ess output files. |

## Value

List containing the scores along with the genes which have changed using it.

---

semantic_copynumber_extract

*Extract multicopy genes from up/down gene expression lists.*

---

## Description

The function semantic_copynumber_filter() is the inverse of this.

## Usage

```
semantic_copynumber_extract(...)
```

**Arguments**

...          Arguments for semantic_copynumber_filter()

**Details**

Currently untested, used for Trypanosome analyses primarily, thus the default strings.

---

semantic_copynumber_filter

*Remove multicopy genes from up/down gene expression lists.*

---

**Description**

In our parasite data, there are a few gene types which are consistently obnoxious. Multi-gene families primarily where the coding sequences are divergent, but the UTRs nearly identical. For these genes, our sequence based removal methods fail and so this just excludes them by name.

**Usage**

```
semantic_copynumber_filter(
  input,
  max_copies = 2,
  use_files = FALSE,
  invert = TRUE,
  semantic = c("mucin", "sialidase", "RHS", "MASP", "DGF", "GP63"),
  semantic_column = "1.tooltip"
)
```

**Arguments**

| | |
|---|---|
| input | List of sets of genes deemed significantly up/down with a column expressing approximate count numbers. |
| max_copies | Keep only those genes with <= n putative copies. |
| use_files | Use a set of sequence alignments to define the copy numbers? |
| invert | Keep these genes rather than drop them? |
| semantic | Set of strings with gene names to exclude. |
| semantic_column | |
| | Column in the DE table used to find the semantic strings for removal. |

**Details**

Currently untested, used for Trypanosome analyses primarily, thus the default strings.

**Value**

Smaller list of up/down genes.

## See Also

[semantic_copynumber_extract](semantic_copynumber_extract)

## Examples

```
## Not run:
 pruned <- semantic_copynumber_filter(table, semantic=c("ribosomal"))
 ## Get rid of all genes with 'ribosomal' in the annotations.

## End(Not run)
```

---

semantic_expt_filter     *Remove/keep specifically named genes from an expt.*

---

## Description

I find subsetting weirdly confusing. Hopefully this function will allow one to include/exclude specific genes/families based on string comparisons.

## Usage

```
semantic_expt_filter(
  input,
  invert = FALSE,
  topn = NULL,
  semantic = c("mucin", "sialidase", "RHS", "MASP", "DGF", "GP63"),
  semantic_column = "description"
)
```

## Arguments

| | |
|---|---|
| input | Expt to filter. |
| invert | The default is to remove the genes with the semantic strings. Keep them when inverted. |
| topn | Take the topn most abundant genes rather than a text based heuristic. |
| semantic | Character list of strings to search for in the annotation data. |
| semantic_column | |
| | Column in the annotations to search. |

## Value

A presumably smaller expt.

---

sequence_attributes          *Gather some simple sequence attributes.*

---

### Description

This extends the logic of the pattern searching in pattern_count_genome() to search on some other attributes.

### Usage

```
sequence_attributes(fasta, gff = NULL, type = "gene", key = NULL)
```

### Arguments

| | |
|---|---|
| fasta | Genome encoded as a fasta file. |
| gff | Optional gff of annotations (if not provided it will just ask the whole genome). |
| type | Column of the gff file to use. |
| key | What type of entry of the gff file to key from? |

### Value

List of data frames containing gc/at/gt/ac contents.

### See Also

**Biostrings Rsamtools** `FaFile getSeq`

### Examples

```
pa_attribs <- sequence_attributes(fasta_file, gff=gff_file)
head(pa_attribs)
```

---

set_expt_batches          *Change the batches of an expt.*

---

### Description

When exploring differential analyses, it might be useful to play with the conditions/batches of the experiment. Use this to make that easier.

### Usage

```
set_expt_batches(expt, fact, ids = NULL, ...)
```

## Arguments

| | |
|---|---|
| `expt` | Expt to modify. |
| `fact` | Batches to replace using this factor. |
| `ids` | Specific samples to change. |
| `...` | Extra options are like spinach. |

## Value

The original expt with some new metadata.

## See Also

[create_expt](#) [set_expt_conditions](#)

## Examples

```
## Not run:
 expt = set_expt_batches(big_expt, factor=c(some,stuff,here))

## End(Not run)
```

---

set_expt_colors                    *Change the colors of an expt*

---

## Description

When exploring differential analyses, it might be useful to play with the conditions/batches of the experiment. Use this to make that easier.

## Usage

```
set_expt_colors(
  expt,
  colors = TRUE,
  chosen_palette = "Dark2",
  change_by = "condition"
)
```

## Arguments

| | |
|---|---|
| `expt` | Expt to modify |
| `colors` | colors to replace |
| `chosen_palette` | I usually use Dark2 as the RColorBrewer palette. |
| `change_by` | Assuming a list is passed, cross reference by condition or sample? |

**Value**

expt Send back the expt with some new metadata

**See Also**

[set_expt_conditions](#) [set_expt_batches](#)

**Examples**

```
## Not run:
unique(esmer_expt$design$conditions)
chosen_colors <- list(
    "cl14_epi" = "#FF8D59",
    "clbr_epi" = "#962F00",
    "cl14_tryp" = "#D06D7F",
    "clbr_tryp" = "#A4011F",
    "cl14_late" = "#6BD35E",
    "clbr_late" = "#1E7712",
    "cl14_mid" = "#7280FF",
    "clbr_mid" = "#000D7E")
esmer_expt <- set_expt_colors(expt=esmer_expt, colors=chosen_colors)

## End(Not run)
```

set_expt_conditions     *Change the condition of an expt*

**Description**

When exploring differential analyses, it might be useful to play with the conditions/batches of the experiment. Use this to make that easier.

**Usage**

```
set_expt_conditions(expt, fact = NULL, ids = NULL, null_cell = "null", ...)
```

**Arguments**

| | |
|---|---|
| expt | Expt to modify |
| fact | Conditions to replace |
| ids | Specific sample IDs to change. |
| null_cell | How to fill elements of the design which are null? |
| ... | Extra arguments are given to arglist. |

**Value**

expt Send back the expt with some new metadata

## See Also

set_expt_batches create_expt

## Examples

```
## Not run:
 expt = set_expt_conditions(big_expt, factor=c(some,stuff,here))

## End(Not run)
```

---

set_expt_factors          *Change the factors (condition and batch) of an expt*

---

## Description

When exploring differential analyses, it might be useful to play with the conditions/batches of the experiment. Use this to make that easier.

## Usage

```
set_expt_factors(expt, condition = NULL, batch = NULL, ids = NULL, ...)
```

## Arguments

| | |
|---|---|
| expt | Expt to modify |
| condition | New condition factor |
| batch | New batch factor |
| ids | Specific sample IDs to change. |
| ... | Arguments passed along (likely colors) |

## Value

expt Send back the expt with some new metadata

## See Also

set_expt_conditions set_expt_batches

## Examples

```
## Not run:
 expt = set_expt_factors(big_expt, condition="column", batch="another_column")

## End(Not run)
```

---

set_expt_genenames *Change the gene names of an expt.*

---

### Description

I want to change all the gene names of a big expressionset to the ortholog groups. But I want to also continue using my expts. Ergo this little function.

### Usage

```
set_expt_genenames(expt, ids = NULL, ...)
```

### Arguments

| | |
|---|---|
| expt | Expt to modify |
| ids | Specific sample IDs to change. |
| ... | Extra arguments are given to arglist. |

### Value

expt Send back the expt with some new metadata

### See Also

[set_expt_batches](#) [create_expt](#)

### Examples

```
## Not run:
 expt = set_expt_conditions(big_expt, factor=c(some,stuff,here))

## End(Not run)
```

---

set_expt_samplenames *Change the sample names of an expt.*

---

### Description

Sometimes one does not like the hpgl identifiers, so provide a way to change them on-the-fly.

### Usage

```
set_expt_samplenames(expt, newnames)
```

## Arguments

| | |
|---|---|
| `expt` | Expt to modify |
| `newnames` | New names, currently only a character vector. |

## Value

expt Send back the expt with some new metadata

## See Also

[set_expt_conditions](#) [set_expt_batches](#)

## Examples

```
## Not run:
 expt = set_expt_samplenames(expt, c("a","b","c","d","e","f"))

## End(Not run)
```

---

| | |
|---|---|
| `sig_ontologies` | *Take the result from extract_significant_genes() and perform ontology searches.* |

---

## Description

It can be annoying/confusing to extract individual sets of 'significant' genes from a differential expression analysis. This function should make that process easier.

## Usage

```
sig_ontologies(
  significant_result,
  excel_prefix = "excel/sig_ontologies",
  search_by = "deseq",
  excel_suffix = ".xlsx",
  type = "gprofiler",
  ...
)
```

## Arguments

| | |
|---|---|
| `significant_result` | |
| | Result from extract_siggenes() |
| `excel_prefix` | How to start the output filenames? |
| `search_by` | Use the definition of 'significant' from which program? |
| `excel_suffix` | How to end the excel filenames? |
| `type` | Which specific ontology search to use? |
| `...` | Arguments passed to the various simple_ontology() function. |

## Value

A list of the up/down results of the ontology searches.

---

| significant_barplots | *Given the set of significant genes from combine_de_tables(), provide a view of how many are significant up/down.* |

---

## Description

These plots are pretty annoying, and I am certain that this function is not well written, but it provides a series of bar plots which show the number of genes/contrast which are up and down given a set of fold changes and p-value.

## Usage

```
significant_barplots(
  combined,
  lfc_cutoffs = c(0, 1, 2),
  invert = FALSE,
  p = 0.05,
  z = NULL,
  p_type = "adj",
  according_to = "all",
  order = NULL,
  maximum = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| combined | Result from combine_de_tables and/or extract_significant_genes(). |
| lfc_cutoffs | Choose 3 fold changes to define the queries. 0, 1, 2 mean greater/less than 0 followed by 2 fold and 4 fold cutoffs. |
| invert | Reverse the order of contrasts for readability? |
| p | Chosen p-value cutoff. |
| z | Choose instead a z-score cutoff. |
| p_type | Adjusted or not? |
| according_to | limma, deseq, edger, basic, or all of the above. |
| order | Choose a specific order for the plots. |
| maximum | Set a specific limit on the number of genes on the x-axis. |
| ... | More arguments are passed to arglist. |

## Value

list containing the significance bar plots and some information to hopefully help interpret them.

## See Also

**ggplot2**

## Examples

```
## Not run:
 expt <- create_expt(metadata="some_metadata.xlsx", gene_info=annotations)
 pairwise_result <- all_pairwise(expt)
 combined_result <- combine_de_tables(pairwise_result)
 ## Damn I wish I were smrt enough to make this elegant, but I cannot.
 barplots <- significant_barplots(combined_result)

## End(Not run)
```

---

sillydist                    *Calculate a simplistic distance function of a point against two axes.*

---

## Description

Sillydist provides a distance of any point vs. the axes of a plot. This just takes the abs(distances) of each point to the axes, normalizes them against the largest point on the axes, multiplies the result, and normalizes against the max of all point.

## Usage

```
sillydist(firstterm, secondterm, firstaxis = 0, secondaxis = 0)
```

## Arguments

| | |
|---|---|
| firstterm | X-values of the points. |
| secondterm | Y-values of the points. |
| firstaxis | X-value of the vertical axis. |
| secondaxis | Y-value of the second axis. |

## Value

Dataframe of the distances.

## See Also

**ggplot2**

## Examples

```
## Not run:
 mydist <- sillydist(df[,1], df[,2], first_median, second_median)
 first_vs_second <- ggplot2::ggplot(df, ggplot2::aes_string(x="first", y="second"),
                                    environment=hpgl_env) +
   ggplot2::xlab(paste("Expression of", df_x_axis)) +
   ggplot2::ylab(paste("Expression of", df_y_axis)) +
  ggplot2::geom_vline(color="grey", xintercept=(first_median - first_mad), size=line_size) +
  ggplot2::geom_vline(color="grey", xintercept=(first_median + first_mad), size=line_size) +
   ggplot2::geom_vline(color="darkgrey", xintercept=first_median, size=line_size) +
  ggplot2::geom_hline(color="grey", yintercept=(second_median - second_mad), size=line_size) +
  ggplot2::geom_hline(color="grey", yintercept=(second_median + second_mad), size=line_size) +
   ggplot2::geom_hline(color="darkgrey", yintercept=second_median, size=line_size) +
   ggplot2::geom_point(colour=grDevices::hsv(mydist$dist, 1, mydist$dist),
                       alpha=0.6, size=size) +
   ggplot2::theme(legend.position="none")
 first_vs_second  ## dots get colored according to how far they are from the medians
 ## replace first_median, second_median with 0,0 for the axes

 ## End(Not run)
```

---

simple_clusterprofiler

*Perform the array of analyses in the 2016-04 version of clusterProfiler*

---

## Description

The new version of clusterProfiler has a bunch of new toys. However, it is more stringent in terms of input in that it now explicitly expects to receive annotation data in terms of a orgdb object. This is mostly advantageous, but will probably cause some changes in the other ontology functions in the near future. This function is an initial pass at making something similar to my previous 'simple_clusterprofiler()' but using these new toys.

## Usage

```
simple_clusterprofiler(
  sig_genes,
  de_table = NULL,
  orgdb = "org.Dm.eg.db",
  orgdb_from = NULL,
  orgdb_to = "ENTREZID",
  go_level = 3,
  pcutoff = 0.05,
  qcutoff = 0.1,
  fc_column = "logFC",
  second_fc_column = "limma_logfc",
  updown = "up",
  permutations = 1000,
```

```
    min_groupsize = 5,
    kegg_prefix = NULL,
    kegg_organism = NULL,
    do_gsea = TRUE,
    categories = 12,
    excel = NULL,
    do_david = FALSE,
    david_id = "ENTREZ_GENE_ID",
    david_user = "unknown@unknown.org"
)
```

## Arguments

| | |
|---|---|
| `sig_genes` | Dataframe of genes deemed 'significant.' |
| `de_table` | Dataframe of all genes in the analysis, primarily for gse analyses. |
| `orgdb` | Name of the orgDb used for gathering annotation data. |
| `orgdb_from` | Name of a key in the orgdb used to cross reference to entrez IDs. |
| `orgdb_to` | List of keys to grab from the orgdb for cross referencing ontologies. |
| `go_level` | How deep into the ontology tree should this dive for over expressed categories. |
| `pcutoff` | P-value cutoff for 'significant' analyses. |
| `qcutoff` | Q-value cutoff for 'significant' analyses. |
| `fc_column` | When extracting vectors of all genes, what column should be used? |
| `second_fc_column` | |
| | When extracting vectors of all genes, what column should be tried the second time around? |
| `updown` | Include the less than expected ontologies? |
| `permutations` | How many permutations for GSEA-ish analyses? |
| `min_groupsize` | Minimum size of an ontology before it is included. |
| `kegg_prefix` | Many KEGG ids need a prefix before they will cross reference. |
| `kegg_organism` | Choose the 3 letter KEGG organism name here. |
| `do_gsea` | Perform gsea searches? |
| `categories` | How many categories should be plotted in bar/dot plots? |
| `excel` | Print the results to an excel file? |
| `do_david` | Attempt to use the DAVID database for a search? |
| `david_id` | Which column to use for cross-referencing to DAVID? |
| `david_user` | Default registered username to use. |

## Value

a list

## See Also

**clusterProfiler**

## Examples

```
## Not run:
 holyasscrackers <- simple_clusterprofiler(gene_list, all_genes, "org.Dm.eg.db")

## End(Not run)
```

simple_cp_enricher       *Generic enrichment using clusterProfiler.*

## Description

culsterProfiler::enricher provides a quick and easy enrichment analysis given a set of siginficant' genes and a data frame which connects each gene to a category.

## Usage

```
simple_cp_enricher(sig_genes, de_table, go_db = NULL)
```

## Arguments

| | |
|---|---|
| sig_genes | Set of 'significant' genes as a table. |
| de_table | All genes from the original analysis. |
| go_db | Dataframe of GO->ID matching the gene names of sig_genes to GO categories. |

## Value

Table of 'enriched' categories.

simple_filter_counts     *Filter low-count genes from a data set only using a simple threshold and number of samples.*

## Description

This was a function written by Kwame Okrah and perhaps also Laura Dillon to remove low-count genes. It drops genes based on a threshold and number of samples.

## Usage

```
simple_filter_counts(count_table, threshold = 2)
```

## Arguments

| | |
|---|---|
| count_table | Data frame of (pseudo)counts by sample. |
| threshold | Lower threshold of counts for each gene. |

## Value

Dataframe of counts without the low-count genes.

## See Also

**edgeR**

## Examples

```
## Not run:
 filtered_table <- simple_filter_counts(count_table)

## End(Not run)
```

---

simple_gadem                    *run the rGADEM suite*

---

## Description

This should provide a set of rGADEM results given an input file of sequences and a genome.

## Usage

```
simple_gadem(
  inputfile,
  genome = "BSgenome.Hsapiens.UCSC.hs19",
  p = 0.1,
  e = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| inputfile | Fasta or bed file containing sequences to search. |
| genome | BSgenome to read. |
| ... | Parameters for plotting the gadem result. |

## Value

A list containing slots for plots, the stdout output from gadem, the gadem result, set of occurences of motif, and the returned set of motifs.

---

simple_goseq *Perform a simplified goseq analysis.*

---

## Description

goseq can be pretty difficult to get set up for non-supported organisms. This attempts to make that process a bit simpler as well as give some standard outputs which should be similar to those returned by clusterprofiler/topgo/gostats/gprofiler.

## Usage

```
simple_goseq(
  sig_genes,
  go_db = NULL,
  length_db = NULL,
  doplot = TRUE,
  adjust = 0.1,
  pvalue = 0.1,
  length_keytype = "transcripts",
  go_keytype = "entrezid",
  goseq_method = "Wallenius",
  padjust_method = "BH",
  bioc_length_db = "ensGene",
  expand_categories = TRUE,
  excel = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| sig_genes | Data frame of differentially expressed genes, containing IDs etc. |
| go_db | Database of go to gene mappings (OrgDb/OrganismDb) |
| length_db | Database of gene lengths (gff/TxDb) |
| doplot | Include pwf plots? |
| adjust | Minimum adjusted pvalue for 'significant.' |
| pvalue | Minimum pvalue for 'significant.' |
| length_keytype | Keytype to provide to extract lengths |
| go_keytype | Keytype to provide to extract go IDs |
| goseq_method | Statistical test for goseq to use. |
| padjust_method | Which method to use to adjust the pvalues. |
| bioc_length_db | Source of gene lengths? |
| expand_categories | |
| | Expand the GO categories to make the results more readable? |
| excel | Print the results to an excel file? |
| ... | Extra parameters which I do not recall |

## Value

Big list including: the pwd:pwf function, alldata:the godata dataframe, pvalue_histogram:p-value
histograms, godata_interesting:the ontology information of the enhanced groups, term_table:the
goterms with some information about them, mf_subset:a plot of the MF enhanced groups, mfp_plot:the
pvalues of the MF group, bp_subset:a plot of the BP enhanced groups, bpp_plot, cc_subset, and
ccp_plot

## See Also

**goseq GO.db**

## Examples

```
## Not run:
 lotsotables <- simple_goseq(gene_list, godb, lengthdb)

## End(Not run)
```

---

| simple_gostats | *Simplification function for gostats, in the same vein as those written* |
|---|---|
| | *for clusterProfiler, goseq, and topGO.* |

---

## Description

GOstats has a couple interesting peculiarities: Chief among them: the gene IDs must be integers.
As a result, I am going to have this function take a gff file in order to get the go ids and gene ids on
the same page.

## Usage

```
simple_gostats(
  sig_genes,
  go_db = NULL,
  gff = NULL,
  gff_df = NULL,
  universe_merge = "id",
  second_merge_try = "locus_tag",
  species = "fun",
  pcutoff = 0.1,
  conditional = FALSE,
  categorysize = NULL,
  gff_id = "ID",
  gff_type = "cds",
  excel = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `sig_genes` | Input list of differentially expressed genes. |
| `go_db` | Set of GOids, as before in the format ID/GO. |
| `gff` | Annotation information for this genome. |
| `gff_df` | I do not remember what this is for. |
| `universe_merge` | Column from which to create the universe of genes. |
| `second_merge_try` | |
| | If the first universe merge fails, try this. |
| `species` | Genbank organism to use. |
| `pcutoff` | Pvalue cutoff for deciding significant. |
| `conditional` | Perform a conditional search? |
| `categorysize` | Category size below which to not include groups. |
| `gff_id` | key in the gff file containing the unique IDs. |
| `gff_type` | Gff column to use for creating the universe. |
| `excel` | Print the results to an excel file? |
| `...` | More parameters! |

## Value

List of returns from GSEABase, Category, etc.

## See Also

**GSEABase Category**

## Examples

```
## Not run:
 knickerbockers <- simple_gostats(sig_genes, gff_file, goids)

## End(Not run)
```

---

simple_gprofiler       *Run searches against the web service g:Profiler.*

---

## Description

Thank you Ginger for showing me your thesis, gProfiler is pretty cool!

## Usage

```
simple_gprofiler(
  sig_genes,
  species = "hsapiens",
  convert = TRUE,
  first_col = "logFC",
  second_col = "limma_logfc",
  do_go = TRUE,
  do_kegg = TRUE,
  do_reactome = TRUE,
  do_mi = TRUE,
  do_tf = TRUE,
  do_corum = TRUE,
  do_hp = TRUE,
  significant = TRUE,
  pseudo_gsea = TRUE,
  id_col = "row.names",
  excel = NULL
)
```

## Arguments

| | |
|---|---|
| sig_genes | Guess! The set of differentially expressed/interesting genes. |
| species | Organism supported by gprofiler. |
| convert | Use gProfileR's conversion utility? |
| first_col | First place used to define the order of 'significant'. |
| second_col | If that fails, try a second column. |
| do_go | Perform GO search? |
| do_kegg | Perform KEGG search? |
| do_reactome | Perform reactome search? |
| do_mi | Do miRNA search? |
| do_tf | Search for transcription factors? |
| do_corum | Do corum search? |
| do_hp | Do the hp search? |
| significant | Only return the statistically significant hits? |
| pseudo_gsea | Is the data in a ranked order by significance? |
| id_col | Which column in the table should be used for gene ID crossreferencing? gProfiler uses Ensembl ids. So if you have a table of entrez or whatever, translate it! |
| excel | Print the results to an excel file? |

## Value

a list of results for go, kegg, reactome, and a few more.

## See Also

**gProfiler**

## Examples

```
## Not run:
 gprofiler_is_nice_and_easy <- simple_gprofiler(genes, species='mmusculus')

## End(Not run)
```

---

simple_gprofiler2       *Run searches against the web service g:Profiler.*

---

## Description

This is the beginning of a reimplementation to use gprofiler2. However, AFAICT gprofiler2 does not yet actually work for anything other than their GO data.

## Usage

```
simple_gprofiler2(
  sig_genes,
  species = "hsapiens",
  convert = TRUE,
  first_col = "logFC",
  second_col = "limma_logfc",
  do_go = TRUE,
  do_kegg = TRUE,
  do_reactome = TRUE,
  do_mi = TRUE,
  do_tf = TRUE,
  do_corum = TRUE,
  do_hp = TRUE,
  do_hpa = TRUE,
  do_wp = TRUE,
  significant = FALSE,
  exclude_iea = FALSE,
  do_under = FALSE,
  evcodes = TRUE,
  threshold = 0.05,
  adjp = "fdr",
  domain_scope = "annotated",
  bg = NULL,
  pseudo_gsea = TRUE,
  id_col = "row.names",
  excel = NULL
)
```

## Arguments

| | |
|---|---|
| sig_genes | Guess! The set of differentially expressed/interesting genes. |
| species | Organism supported by gprofiler. |
| convert | Use gProfileR's conversion utility? |
| first_col | First place used to define the order of 'significant'. |
| second_col | If that fails, try a second column. |
| do_go | Perform GO search? |
| do_kegg | Perform KEGG search? |
| do_reactome | Perform reactome search? |
| do_mi | Do miRNA search? |
| do_tf | Search for transcription factors? |
| do_corum | Do corum search? |
| do_hp | Do the hp search? |
| do_hpa | Do the hpa search? |
| do_wp | Do the wp search? |
| significant | Only return the statistically significant hits? |
| exclude_iea | Passed directly to gprofiler2. |
| do_under | Perform under-representation search? |
| evcodes | Get the set of evcodes in the data? This makes it take longer. |
| threshold | p-value 'significance' threshold. |
| adjp | Method to adjust p-values. |
| domain_scope | Passed to gprofiler2. |
| bg | Background genes. |
| pseudo_gsea | Is the data in a ranked order by significance? |
| id_col | Which column in the table should be used for gene ID crossreferencing? gProfiler uses Ensembl ids. So if you have a table of entrez or whatever, translate it! |
| excel | Print the results to an excel file? |

## Value

a list of results for go, kegg, reactome, and a few more.

## See Also

**gProfiler**

## Examples

```
## Not run:
 gprofiler_is_nice_and_easy <- simple_gprofiler(genes, species='mmusculus')

## End(Not run)
```

simple_gsva *Provide some defaults and guidance when attempting to use gsva.*

### Description

gsva seems to hold a tremendous amount of potential. Unfortunately, it is somewhat opaque and its requirements are difficult to pin down. This function will hopefully provide some of the requisite defaults and do some sanity checking to make it more likely that a gsva analysis will succeed.

### Usage

```
simple_gsva(
  expt,
  signatures = "c2BroadSets",
  data_pkg = "GSVAdata",
  signature_category = "c2",
  cores = 1,
  current_id = "ENSEMBL",
  required_id = "ENTREZID",
  min_catsize = 5,
  orgdb = "org.Hs.eg.db",
  method = "ssgsea",
  kcdf = NULL,
  ranking = FALSE
)
```

### Arguments

| | |
|---|---|
| expt | Expt object to be analyzed. |
| signatures | Provide an alternate set of signatures (GeneSetCollections) |
| data_pkg | What package contains the requisite dataset? |
| cores | How many CPUs to use? |
| current_id | Where did the IDs of the genes come from? |
| required_id | gsva (I assume) always requires ENTREZ IDs, but just in case this is a parameter. |
| min_catsize | This provides the min.sz parameter to gsva(). |
| orgdb | What is the data source for the rownames()? |
| method | Which gsva method to use? Changed this from gsva to ssgsea because it was throwing segmentation faults. |
| kcdf | Options for the gsva methods. |
| ranking | another gsva option. |
| datasets | Name of the variable from which to acquire the gsva data, if it does not exist, then data() will be called upon it. |

## Value

List containing three elements: first a modified expressionset using the result of gsva in place of the original expression data; second the result from gsva, and third a data frame of the annotation data for the gene sets in the expressionset. This seems a bit redundant, perhaps I should revisit it?

---

simple_mlseq                    *Use MLSeq to seek important genes given an experimental factor and*
                                *an expressionSet.*

---

## Description

MLSeq provides interfaces to the various machine learning methodologies from caret in the context of RNASeq data. It furthermore provides bridge methods which provide links from the normalization methods from limma/edgeR/DESeq2 to the various ML methods in caret.

## Usage

```
simple_mlseq(
  expt,
  comparison = "condition",
  number_by_var = 100,
  ceiling_factor = 1/3,
  training_number = 2,
  training_repeats = 10,
  training_method = "repeatedcv",
  classify_method = "svmRadial",
  classify_preprocess = "deseq-rlog",
  reference_factor = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| expt | Input expressionset. |
| comparison | Metadata column from the experimental design for the search. |
| number_by_var | Take the top-n most variant genes. Use all genes if null. |
| ceiling_factor | Define how many columns(experimental samples) to take when sampling the expressionset for training vs. testing data. |
| training_number | |
| | Iterations when training. |
| training_repeats | |
| | Also iterations when training... (in other words, I dunno). |
| training_method | |
| | which caret method to train? |

classify_method

        which caret method to classify the data?

classify_preprocess

        Which mlseq method to preprocess/normalize the data?

reference_factor

        What factor in the experimental metadata contains the reference?

...        Extra arguments

---

simple_pathview       *Print some data onto KEGG pathways.*

---

## Description

KEGGREST and pathview provide neat functions for coloring molecular pathways with arbitrary data. Unfortunately they are somewhat evil to use. This attempts to alleviate that.

## Usage

```
simple_pathview(
  path_data,
  indir = "pathview_in",
  outdir = "pathview",
  pathway = "all",
  species = "lma",
  from_list = NULL,
  to_list = NULL,
  suffix = "_colored",
  filenames = "id",
  fc_column = "limma_logfc",
  format = "png",
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| path_data | Some differentially expressed genes. |
| indir | Directory into which the unmodified kegg images will be downloaded (or already exist). |
| outdir | Directory which will contain the colored images. |
| pathway | Perform the coloring for a specific pathway? |
| species | Kegg identifier for the species of interest. |
| from_list | Regex to help in renaming KEGG categories/gene names from one format to another. |
| to_list | Regex to help in renaming KEGG categories/gene names from one format to another. |

| suffix | Add a suffix to the completed, colored files. |
|---|---|
| filenames | Name the final files by id or name? |
| fc_column | What is the name of the fold-change column to extract? |
| format | Format of the resulting images, I think only png really works well. |
| verbose | When on, this function is quite chatty. |

## Value

A list of some information for every KEGG pathway downloaded/examined. This information includes: a. The filename of the final image for each pathway. b. The number of genes which were found in each pathway image. c. The number of genes in the 'up' category d. The number of genes in the 'down' category

## See Also

**Ramigo pathview**

## Examples

```
## Not run:
 thy_el_comp2_path = hpgl_pathview(thy_el_comp2_kegg, species="spz", indir="pathview_in",
                                   outdir="kegg_thy_el_comp2", string_from="_Spy",
                                   string_to="_Spy_", filenames="pathname")

## End(Not run)
```

---

simple_proper                    *Invoke PROPER and replace its default data set with data of interest.*

---

## Description

Recent reviewers of Najib's grants have taken an increased interest in knowing the statistical power of the various experiments. He queried Dr. Corrada-Bravo who suggested PROPER. I spent some time looking through it and, with some reservations, modified its workflow to (at least in theory) be able to examine any dataset. The workflow in question is particularly odd and warrants further discussion/analysis. This function is a modified version of 'default_proper()' above and invokes PROPER after re-formatting a given dataset in the way expected by PROPER.

## Usage

```
simple_proper(
  de_tables,
  p = 0.05,
  experiment = "cheung",
  nsims = 20,
  reps = c(3, 5, 7, 10),
  de_method = "edger",
```

```
    alpha_type = "fdr",
    alpha = 0.1,
    stratify = "expr",
    target = "lfc",
    mean_or_median = "mean",
    filter = "none",
    delta = 0.5
)
```

## Arguments

| | |
|---|---|
| de_tables | A set of differential expression results, presumably from EdgeR or DESeq2. |
| p | Cutoff |
| experiment | The default data set in PROPER is entitled 'cheung'. |
| nsims | Number of simulations to perform. |
| reps | Simulate these number of experimental replicates. |
| de_method | There are a couple choices here for tools which are pretty old, my version of this only accepts deseq or edger. |
| alpha_type | I assume p-adjust type. |
| alpha | Accepted fdr rate. |
| stratify | There are a few options here, I don't fully understand them. |
| target | Cutoff. |
| mean_or_median | Use mean or median values? |
| filter | Apply a filter? |
| delta | Not epsilon! (E.g. I forget what this does). |

## Value

A list containin the various tables and plots returned by PROPER.

---

simple_topgo                  *Perform a simplified topgo analysis.*

---

## Description

This will attempt to make it easier to run topgo on a set of genes.

## Usage

```
simple_topgo(
  sig_genes,
  goid_map = "id2go.map",
  go_db = NULL,
  pvals = NULL,
  limitby = "fisher",
  limit = 0.1,
  signodes = 100,
  sigforall = TRUE,
  numchar = 300,
  selector = "topDiffGenes",
  pval_column = "adj.P.Val",
  overwrite = FALSE,
  densities = FALSE,
  pval_plots = TRUE,
  excel = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| sig_genes | Data frame of differentially expressed genes, containing IDs any other columns. |
| goid_map | File containing mappings of genes to goids in the format expected by topgo. |
| go_db | Data frame of the goids which may be used to make the goid_map. |
| pvals | Set of pvalues in the DE data which may be used to improve the topgo results. |
| limitby | Test to index the results by. |
| limit | Ontology pvalue to use as the lower limit. |
| signodes | I don't remember right now. |
| sigforall | Provide the significance for all nodes? |
| numchar | Character limit for the table of results. |
| selector | Function name for choosing genes to include. |
| pval_column | Column from which to acquire scores. |
| overwrite | Yeah I do not remember this one either. |
| densities | Densities, yeah, the densities... |
| pval_plots | Include pvalue plots of the results a la clusterprofiler? |
| excel | Print the results to an excel file? |
| ... | Other options which I do not remember right now! |

## Value

Big list including the various outputs from topgo

## See Also

**topGO**

---

| simple_varpart | *Use variancePartition to try and understand where the variance lies in a data set.* |
|---|---|

---

### Description

The arguments and usage of variancePartition are a bit opaque. This function attempts to fill in reasonable values and simplify its invocation.

### Usage

```
simple_varpart(
  expt,
  predictor = NULL,
  factors = c("condition", "batch"),
  chosen_factor = "batch",
  do_fit = FALSE,
  cor_gene = 1,
  cpus = NULL,
  genes = 40,
  parallel = TRUE,
  mixed = FALSE,
  modify_expt = TRUE
)
```

### Arguments

| | |
|---|---|
| expt | Some data |
| predictor | Non-categorical predictor factor with which to begin the model. |
| factors | Character list of columns in the experiment design to query |
| chosen_factor | When checking for sane 'batches', what column to extract from the design? |
| do_fit | Perform a fitting using variancePartition? |
| cor_gene | Provide a set of genes to look at the correlations, defaults to the first gene. |
| cpus | Number cpus to use |
| genes | Number of genes to count. |
| parallel | Use doParallel? |
| modify_expt | Add annotation columns with the variance/factor? |

### Value

List of plots and variance data frames

### See Also

**doParallel variancePartition**

---

simple_xcell                    *Invoke xCell and pretty-ify the result.*

---

### Description

I initially thought xCell might prove the best tool/method for exploring cell deconvolution. I slowly
figured out its limitations, but still think it seems pretty nifty for its use case. Thus this function is
intended to make invoking it easier/faster.

### Usage

```
simple_xcell(
  expt,
  signatures = NULL,
  genes = NULL,
  spill = NULL,
  expected_types = NULL,
  label_size = NULL,
  col_margin = 6,
  row_margin = 12,
  ...
)
```

### Arguments

| | |
|---|---|
| expt | Expressionset to query. |
| signatures | Alternate set of signatures to use. |
| genes | Subset of genes to query. |
| spill | The xCell spill parameter. |
| expected_types | Set of assumed types in the data. |
| label_size | How large to make labels when printing the final heatmap. |
| col_margin | Used by par() when printing the final heatmap. |
| row_margin | Ibid. |
| ... | Extra arguments when normalizing the data for use with xCell. |

### Value

Small list providing the output from xCell, the set of signatures, and heatmap.

| sm | *Silence* |
|---|---|

## Description

Some libraries/functions just won't shut up. Ergo, silence, peasant! This is a simpler silence peasant.

## Usage

```
sm(..., wrap = TRUE)
```

## Arguments

| | |
|---|---|
| `...` | Some code to shut up. |
| `wrap` | Wrap the invocation and try again if it failed? |

## Value

Whatever the code would have returned.

| snp_by_chr | *The real worker. This extracts positions for a single chromosome and puts them into a parallelizable data structure.* |
|---|---|

## Description

The real worker. This extracts positions for a single chromosome and puts them into a parallelizable data structure.

## Usage

```
snp_by_chr(medians, chr_name = "01", limit = 1)
```

## Arguments

| | |
|---|---|
| `medians` | A set of medians by position to look through |
| `chr_name` | Chromosome name to search |
| `limit` | Minimum number of median hits/position to count as a snp. |

## Value

A list of variant positions where each element is one chromosome.

---

snp_subset_genes                    *Look for only the variant positions in a subset of genes.*

---

### Description

This was written in response to a query from Nancy and Maria Adelaida who wanted to look only at the variant positions in a few specific genes.

### Usage

```
snp_subset_genes(
  expt,
  snp_expt,
  start_col = "start",
  end_col = "end",
  expt_name_col = "chromosome",
  snp_name_col = "chromosome",
  snp_start_col = "position",
  expt_gid_column = "gid",
  genes = c("LPAL13_120010900", "LPAL13_340013000", "LPAL13_000054100",
    "LPAL13_140006100", "LPAL13_180018500", "LPAL13_320022300")
)
```

### Arguments

| | |
|---|---|
| expt | Initial expressionset. |
| snp_expt | Variant position expressionset. |
| start_col | Metadata column with the start positions for each gene. |
| end_col | Metadata column with the end of the genes. |
| expt_name_col | Metadata column with the chromosome names. |
| snp_name_col | Ditto for the snp_expressionset. |
| snp_start_col | Metadata column containing the variant positions. |
| expt_gid_column | |
| | ID column for the genes. |
| genes | Set of genes to cross reference. |

### Value

New expressionset with only the variants for the genes of interest.

snps_intersections    *Cross reference observed variants against the transcriptome annotation.*

### Description

This function should provide counts of how many variant positions were observed with respect to each chromosome and with respect to each annotated sequence (currently this is limited to CDS, but that is negotiable).

### Usage

```
snps_intersections(
  expt,
  snp_result,
  start_column = "start",
  end_column = "end",
  chr_column = "seqnames"
)
```

### Arguments

| | |
|---|---|
| expt | The original expressionset. This provides the annotation data. |
| snp_result | The result from get_snp_sets or count_expt_snps. |
| chr_column | Column in the annotation with the chromosome names. |

### Value

List containing the set of intersections in the conditions contained in snp_result, the summary of numbers of variants per chromosome, and

### See Also

[snps_vs_genes](#)

### Examples

```
## Not run:
expt <- create_expt(metadata, gene_information)
snp_expt <- count_expt_snps(expt)
snp_result <- get_snp_sets(snp_expt)
intersections <- snps_vs_intersections(expt, snp_result)

## End(Not run)
```

---

snps_vs_genes                    *Make a summary of the observed snps by gene ID.*

---

### Description

Instead of cross referencing variant positions against experimental condition, one might be interested in seeing what variants are observed per gene. This function attempts to answer that question.

### Usage

```
snps_vs_genes(
  expt,
  snp_result,
  start_col = "start",
  end_col = "end",
  snp_name_col = "seqnames",
  expt_name_col = "chromosome"
)
```

### Arguments

| | |
|---|---|
| expt | The original expressionset. |
| snp_result | The result from get_snp_sets(). |
| start_col | Which column provides the start of each gene? |
| end_col | and the end column of each gene? |
| snp_name_col | Name of the column in the metadata with the sequence names. |
| expt_name_col | Name of the metadata column with the chromosome names. |

### Value

List with some information by gene.

### Examples

```
## Not run:
expt <- create_expt(metadata, gene_information)
snp_expt <- count_expt_snps(expt)
snp_result <- get_snp_sets(snp_expt)
gene_intersections <- snps_vs_genes(expt, snp_result)

## End(Not run)
```

---

| | |
|---|---|
| subset_expt | *Extract a subset of samples following some rule(s) from an experiment class.* |

---

### Description

Sometimes an experiment has too many parts to work with conveniently, this operation allows one to break it into smaller pieces.

### Usage

```
subset_expt(expt, subset = NULL, ids = NULL, coverage = NULL)
```

### Arguments

| | |
|---|---|
| expt | Expt chosen to extract a subset of data. |
| subset | Valid R expression which defines a subset of the design to keep. |
| ids | List of sample IDs to extract. |
| coverage | Request a minimum coverage/sample rather than text-based subset. |

### Value

metadata Expt class which contains the smaller set of data.

### See Also

**Biobase** [pData](#) [exprs](#) [fData](#)

### Examples

```
## Not run:
 smaller_expt <- expt_subset(big_expt, "condition=='control'")
 all_expt <- expt_subset(expressionset, "")  ## extracts everything

## End(Not run)
```

```
subset_ontology_search
```
                                    *Perform ontology searches on up/down subsets of differential expres-*
                                    *sion data.*

### Description

In the same way all_pairwise() attempts to simplify using multiple DE tools, this function seeks to
make it easier to extract subsets of differentially expressed data and pass them to goseq, clusterPro-
filer, topGO, GOstats, and gProfiler.

### Usage

```
subset_ontology_search(
  changed_counts,
  doplot = TRUE,
  do_goseq = TRUE,
  do_cluster = TRUE,
  do_topgo = TRUE,
  do_gostats = TRUE,
  do_gprofiler = TRUE,
  according_to = "limma",
  ...
)
```

### Arguments

| | |
|---|---|
| changed_counts | List of changed counts as ups and downs. |
| doplot | Include plots in the results? |
| do_goseq | Perform goseq search? |
| do_cluster | Perform clusterprofiler search? |
| do_topgo | Perform topgo search? |
| do_gostats | Perform gostats search? |
| do_gprofiler | Do a gprofiler search? |
| according_to | If results from multiple DE tools were passed, which one defines 'significant'? |
| ... | Extra arguments! |

### Value

List of ontology search results, up and down for each contrast.

### See Also

**goseq clusterProfiler topGO goStats gProfiler**

sum_eupath_exon_counts

> *I want an easy way to sum counts in eupathdb-derived data sets. These have a few things which should make this relatively easy. Notably: The gene IDs look like: "exon_ID-1 exon_ID-2 exon_ID-3" Therefore we should be able to quickly merge these.*

## Description

I want an easy way to sum counts in eupathdb-derived data sets. These have a few things which should make this relatively easy. Notably: The gene IDs look like: "exon_ID-1 exon_ID-2 exon_ID-3" Therefore we should be able to quickly merge these.

## Usage

```
sum_eupath_exon_counts(counts)
```

## Arguments

counts          Matrix/df/dt of count data.

## Value

The same data type but with the exons summed.

sum_exon_widths          *Given a data frame of exon counts and annotation information, sum the exons.*

## Description

This function will merge a count table to an annotation table by the child column. It will then sum all rows of exons by parent gene and sum the widths of the exons. Finally it will return a list containing a df of gene lengths and summed counts.

## Usage

```
sum_exon_widths(
  data = NULL,
  gff = NULL,
  annotdf = NULL,
  parent = "Parent",
  child = "row.names"
)
```

## Arguments

| | |
|---|---|
| `data` | Count tables of exons. |
| `gff` | Gff filename. |
| `annotdf` | Dataframe of annotations (probably from load_gff_annotations). |
| `parent` | Column from the annotations with the gene names. |
| `child` | Column from the annotations with the exon names. |

## Value

List of 2 data frames, counts and lengths by summed exons.

## Author(s)

Keith Hughitt with some modifications by atb.

## See Also

**rtracklayer** [`load_gff_annotations`](load_gff_annotations)

## Examples

```
## Not run:
 summed <- sum_exon_widths(counts, gff="reference/xenopus_laevis.gff.xz")

## End(Not run)
```

---

sva_modify_pvalues          *Use sva's f.pvalue to adjust p-values for data adjusted by combat.*

---

## Description

This is from section 5 of the sva manual: "Adjusting for surrogate values using the f.pvalue function." The following chunk of code is longer and more complex than I would like. This is because f.pvalue() assumes a pairwise comparison of a data set containing only two experimental factors. As a way to provide an example of _how_ to calculate appropriately corrected p-values for surrogate factor adjusted models, this is great; but when dealing with actual data, it falls a bit short.

## Usage

```
sva_modify_pvalues(results)
```

## Arguments

| | |
|---|---|
| `results` | Table of differential expression results. |

---

table_style          *Set the xlsx table style*

---

### Description

Set the xlsx table style

### Usage

```
table_style
```

### Format

An object of class `character` of length 1.

---

tnseq_multi_saturation

                   *Plot the saturation of multiple libraries simultaneously.*

---

### Description

Plot the saturation of multiple libraries simultaneously.

### Usage

```
tnseq_multi_saturation(
  meta,
  meta_column,
  ylimit = 100,
  column = "Reads",
  adjust = 1,
  ggstatsplot = FALSE
)
```

### Arguments

| | |
|---|---|
| meta | Experimental metadata |
| meta_column | Metadata column containing the filenames to query. |
| ylimit | Maximum y axis |
| column | Data file column to use for density calculation. |
| adjust | Density adjustment. |
| ggstatsplot | Include pretty ggstatsplot plot? |

### Value

a plot and table of the saturation for all samples.

---

tnseq_saturation                 *Make a plot and some simple numbers about tnseq saturation*

---

**Description**

This function takes as input a tab separated file from essentiality_tas.pl This is a perl script written to read a bam alignment of tnseq reads against a genome and count how many hits were observed on every TA in the given genome. It furthermore has some logic to tell the difference between reads which were observed on the forward vs. reverse strand as well as reads which appear to be on both strands (eg. they start and end with 'TA').

**Usage**

```
tnseq_saturation(data, column = "Reads", ylimit = 100, adjust = 2)
```

**Arguments**

| | |
|---|---|
| data | data to plot |
| column | which column to use for plotting |
| ylimit | Define the y axis? |
| adjust | Prettification parameter from ggplot2. |

**Value**

A plot and some numbers:

1. maximum_reads = The maximum number of reads observed in a single position.
2. hits_by_position = The full table of hits / position
3. num_hit_table = A table of how many times every number of hits was observed.
4. eq_0 = How many times were 0 hits observed?
5. gt_1 = How many positions have > 1 hit?
6. gt_2 = How many positions have > 2 hits?
7. gt_4 = How many positions have > 4 hits?
8. gt_8 = How many positions have > 8 hits?
9. gt_16 = How many positions have > 16 hits?
10. gt_32 = How many positions have > 32 hits?
11. ratios = Character vector of the ratios of each number of hits vs. 0 hits.
12. hit_positions = 2 column data frame of positions and the number of observed hits.
13. hits_summary = summary(hit_positions)
14. plot = Histogram of the number of hits observed.

**See Also**

**ggplot2**

## Examples

```
## Not run:
input <- "preprocessing/hpgl0837/essentiality/hpgl0837-trimmed_ca_ta-v0M1.wig"
saturation <- tnseq_saturation(file=input)

## End(Not run)
```

---

topDiffGenes                    *A very simple selector of strong scoring genes (by p-value)*

---

## Description

This function was provided in the topGO documentation, but not defined. It was copied/pasted here.
I have ideas for including up/down expression but have so far deemed them not needed because I
am feeding topGO already explicit lists of genes which are up/down/whatever. But it still is likely
to be useful to be able to further subset the data.

## Usage

```
topDiffGenes(allScore)
```

## Arguments

allScore            The scores of the genes

---

topgo_tables                    *Make pretty tables out of topGO data*

---

## Description

The topgo function GenTable is neat, but it needs some simplification to not be obnoxious.

## Usage

```
topgo_tables(
  result,
  limit = 0.1,
  limitby = "fisher",
  numchar = 300,
  orderby = "fisher",
  ranksof = "fisher"
)
```

## Arguments

| | |
|---|---|
| `result` | Topgo result. |
| `limit` | Pvalue limit defining 'significant'. |
| `limitby` | Type of test to perform. |
| `numchar` | How many characters to allow in the description? |
| `orderby` | Which of the available columns to order the table by? |
| `ranksof` | Which of the available columns are used to rank the data? |

## Value

prettier tables

## See Also

**topGO**

---

`topgo_trees`        *Print trees from topGO.*

---

## Description

The tree printing functionality of topGO is pretty cool, but difficult to get set correctly.

## Usage

```
topgo_trees(
  tg,
  score_limit = 0.01,
  sigforall = TRUE,
  do_mf_fisher_tree = TRUE,
  do_bp_fisher_tree = TRUE,
  do_cc_fisher_tree = TRUE,
  do_mf_ks_tree = FALSE,
  do_bp_ks_tree = FALSE,
  do_cc_ks_tree = FALSE,
  do_mf_el_tree = FALSE,
  do_bp_el_tree = FALSE,
  do_cc_el_tree = FALSE,
  do_mf_weight_tree = FALSE,
  do_bp_weight_tree = FALSE,
  do_cc_weight_tree = FALSE,
  parallel = FALSE
)
```

## Arguments

| | |
|---|---|
| `tg` | Data from simple_topgo(). |
| `score_limit` | Score limit to decide whether to add to the tree. |
| `sigforall` | Add scores to the tree? |
| `do_mf_fisher_tree` | |
| | Add the fisher score molecular function tree? |
| `do_bp_fisher_tree` | |
| | Add the fisher biological process tree? |
| `do_cc_fisher_tree` | |
| | Add the fisher cellular component tree? |
| `do_mf_ks_tree` | Add the ks molecular function tree? |
| `do_bp_ks_tree` | Add the ks biological process tree? |
| `do_cc_ks_tree` | Add the ks cellular component tree? |
| `do_mf_el_tree` | Add the el molecular function tree? |
| `do_bp_el_tree` | Add the el biological process tree? |
| `do_cc_el_tree` | Add the el cellular component tree? |
| `do_mf_weight_tree` | |
| | Add the weight mf tree? |
| `do_bp_weight_tree` | |
| | Add the bp weighted tree? |
| `do_cc_weight_tree` | |
| | Add the guess |
| `parallel` | Perform operations in parallel to speed this up? |

## Value

Big list including the various outputs from topgo.

## See Also

**topGO**

---

| | |
|---|---|
| `transform_counts` | *Perform a simple transformation of a count table (log2)* |

---

## Description

the add argument is only important if the data was previously cpm'd because that does a +1, thus this will avoid a double+1 on the data.

## Usage

```
transform_counts(count_table, design = NULL, method = "raw", base = NULL, ...)
```

## Arguments

| | |
|---|---|
| `count_table` | A matrix of count data |
| `design` | Sometimes the experimental design is also required. |
| `method` | A type of transformation to perform: log2/log10/log. |
| `base` | Other log scales? |
| `...` | Options I might pass from other functions are dropped into arglist. |

## Value

dataframe of transformed counts.

## See Also

**limma**

## Examples

```
## Not run:
 filtered_table = transform_counts(count_table, transform='log2', converted='cpm')

## End(Not run)
```

---

| | |
|---|---|
| u_plot | *Plot the rank order svd$u elements to get a view of how much the first genes contribute to the total variance by PC.* |

---

## Description

Plot the rank order svd$u elements to get a view of how much the first genes contribute to the total variance by PC.

## Usage

```
u_plot(plotted_us)
```

## Arguments

| | |
|---|---|
| `plotted_us` | a list of svd$u elements |

## Value

a recordPlot() plot showing the first 3 PCs by rank-order svd$u.

---

unAsIs                          *Remove the AsIs attribute from some data structure.*

---

## Description

Notably, when using some gene ontology libraries, the returned data structures include information which is set to type 'AsIs' which turns out to be more than slightly difficult to work with.

## Usage

```
unAsIs(stuff)
```

## Arguments

stuff            The data from which to remove the AsIs classification.

---

varpart_summaries           *Attempt to use variancePartition's fitVarPartModel() function.*

---

## Description

Note the word 'attempt'. This function is so ungodly slow that it probably will never be used.

## Usage

```
varpart_summaries(expt, factors = c("condition", "batch"), cpus = 6)
```

## Arguments

expt             Input expressionset.

factors          Set of factors to query

cpus             Number of cpus to use in doParallel.

## Value

Summaries of the new model, in theory this would be a nicely batch-corrected data set.

## See Also

**variancePartition**

---

what_happened                    *Print a string describing what happened to this data.*

---

### Description

Sometimes it is nice to have a string like: log2(cpm(data)) describing what happened to the data.

### Usage

```
what_happened(
  expt = NULL,
  transform = "raw",
  convert = "raw",
  norm = "raw",
  filter = "raw",
  batch = "raw"
)
```

### Arguments

| | |
|---|---|
| expt | The expressionset. |
| transform | How was it transformed? |
| convert | How was it converted? |
| norm | How was it normalized? |
| filter | How was it filtered? |
| batch | How was it batch-corrected? |

### Value

An expression describing what has been done to this data.

### See Also

[create_expt](#)

---

write_basic                    *Writes out the results of a basic search using write_de_table()*

---

### Description

Looking to provide a single interface for writing tables from basic and friends.

### Usage

```
write_basic(data, ...)
```

## Arguments

| | |
|---|---|
| data | Output from basic_pairwise() |
| ... | Options for writing the xlsx file. |

## Details

Tested in test_26basic.R

## See Also

[write_de_table](#)

## Examples

```
## Not run:
 finished_comparison <- basic_pairwise(expressionset)
 data_list <- write_basic(finished_comparison)

## End(Not run)
```

---

write_combined_legend    *Write the legend of an excel file for combine_de_tables()*

---

## Description

Write the legend of an excel file for combine_de_tables()

## Usage

```
write_combined_legend(
  wb,
  excel_basename,
  plot_dim,
  apr,
  limma,
  include_limma,
  deseq,
  include_deseq,
  edger,
  include_edger,
  ebseq,
  include_ebseq,
  basic,
  include_basic,
  padj_type
)
```

**Arguments**

| | |
|---|---|
| `wb` | Workbook to write |
| `excel_basename` | Where to write it |
| `plot_dim` | Default plot size. |
| `apr` | The all_pairwise() result. |
| `limma` | The limma result, which is redundant. |
| `include_limma` | Include the limma result? |
| `deseq` | The deseq result, which is redundant. |
| `include_deseq` | Include the deseq result? |
| `edger` | The edger result, which is redundant. |
| `include_edger` | Include the edger result? |
| `ebseq` | The ebseq result, which is redundant. |
| `include_ebseq` | Include the ebseq result? |
| `basic` | Basic data |
| `include_basic` | Include the basic result? |
| `padj_type` | P-adjustment employed. |

---

`write_combined_summary`
                        *Internal function to write a summary of some combined data*

---

**Description**

Internal function to write a summary of some combined data

**Usage**

```
write_combined_summary(
  wb,
  excel_basename,
  apr,
  extracted,
  compare_plots,
  lfc_cutoff = 1,
  p_cutoff = 0.05
)
```

**Arguments**

| | |
|---|---|
| `wb` | xlsx workbook to which to write. |
| `excel_basename` | basename for printing plots. |
| `apr` | a pairwise result |
| `extracted` | table extracted from the pairwise result |
| `compare_plots` | series of plots to print out. |

---

write_cp_data *Make a pretty table of clusterprofiler data in excel.*

---

### Description

It is my intention to make a function like this for each ontology tool in my repetoire

### Usage

```
write_cp_data(
  cp_result,
  excel = "excel/clusterprofiler.xlsx",
  wb = NULL,
  add_trees = TRUE,
  order_by = "qvalue",
  pval = 0.1,
  add_plots = TRUE,
  height = 15,
  width = 10,
  decreasing = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| cp_result | A set of results from simple_clusterprofiler(). |
| excel | An excel file to which to write some pretty results. |
| wb | Workbook object to write to. |
| add_trees | Include topgoish ontology trees? |
| order_by | What column to order the data by? |
| pval | Choose a cutoff for reporting by p-value. |
| add_plots | Include some pvalue plots in the excel output? |
| height | Height of included plots. |
| width | and their width. |
| decreasing | which direction? |
| ... | Extra arguments are passed to arglist. |

### Value

The result from openxlsx in a prettyified xlsx file.

### See Also

**openxlsx goseq**

---

write_de_table            *Writes out the results of a single pairwise comparison.*

---

### Description

However, this will do a couple of things to make one's life easier: 1. Make a list of the output, one element for each comparison of the contrast matrix. 2. Write out the results() output for them in separate sheets in excel. 3. Since I have been using qvalues a lot for other stuff, add a column.

### Usage

```
write_de_table(data, type = "limma", excel = "de_table.xlsx", ...)
```

### Arguments

| | |
|---|---|
| data | Output from results(). |
| type | Which DE tool to write. |
| excel | Filename into which to save the xlsx data. |
| ... | Parameters passed downstream, dumped into arglist and passed, notably the number of genes (n), the coefficient column (coef) |

### Details

Tested in test_24deseq.R Rewritten in 2016-12 looking to simplify combine_de_tables(). That function is far too big, this should become a template for that.

### Value

List of data frames comprising the toptable output for each coefficient, I also added a qvalue entry to these toptable() outputs.

### See Also

[write_xlsx](#)

### Examples

```
## Not run:
 finished_comparison <- eBayes(deseq_output)
 data_list <- write_deseq(finished_comparison, workbook="excel/deseq_output.xls")

## End(Not run)
```

---

write_deseq                *Writes out the results of a deseq search using write_de_table()*

---

### Description

Looking to provide a single interface for writing tables from deseq and friends.

### Usage

```
write_deseq(data, ...)
```

### Arguments

| | |
|---|---|
| data | Output from deseq_pairwise() |
| ... | Options for writing the xlsx file. |

### Details

Tested in test_24deseq.R

### See Also

**DESeq2** [write_xlsx](write_xlsx)

### Examples

```
## Not run:
 finished_comparison <- deseq2_pairwise(expressionset)
 data_list <- write_deseq(finished_comparison)

## End(Not run)
```

---

write_edger                *Writes out the results of a edger search using write_de_table()*

---

### Description

Looking to provide a single interface for writing tables from edger and friends.

### Usage

```
write_edger(data, ...)
```

### Arguments

| | |
|---|---|
| data | Output from deseq_pairwise() |
| ... | Options for writing the xlsx file. |

## Details

Tested in test_26edger.R

## See Also

**limma** [toptable](#) [write_xlsx](#)

## Examples

```
## Not run:
 finished_comparison <- edger_pairwise(expressionset)
 data_list <- write_edger(finished_comparison, excel="edger_result.xlsx")

## End(Not run)
```

---

write_expt                          *Make pretty xlsx files of count data.*

---

## Description

Some folks love excel for looking at this data. ok.

## Usage

```
write_expt(
  expt,
  excel = "excel/pretty_counts.xlsx",
  norm = "quant",
  violin = TRUE,
  sample_heat = TRUE,
  convert = "cpm",
  transform = "log2",
  batch = "sva",
  filter = TRUE,
  med_or_mean = "mean",
  ...
)
```

## Arguments

| | |
|---|---|
| expt | An expressionset to print. |
| excel | Filename to write. |
| norm | Normalization to perform. |
| violin | Include violin plots? |
| sample_heat | Include sample heatmaps? |
| convert | Conversion to perform. |

| | |
|---|---|
| transform | Transformation used. |
| batch | Batch correction applied. |
| filter | Filtering method used. |
| med_or_mean | When printing mean by condition, one may want median. |
| ... | Parameters passed down to methods called here (graph_metrics, etc). |

### Details

Tested in test_03graph_metrics.R This performs the following: Writes the raw data, graphs the raw data, normalizes the data, writes it, graphs it, and does a median-by-condition and prints that. I replaced the openxlsx function which writes images into xlsx files with one which does not require an opening of a pre-existing plotter. Instead it (optionally)opens a pdf device, prints the plot to it, opens a png device, prints to that, and inserts the resulting png file. Thus it sacrifices some flexibility for a hopefully more consistent behaivor. In addition, one may use the pdfs as a set of images importable into illustrator or whatever.

### Value

A big honking excel file and a list including the dataframes and images created.

### See Also

**openxlsx Biobase** [normalize_expt](#) [graph_metrics](#)

### Examples

```
## Not run:
 excel_sucks <- write_expt(expt)

## End(Not run)
```

---

| | |
|---|---|
| write_go_xls | *Write gene ontology tables for excel* |

---

### Description

Combine the results from goseq, cluster profiler, topgo, and gostats and drop them into excel. Hopefully with a relatively consistent look.

### Usage

```
write_go_xls(
  goseq,
  cluster,
  topgo,
  gostats,
  gprofiler,
```

```
    file = "excel/merged_go",
    dated = TRUE,
    n = 30,
    overwritefile = TRUE
)
```

## Arguments

| | |
|---|---|
| `goseq` | The goseq result from simple_goseq() |
| `cluster` | The result from simple_clusterprofiler() |
| `topgo` | Guess |
| `gostats` | Yep, ditto |
| `gprofiler` | woo hoo! |
| `file` | the file to save the results. |
| `dated` | date the excel file |
| `n` | the number of ontology categories to include in each table. |
| `overwritefile` | overwrite an existing excel file |

## Value

the list of ontology information

## See Also

**openxlsx goseq clusterProfiler goStats topGO gProfiler**

---

| | |
|---|---|
| `write_goseq_data` | *Make a pretty table of goseq data in excel.* |

---

## Description

It is my intention to make a function like this for each ontology tool in my repetoire

## Usage

```
write_goseq_data(
  goseq_result,
  excel = "excel/goseq.xlsx",
  wb = NULL,
  add_trees = TRUE,
  order_by = "qvalue",
  pval = 0.1,
  add_plots = TRUE,
  height = 15,
  width = 10,
  decreasing = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `goseq_result` | A set of results from simple_goseq(). |
| `excel` | An excel file to which to write some pretty results. |
| `wb` | Workbook object to write to. |
| `add_trees` | Include topgoish ontology trees? |
| `order_by` | What column to order the data by? |
| `pval` | Choose a cutoff for reporting by p-value. |
| `add_plots` | Include some pvalue plots in the excel output? |
| `height` | Height of included plots. |
| `width` | and their width. |
| `decreasing` | In forward or reverse order? |
| `...` | Extra arguments are passed to arglist. |

## Value

The result from openxlsx in a prettyified xlsx file.

## See Also

**openxlsx goseq**

---

write_gostats_data    *Make a pretty table of gostats data in excel.*

---

## Description

It is my intention to make a function like this for each ontology tool in my repetoire

## Usage

```
write_gostats_data(
  gostats_result,
  excel = "excel/gostats.xlsx",
  wb = NULL,
  add_trees = TRUE,
  order_by = "qvalue",
  pval = 0.1,
  add_plots = TRUE,
  height = 15,
  width = 10,
  decreasing = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `gostats_result` | A set of results from simple_gostats(). |
| `excel` | An excel file to which to write some pretty results. |
| `wb` | Workbook object to write to. |
| `add_trees` | Include topgoish ontology trees? |
| `order_by` | Which column to order the data by? |
| `pval` | Choose a cutoff for reporting by p-value. |
| `add_plots` | Include some pvalue plots in the excel output? |
| `height` | Height of included plots. |
| `width` | and their width. |
| `decreasing` | Which order? |
| `...` | Extra arguments are passed to arglist. |

## Value

The result from openxlsx in a prettyified xlsx file.

## See Also

**openxlsx gostats**

---

`write_gprofiler_data`     *Write some excel results from a gprofiler search.*

---

## Description

Gprofiler is pretty awesome. This function will attempt to write its results to an excel file.

## Usage

```
write_gprofiler_data(
  gprofiler_result,
  wb = NULL,
  excel = "excel/gprofiler_result.xlsx",
  order_by = "recall",
  add_plots = TRUE,
  height = 15,
  width = 10,
  decreasing = FALSE,
  ...
)
```

## Arguments

gprofiler_result

        The result from simple_gprofiler().

| | |
|---|---|
| wb | Optional workbook object, if you wish to append to an existing workbook. |
| excel | Excel file to which to write. |
| order_by | Which column to order the data by? |
| add_plots | Add some pvalue plots? |
| height | Height of included plots? |
| width | And their width. |
| decreasing | Which order? |
| ... | More options, not currently used I think. |

## Value

A prettyified table in an xlsx document.

## See Also

**openxlsx gProfiler**

---

write_gsva                 *Write out my various attempts at making sense of gsva.*

---

## Description

While I am trying to make sense of gsva, I will use this function to write out the results I get so I can pass them to Najib/Maria Adelaida/Theresa to see if I am making sense.

## Usage

```
write_gsva(retlist, excel, plot_dim = 6)
```

## Arguments

| | |
|---|---|
| retlist | Result from running get_sig_gsva |
| excel | Excel file to write |
| plot_dim | Plot dimensions, likely needs adjustment. |

---

write_limma                    *Writes out the results of a limma search using write_de_table()*

---

### Description

Looking to provide a single interface for writing tables from limma and friends.

### Usage

```
write_limma(data, ...)
```

### Arguments

| | |
|---|---|
| data | Output from limma_pairwise() |
| ... | Options for writing the xlsx file. |

### See Also

[write_de_table](write_de_table)

### Examples

```
## Not run:
 finished_comparison = limma_pairwise(expressionset)
 data_list = write_limma(finished_comparison)

## End(Not run)
```

---

write_sample_design      *Put the metadata at the end of combined_de_tables()*

---

### Description

For the moment this is a stupidly short function. I am betting we will elaborate on this over time.

### Usage

```
write_sample_design(wb, apr)
```

### Arguments

| | |
|---|---|
| wb | workbook object. |
| apr | Pairwise result. |

---

write_sig_legend          *Internal function to write a legend for significant gene tables.*

---

### Description

Internal function to write a legend for significant gene tables.

### Usage

```
write_sig_legend(wb)
```

### Arguments

wb                xlsx workbook object from openxlsx.

---

write_subset_ontologies

*Write gene ontology tables for data subsets*

---

### Description

Given a set of ontology results, this attempts to write them to an excel workbook in a consistent and relatively easy-to-read fashion.

### Usage

```
write_subset_ontologies(
  kept_ontology,
  outfile = "excel/subset_go",
  dated = TRUE,
  n = NULL,
  overwritefile = TRUE,
  add_plots = TRUE,
  ...
)
```

### Arguments

kept_ontology   A result from subset_ontology_search()

outfile         Workbook to which to write.

dated           Append the year-month-day-hour to the workbook.

n               How many ontology categories to write for each search

overwritefile   Overwrite an existing workbook?

add_plots       Add the various p-value plots to the end of each sheet?

...             some extra parameters

**Value**

a set of excel sheet/coordinates

**See Also**

**openxlsx**

**Examples**

```
## Not run:
 all_contrasts <- all_pairwise(expt, model_batch=TRUE)
 keepers <- list(bob = ('numerator','denominator'))
 kept <- combine_de_tables(all_contrasts, keepers=keepers)
 changed <- extract_significant_genes(kept)
 kept_ontologies <- subset_ontology_search(changed, lengths=gene_lengths,
                                           goids=goids, gff=gff, gff_type='gene')
 go_writer <- write_subset_ontologies(kept_ontologies)

## End(Not run)
```

---

| write_suppa_table | *Take a set of results from suppa and attempt to write it to a pretty xlsx file.* |

---

**Description**

Suppa provides a tremendous amount of output, this attempts to standardize those results and print them to an excel sheet.

**Usage**

```
write_suppa_table(
  table,
  annotations = NULL,
  by_table = "gene_name",
  by_annot = "ensembl_gene_id",
  columns = "default",
  excel = "excel/suppa_table.xlsx"
)
```

**Arguments**

| | |
|---|---|
| table | Result table from suppa. |
| annotations | Set of annotation data to include with the suppa result. |
| by_table | Use this column to merge the annotations and data tables from the perspective of the data table. |

| | |
|---|---|
| by_annot | Use this column to merge the annotations and data tables from the perspective of the annotations. |
| columns | Choose a subset of columns to include, or leave the defaults. |
| excel | Provide an excel file to write. |

## Value

Data frame of the merged data.

## Examples

```
## Not run:
prettier_table <- write_suppa_table(suppa_result_file,
                                    annotations=gene_info,
                                    excel="excel/pretty_suppa_table.xlsx")

## End(Not run)
```

---

write_topgo_data          *Make a pretty table of topgo data in excel.*

---

## Description

It is my intention to make a function like this for each ontology tool in my repetoire

## Usage

```
write_topgo_data(
  topgo_result,
  excel = "excel/topgo.xlsx",
  wb = NULL,
  order_by = "fisher",
  decreasing = FALSE,
  pval = 0.1,
  add_plots = TRUE,
  height = 15,
  width = 10,
  ...
)
```

## Arguments

| | |
|---|---|
| topgo_result | A set of results from simple_topgo(). |
| excel | An excel file to which to write some pretty results. |
| wb | Workbook object to write to. |
| order_by | Which column to order the results by? |

| decreasing | In forward or reverse order? |
| pval | Choose a cutoff for reporting by p-value. |
| add_plots | Include some pvalue plots in the excel output? |
| height | Height of included plots. |
| width | and their width. |
| ... | Extra arguments are passed to arglist. |

## Value

The result from openxlsx in a prettyified xlsx file.

## See Also

**openxlsx topgo**

---

write_xlsx                          *Write a dataframe to an excel spreadsheet sheet.*

---

## Description

I like to give folks data in any format they prefer, even though I sort of hate excel. Most people I work with use it, so therefore I do too. This function has been through many iterations, first using XLConnect, then xlsx, and now openxlsx. Hopefully this will not change again.

## Usage

```
write_xlsx(
  data = "undef",
  wb = NULL,
  sheet = "first",
  excel = NULL,
  rownames = TRUE,
  start_row = 1,
  start_col = 1,
  title = NULL,
  ...
)
```

## Arguments

| data | Data frame to print. |
| wb | Workbook to which to write. |
| sheet | Name of the sheet to write. |
| excel | Filename of final excel workbook to write |
| rownames | Include row names in the output? |

| start_row | First row of the sheet to write. Useful if writing multiple tables. |
|---|---|
| start_col | First column to write. |
| title | Title for this xlsx table. |
| ... | Set of extra arguments given to openxlsx. |

## Value

List containing the sheet and workbook written as well as the bottom-right coordinates of the last row/column written to the worksheet.

## See Also

**openxlsx**

## Examples

```
 ## Not run:
  xls_coords <- write_xlsx(dataframe, sheet="hpgl_data", excel="testing.xlsx")
  xls_coords <- write_xlsx(another_df, wb=xls_coords$workbook,
                           sheet="hpgl_data", start_row=xls_coords$end_col)

## End(Not run)
```

---

| xlsx_plot_png | *An attempt to improve the behaivor of openxlsx's plot inserter.* |
|---|---|

---

## Description

The functions provided by openxlsx for adding plots to xlsx files are quite nice, but they can be a little annoying. This attempt to catch some corner cases and potentially save an extra svg-version of each plot inserted.

## Usage

```
xlsx_plot_png(
  a_plot,
  wb = NULL,
  sheet = 1,
  width = 6,
  height = 6,
  res = 90,
  plotname = "plot",
  savedir = "saved_plots",
  fancy_type = "pdf",
  start_row = 1,
  start_col = 1,
```

```
  file_type = "png",
  units = "in",
  ...
)
```

## Arguments

| | |
|---|---|
| `a_plot` | The plot provided |
| `wb` | Workbook to which to write. |
| `sheet` | Name or number of the sheet to which to add the plot. |
| `width` | Plot width in the sheet. |
| `height` | Plot height in the sheet. |
| `res` | Resolution of the png image inserted into the sheet. |
| `plotname` | Prefix of the pdf file created. |
| `savedir` | Directory to which to save pdf copies of the plots. |
| `fancy_type` | Plot publication quality images in this format. |
| `start_row` | Row on which to place the plot in the sheet. |
| `start_col` | Column on which to place the plot in the sheet. |
| `file_type` | Currently this only does pngs, but perhaps I will parameterize this. |
| `units` | Units for the png plotter. |
| `...` | Extra arguments are passed to arglist (Primarily for vennerable plots which are odd) |

## Value

A list containing the result of the tryCatch used to invoke the plot prints.

## See Also

**openxlsx**

## Examples

```
## Not run:
 fun_plot <- plot_pca(stuff)$plot
 df <- some_data_frame
 wb <- write_xlsx(df, excel="funkytown.xlsx")$workbook
 try_results <- xlsx_plot_png(fun_plot, wb=wb)

## End(Not run)
```

---

ymxb_print                          *Print a model as y = mx + b just like in grade school!*

---

### Description

Because, why not!?

### Usage

```
ymxb_print(lm_model, as = "glue")
```

### Arguments

lm_model        Model to print from glm/lm/robustbase.

as              Type to return.

### Value

a string representation of that model.

# Index