# Package ‘hpgltools’

June 19, 2017

**Type** Package

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

**Version** 2017.01

**Date** 2017-01-12

**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** acepack, ade4, affy, AnnotationDbi, AnnotationForge, AnnotationHub,
base64enc, Biobase, BiocGenerics, BiocInstaller, Biostrings,
biomaRt, bumphunter, Category, caTools, clusterProfiler, corpcor, corrplot,
DBI, DESeq2, DESeq, devtools, directlabels, dplyr, doParallel, DOSE, EDASeq,
edgeR, ffpe, fission, Formula, gdata, genbankr, genefilter,
genomeIntervals, GenomeInfoDb, GenomicFeatures, genoPlotR, GenomicRanges,
ggdendro, ggrepel, GO.db, googleVis, goseq, GOstats, gplots, graph,
gProfileR, GSEABase, gtools, gridExtra, hash, Heatplus, Hmisc, htmlTable,
igraph, inflection, IRanges, iterators, jsonlite, KEGGgraph, KEGGREST,
knitcitations, knitr, lattice, limma, matrixStats, motifRG, multtest,
mygene, openxlsx, OrganismDbi, pander, parallel, pasilla, pathview, plyr,
preprocessCore, qvalue, RamiGO, RColorBrewer, ReactomePA, readr, rentrez,
reshape2, RCurl, rGADEM, Rgraphviz, rmarkdown, RMySQL, robustbase,
RefManageR, reshape, rjson, robust, Rsamtools, rtracklayer, RUVSeq,
S4Vectors, scales, seqinr, seqLogo, statmod, stringi, stringr, survival,
sva, taxize, testthat, topGO, variancePartition, xtable, XVector, survJamda

**Imports** Biobase, data.table, knitr, ggplot2, magrittr, methods, foreach

**VignetteBuilder** knitr

**RoxygenNote** 6.0.1

## R topics documented:

---

`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, fc = 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.' |
| `fc` | 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,
  model_intercept = TRUE, extra_contrasts = NULL, alt_model = NULL,
  libsize = NULL, test_pca = TRUE, annot_df = NULL, parallel = 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 get_model_adjust(). |
| `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. |
| `...` | 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: 
  finished_comparison = eBayes(limma_output)
  data_list = all_pairwise(expt)

## 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? |

## `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, force = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `input` | Count table by sample. |
| `design` | Data frame of samples and conditions. |
| `force` | Force as input non-normalized data? |
| `...` | 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:
stupid_de <- basic_pairwise(expt)

## 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, design, batch = TRUE, batch1 = "batch",
  expt_state = NULL, batch2 = NULL, noscale = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `count_table` | Matrix of (pseudo)counts. |
| `design` | Model matrix defining the experimental conditions/batches/etc. |
| `batch` | String describing the method to try to remove the batch effect (or FALSE to leave it alone, TRUE uses limma). |
| `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! |

## 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 BiocInstaller 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 = "3.4", 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

**BiocInstaller**

## Examples

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

## End(Not run)
```

---

`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

```
biomart_orthologs(gene_ids, first_species = "hsapiens",
  second_species = "mmusculus", host = "dec2015.archive.ensembl.org",
  trymart = "ENSEMBL_MART_ENSEMBL", first_attributes = "ensembl_gene_id",
  second_attributes = c("ensembl_gene_id", "hgnc_symbol"))
```

## 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. |
| `first_attributes` | Key(s) of the first database to use. |
| `second_attributes` | Key(s) of the second database to use. |

## 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

Df of orthologs.

### See Also

**biomaRt** getLDS useMart

### Examples

```
## Not run:
 mouse_genes <- biomart_orthologs(some_ids)
 ## Hopefully the defaults are sufficient to translate from human to mouse.
yeast_genes <- biomart_orthologs(some_ids, first_species='mmusculus', second_species='scerevisiae')

## End(Not run)
```

---

## cbcb_batch_effect *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

### Usage

```
cbcb_batch_effect(normalized_counts, model)
```

### Arguments

| | |
|---|---|
| `normalized_counts` | Data frame of log2cpm counts. |
| `model` | Balanced experimental model containing condition and batch factors. |

### 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_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 = 2, min_samples = 2)
```

### Arguments

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

### 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)
```

## 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 analysis by edgeR or DESeq2.*

---

### Description

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

### Usage

```
choose_binom_dataset(input, 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, ...)
```

### 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`

## `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", ...)
```

### 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, batches, model_batch = TRUE,
  model_cond = TRUE, model_intercept = TRUE, alt_model = NULL,
  alt_string = NULL, intercept = 0, reverse = FALSE, surrogates = "be",
  ...)
```

### 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. |
| `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`

---

## `choose_orgdb` *Load the appropriate orgDb environment for a given species.*

---

### Description

Ok, so these are a bit more complex than I realized. The heirarchy as I now understand it (probably wrong) is that orgdb objects provide ID mappings among the various DBs. txdb objects provide the actual annotation information, and organismdbs acquire both (but only exist for a few species). Let's face it, I will never remember that the yeast orgdb is 'org.Sc.sgd.something'. This function is intended to make that process easier. Feed it a species name which makes sense: 'homo_sapiens' and it will assume you mean orgdb.whatever and load that into your environment. This should also make a reasonable attempt at installing the appropriate orgdb if it is not already in your R library tree.

### Usage

```
choose_orgdb(species = "saccharomyces_cerevisiae")
```

### Arguments

| | |
|---|---|
| `species` | Human readable species name |

### Value

orgdb object for the relevant species, or an error if I don't have a mapping for it.

### See Also

**AnnotationDbi** `keytypes`

### Examples

```
## Not run:
 object <- choose_orgdb("homo_sapiens")

## End(Not run)
```

---

## choose_txdb *Load the appropriate TxDb environment for a given species.*

---

### Description

Ok, so these are a bit more complex than I realized. The heirarchy as I now understand it (probably wrong) is that orgdb objects provide ID mappings among the various DBs. txdb objects provide the actual annotation information, and organismdbs acquire both (but only exist for a few species). Let's face it, I will never remember that the yeast orgdb is 'org.Sc.sgd.something'. This function is intended to make that process easier. Feed it a species name which makes sense: 'homo_sapiens' and it will assume you mean orgdb.whatever and load that into your environment. This should also make a reasonable attempt at installing the appropriate orgdb if it is not already in your R library tree.

### Usage

```
choose_txdb(species = "saccharomyces_cerevisiae")
```

### Arguments

| | |
|---|---|
| `species` | Human readable species name |

### Value

orgdb object for the relevant species, or an error if I don't have a mapping for it.

### See Also

**AnnotationDbi** `keytypes`

### Examples

```
## Not run:
 object <- choose_txdb("homo_sapiens")

## 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(df, cfgout = "circos/conf/default.conf", first_col = "chr1",
  second_col = "chr2", color = "blue", radius = 0.75, thickness = 3)
```

### Arguments

| | |
|---|---|
| `df` | Dataframe with starts/ends and the floating point information. |
| `cfgout` | Master configuration file to write. |
| `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. |

### 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(df, annot_df, cfgout = "circos/conf/default.conf",
  colname = "logFC", chr = "chr1", colors = NULL, outer = 0.9,
  width = 0.08, spacing = 0)
```

### Arguments

| | |
|---|---|
| `df` | Dataframe with starts/ends and the floating point information. |
| `annot_df` | Annotation data frame with starts/ends. |
| `cfgout` | Master configuration file to write. |
| `colname` | Name of the column with the data of interest. |
| `chr` | Name of the chromosome (This currently assumes a bacterial chromosome). |
| `colors` | Colors of the heat map. |
| `outer` | Floating point radius of the circle into which to place the heatmap. |
| `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(df, annot_df, cfgout = "circos/conf/default.conf",
  colname = "logFC", chr = "chr1", color = "blue", fill_color = "blue",
  outer = 0.9, width = 0.08, spacing = 0)
```

## Arguments

| | |
|---|---|
| df | Dataframe with starts/ends and the floating point information. |
| annot_df | Annotation data frame containing starts/ends. |
| cfgout | Master configuration file to write. |
| colname | Name of the column with the data of interest. |
| chr | Name of the chromosome (This currently assumes a bacterial chromosome). |
| color | Color of the plotted data. |
| fill_color | Guess! |
| 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)
```

## 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? |

## 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(name = "default", conf_dir = "circos/conf",
  length = NULL, chr_name = "chr1", segments = 6, color = "white",
  chr_num = 1, fasta = NULL)
```

## Arguments

| | |
|---|---|
| `name` | Name of the chromosome (This currently assumes a bacterial chromosome). |
| `conf_dir` | Where to put the circos configuration file(s). |
| `length` | Length of the chromosome (the default is mgas5005). |
| `chr_name` | Short name of the chromosome. |
| `segments` | How many segments to cut the chromosome into? |
| `color` | Color segments of the chromosomal arc? |
| `chr_num` | Number to record for each chromosome. |
| `fasta` | Fasta file to use to create the karyotype. |

## 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(target = "", output = "circos/Makefile", circos = "circos")
```

## Arguments

| | |
|---|---|
| `target` | Default make target. |
| `output` | Makefile to write. |
| `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(go_table, cfgout = "circos/conf/default.conf",
  chr = "chr1", outer = 1, width = 0.08, spacing = 0)
```

## Arguments

| | |
|---|---|
| `go_table` | Dataframe with starts/ends and categories. |
| `cfgout` | Master configuration file to write. |
| `chr` | Name of the chromosome. |
| `outer` | Floating point radius of the circle into which to place the plus-strand data. |
| `width` | Radial width of each tile. |
| `spacing` | Radial distance between outer, inner, and inner to whatever follows. |

## 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(name = "mgas", conf_dir = "circos/conf", radius = 1800,
  band_url = NULL)
```

### Arguments

| | |
|---|---|
| `name` | Name of the map, called with 'make name'. |
| `conf_dir` | Directory containing the circos configuration data. |
| `radius` | Size of the image. |
| `band_url` | Place to imagemap link. |

### 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(cfgout = "circos/conf/default.conf")
```

### Arguments

| | |
|---|---|
| `cfgout` | Master configuration file to write. |

## Value

The filename of the configuration.

---

`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(df, annot_df, cfgout = "circos/conf/default.conf",
  colname = "logFC", chr = "chr1", colors = NULL, outer = 0.9,
  width = 0.08, spacing = 0)
```

## Arguments

| | |
|---|---|
| `df` | Dataframe with starts/ends and the floating point information. |
| `annot_df` | Annotation data frame defining starts/stops. |
| `cfgout` | Master configuration file to write. |
| `colname` | Name of the column with the data of interest. |
| `chr` | Name of the chromosome (This currently assumes a bacterial chromosome) |
| `colors` | Colors of the data. |
| `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.

## 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", goids_df = 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. |
| `goids_df` | 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_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_de_table(li, ed, de, ba, table_name, annot_df = NULL,
  inverse = FALSE, adjp = TRUE, padj_type = "fdr", include_deseq = TRUE,
  include_edger = TRUE, include_limma = TRUE, include_basic = TRUE,
  fc_cutoff = 1, p_cutoff = 0.05, excludes = NULL)
```

### Arguments

| | |
|---|---|
| `li` | Limma output table. |
| `ed` | Edger output table. |
| `de` | Deseq2 output table. |
| `ba` | Basic output table. |
| `table_name` | Name of the table to merge. |
| `annot_df` | Add some annotation information? |
| `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_limma` | Include tables from limma? |
| `include_basic` | Include the basic table? |
| `fc_cutoff` | Preferred logfoldchange cutoff. |
| `p_cutoff` | Preferred pvalue cutoff. |
| `excludes` | Set of genes to exclude from the output. |

### 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**

## 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(all_pairwise_result, extra_annot = NULL, 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_basic = TRUE,
  rownames = TRUE, add_plots = TRUE, loess = FALSE, plot_dim = 6,
  compare_plots = TRUE, padj_type = "fdr")
```

### Arguments

| | |
|---|---|
| `all_pairwise_result` | Output from all_pairwise(). |
| `extra_annot` | Add some annotation information? |
| `excel` | Filename for the excel workbook, or null if not printed. |
| `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_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` | In an attempt to save memory when printing to excel, make it possible to |
| `padj_type` | Add a consistent p adjustment of this type. exclude comparison plots in the summary sheet. |

### Value

Table combining limma/edger/deseq outputs.

### See Also

`all_pairwise`

### Examples

```
## Not run:
 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)
```

---

## `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`

## `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, 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. |
| `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. |

### Value

List of the results.

### See Also

`get_model_adjust`

---

## `compare_tables` *See how similar are results from limma/deseq/edger.*

---

### 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

```
compare_tables(limma = NULL, deseq = NULL, edger = NULL, basic = NULL,
  include_basic = TRUE, annot_df = NULL, ...)
```

### Arguments

| | |
|---|---|
| `limma` | Data from limma_pairwise(). |
| `deseq` | Data from deseq2_pairwise(). |
| `edger` | Data from edger_pairwise(). |
| `basic` | Data from basic_pairwise(). |
| `include_basic` | include the basic data? |
| `annot_df` | Include annotation data? |
| `...` | More options! |

### Details

Invoked by all_pairwise().

### Value

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

### See Also

`limma_pairwise` `edger_pairwise` `deseq2_pairwise`

### Examples

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

## End(Not run)
```

## `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(data, convert = "raw", ...)
```

## Arguments

| | |
|---|---|
| data | Matrix of count data. |
| convert | 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). |

## Value

Dataframe of cpm/rpkm/whatever(counts)

## See Also

**edgeR Biobase** cpm

## Examples

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

## End(Not run)
```

## `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, design = NULL)
```

### 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. |

### Value

A data frame of adjusted counts.

### See Also

**Biobase**

## `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.

---

`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 other extraneous information.*

---

## Description

It is worth noting that this function has a lot of logic used to find the count tables in the local filesystem. This logic has been superceded by simply adding a field to the .csv file called 'file'. create_expt() will then just read that filename, it may be a full pathname or local to the cwd of the project.

## Usage

```
create_expt(metadata, gene_info = NULL, count_dataframe = NULL,
  sample_colors = NULL, title = NULL, notes = NULL,
  include_type = "all", include_gff = NULL, savefile = "expt",
  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. |

| | |
|---|---|
| `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? |
| `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. |
| `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` `expt_read_counts` `as.list.hash`

### Examples

```
## Not run:
 new_experiment = create_expt("some_csv_file.csv", color_hash)
 ## Remember that this depends on an existing data structure of gene annotations.

## 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`

## 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). somefunction 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)
```

## 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? DESeq seems to not be a fan of them. |
| `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

Tested in test_24de_deseq.R 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)
```

---

## `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`

## 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, euler = FALSE, p = 0.05, fc = 0, ...)
```

### Arguments

| | |
|---|---|
| `table` | Which table to query? |
| `adjp` | Use adjusted p-values |
| `euler` | Perform a euler plot |
| `p` | p-value cutoff, I forget what for right now. |
| `fc` | 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)
```

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

### Description

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)

### Usage

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

### Arguments

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

## `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)
```

---

`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.

## See Also

**ape**

## Examples

```
## Not run:
 gbk_file <- download_gbk(accessions=c("AE009949","AE009948"))

## 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 |
| `...` | The set of arguments intended for limma_pairwise(), edger_pairwise(), and friends. |

### Details

Used to make parallel operations easier.

### Value

The result from limma/deseq/edger/basic

### See Also

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

## 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 = TRUE,
  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 cell means or intercept? |
| `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**

Tested in test_26de_edger.R 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:
 pretend = edger_pairwise(data, conditions, batches)

## End(Not run)
```

---

`expt_exclude_genes` *Exclude some genes given a pattern match*

---

## Description

Because I am too lazy to remember that expressionsets use matrix subsets for [gene,sample]

## Usage

```
expt_exclude_genes(expt, column = "txtype", method = "remove",
  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. |
| `patterns` | Character list of patterns to remove/keep |
| `...` | Extra arguments are passed to arglist, currently unused. |

## Value

A smaller expt

## See Also

`create_expt`

| | |
|---|---|
| `expt_read_counts` | *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

```
expt_read_counts(ids, files, header = FALSE, include_summary_rows = FALSE,
  suffix = 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. |
| `...` | 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)
```

## expt_snp — *Gather snp information for an expt*

### Description

I have some initial code for working with snps, but it seems that it will be getting more use, so make it testable etc.

### Usage

```
expt_snp(expt, input_dir = "preprocessing/outputs",
  file_suffix = "_parsed_ratio.txt", bam_suffix = "_accepted_paired.bam",
  tolower = TRUE)
```

### Arguments

| | |
|---|---|
| `expt` | an expressionset from which to extract information. |
| `input_dir` | Directory to scan for snps output files. |
| `file_suffix` | What to add on the end of the files for the resulting output. |
| `bam_suffix` | How do we find the bam files? |
| `tolower` | Lowercase stuff like 'HPGL'? |

### Value

some stuff

## expt_subset — *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

```
expt_subset(expt, subset = NULL)
```

### Arguments

| | |
|---|---|
| `expt` | Expt chosen to extract a subset of data. |
| `subset` | Valid R expression which defines a subset of the design to keep. |

## 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)
```

---

`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 = 100, 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. |

## 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, fc = NULL, n = NULL,
  loess = FALSE, color_low = "#DD0000", 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). |
| `fc` | 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.) |
| `color_low` | Color for the genes less than the mean. |
| `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:
 scatter_plot <- extract_coefficient_scatter(pairwise_output, type="deseq", x="uninfected", y="infected")

## End(Not run)
```

---

`extract_de_ma` *Make a MA plot of some limma output with pretty colors and shapes*

---

## Description

Yay pretty colors and shapes!

## Usage

```
extract_de_ma(pairwise, type = "edger", table = NULL, logfc = 1,
  pval_cutoff = 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. |
| `pval_cutoff` | 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

## 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_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_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", fc = 1,
  p = 0.05, sig_bar = TRUE, z = NULL, n = NULL, ma = TRUE,
  p_type = "adj", invert_barplots = FALSE,
  excel = "excel/significant_genes.xlsx", 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. |
| `fc` | 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. |
| `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.' |

### Value

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

### See Also

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(svd_v, fact, type = "factor")
```

### Arguments

| | |
|---|---|
| `svd_v` | V' V = I portion of a fast.svd call. |
| `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)
```

### Arguments

| | |
|---|---|
| `data` | A dataframe/exprs/matrix/whatever of counts. |
| `cutoff` | Minimum number of counts. |
| `hard` | Greater-than is hard, greater-than-equals is not. |

### 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

Number of genes.

### See Also

**Biobase**

### Examples

```
## Not run:
 features <- features_greater_than(expt)

## End(Not run)
```

---

## 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, filter = "cbcb", p = 0.01, A = 1, k = 1,
  cv_min = 0.01, cv_max = 1000, thresh = 4, min_samples = 2, ...)
```

### Arguments

| | |
|---|---|
| count_table | Some counts to filter. |
| filter | 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)
```

---

`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_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_goseq_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_goseq_genes(goseq, ontology = NULL, pval = 0.1,
  include_all = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `goseq` | List of goseq specific results as generated by simple_goseq(). |
| `ontology` | Ontology to search (MF/BP/CC). |
| `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)
```

---

## `gbk2txdb` *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

```
gbk2txdb(accession = "AE009949", savetxdb = FALSE)
```

### Arguments

| | |
|---|---|
| `accession` | Accession to download and import |
| `savetxdb` | Save a txdb package from this? FIXME THIS DOES NOT WORK. |

### Details

Tested in test_40ann_biomartgenbank.R and test_70expt_spyogenes.R This just sets some defaults for the genbankr service in order to facilitate downloading genomes and such 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`

## Examples

```
## Not run:
 txdb_result <- gbk2txdb(accession="AE009948", savetxdb=TRUE)

## End(Not run)
```

---

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

---

## Description

Maybe this should get pulled into the previous function?

## 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.

## Value

Granges data

## See Also

**AnnotationDbi** **GenomeInfoDb** **GenomicFeatures** `select`

## Examples

```
## Not run:
 annotations <- gbk_annotations("saureus_txdb")

## End(Not run)
```

## `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_gene_kegg_mapping`
*Generate GENE/KEGG mapping.*

---

## Description

This uses KEGGREST and related function kegg_to_ensembl() to associate genes to KEGG pathways.

## Usage

```
generate_gene_kegg_mapping(pathways, org_abbreviation, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `pathways` | Vector of KEGG pathway IDs returned from call to keggLink() e.g. "path:mmu05134". |
| `org_abbreviation` | KEGG identifier for the species of interest (e.g. "hsa" for Homo sapiens). |
| `verbose` | talky talky? |

## Value

Df mapping kegg and gene IDs.

## See Also

**KEGGREST** `keggLink`

## Examples

```
## Not run:
 kegg_df <- generate_gene_kegg_mapping(path, org)

## End(Not run)
```

## `generate_kegg_pathway_mapping` *Generate a KEGG PATHWAY / description mapping.*

### Description

Make an easier to use df of KEGG -> descriptions using keggGet.

### Usage

```
generate_kegg_pathway_mapping(pathways, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `pathways` | Vector of KEGG pathway identifiers. |
| `verbose` | talk talk? |

### Value

Data frame describing some kegg pathways

### See Also

**KEGGREST** `keggLink`

### Examples

```
## Not run:
 mapping <- generate_kegg_pathway_mapping(c("hsa00040", "hsa00100"))

## End(Not run)
```

## `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**

## `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!

## 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**

## get_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

```
get_biomart_annotations(species = "hsapiens", overwrite = FALSE,
  do_save = TRUE, host = "dec2015.archive.ensembl.org",
  trymart = "ENSEMBL_MART_ENSEMBL", gene_requests = c("ensembl_gene_id",
  "ensembl_transcript_id", "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... |
| `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

Df of some (by default) human annotations.

## See Also

**biomaRt** `listDatasets` `getBM`

## Examples

```
## Not run:
 tt = get_biomart_annotations()

## End(Not run)
```

## get_biomart_ontologies

*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

```
get_biomart_ontologies(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

Df of geneIDs and GOIDs.

### See Also

**biomaRt** `listMarts` `useDataset` `getBM`

### Examples

```
## Not run:
 tt = get_biomart_ontologies()

## End(Not run)
```

---

## get_eupath_config *Grab some configuration data collated and used to make OrganismDbi/OrgDb/TxDb objects.*

---

### Description

This function uses some data copied into inst/ to decide some parameters used for generating the various packages generated here.

### Usage

```
get_eupath_config(cfg = NULL)
```

### Arguments

| | |
|---|---|
| `cfg` | Optional data frame |

### Details

Tested in test_46ann_tritrypdb.R This function is sort of stupid and perhaps will be removed. I keep a small csv file of some TriTrypDB specific metadata, things like data base version number, URL schemes, etc. This reads that and extracts the relevant information.

### Value

Dataframe of configuration data, a few columns are required, run it with no args to see which ones.

---

## get_genelengths *Grab gene lengths from a gff file.*

---

### Description

This function attempts to be robust to the differences in output from importing gff2/gff3 files. But it certainly isn't perfect.

### Usage

```
get_genelengths(gff, type = "gene", key = "ID", ...)
```

### Arguments

| | |
|---|---|
| `gff` | Gff file with (hopefully) IDs and widths. |
| `type` | Annotation type to use (3rd column). |
| `key` | Identifier in the 10th column of the gff file to use. |
| `...` | Extra arguments likely for gff2df |

### Value

Data frame of gene IDs and widths.

### See Also

**rtracklayer** `gff2df`

### Examples

```
## Not run:
 tt = get_genelengths('reference/fun.gff.gz')
 head(tt)
##           ID width
## 1   YAL069W   312
## 2   YAL069W   315
## 3   YAL069W     3
## 4 YAL068W-A   252
## 5 YAL068W-A   255
## 6 YAL068W-A     3

## End(Not run)
```

---

## `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_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_loci_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

```
get_loci_go(taxonid = "160490")
```

### Arguments

| | |
|---|---|
| `taxonid` | Which species to query. |

### Details

Tested in test_42ann_microbes.R I am not 100 At the very least, it does return a large number of them, which is a start.

### Value

data frame of GO terms from pub.microbesonline.org

### See Also

**DBI** `dbSendQuery` `fetch`

### Examples

```
## Not run:
 go_df <- get_loci_go(taxonid="160490")

## End(Not run)
```

## get_microbesonline_annotation *Skip the db and download all the text annotations for a given species.*

### Description

Like I said, the microbesonline mysqldb is rather more complex than I prefer. This shortcuts that process and just grabs a tsv copy of everything and loads it into a dataframe.

### Usage

```
get_microbesonline_annotation(ids = "160490", species = NULL)
```

### Arguments

| | |
|---|---|
| `ids` | List of ids to query. |
| `species` | Species name(s) to use instead. |

### 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

List of dataframes with the annotation information.

### See Also

**RCurl** `getURL`

### Examples

```
## Not run:
 annotations <- get_microbesonline_annotation(ids=c("160490","160491"))

## End(Not run)
```

---

## `get_microbesonline_ids` *Use the publicly available microbesonline mysql instance to get species ids.*

---

### Description

The microbesonline mysql instance is more complex than I like. Their id system is reminiscent of KEGG's and similarly annoying. Though I haven't figured out how the tables interact, a query to get ids is simple enough.

### Usage

```
get_microbesonline_ids(name = "Escherichia", exact = FALSE)
```

### Arguments

| | |
|---|---|
| `name` | Text string containing some part of the species name of interest. |
| `exact` | Use an exact species name? |

### Details

Tested in test_42ann_microbes.R This function sets the defaults required for getting a quick and dirty connection to the public microbesonline database and returning the ids associated with a given name.

### Value

Dataframe of ids and names.

### See Also

**DBI** dbSendQuery fetch

### Examples

```
## Not run:
 microbes_ids <- get_microbesonline_ids(name="Streptococcus")

## End(Not run)
```

---

## get_microbesonline_name

*Use the publicly available microbesonline mysql instance to get species name(s).*

---

### Description

The microbesonline mysql instance is more complex than I like. Their id system is reminiscent of KEGG's and similarly annoying. Though I haven't figured out how the tables interact, a query to get ids is simple enough.

### Usage

```
get_microbesonline_name(id = 316385)
```

### Arguments

| | |
|---|---|
| id | Text string containing some part of the species name of interest. |

### Details

Tested in test_42ann_microbesonline.R This is essentially covered in get_micrboesonline_ids(), but this works too.

### Value

Dataframe of ids and names.

## See Also

**DBI** dbSendQuery fetch

## Examples

```
## Not run:
 names <- get_microbesonline_name(id=316385)

## End(Not run)
```

---

`get_model_adjust` *Extract some surrogate estimations from a raw data set using sva, ruv, and/or pca.*

---

## Description

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.

## Usage

```
get_model_adjust(input, design = NULL, estimate_type = "sva",
  surrogates = "be", expt_state = NULL, ...)
```

## Arguments

| | |
|---|---|
| `input` | Expt or data frame to manipulate. |
| `design` | If the data is not an expt, provide experimental design here. |
| `estimate_type` | One of: sva_supervised, sva_unsupervised, ruv_empirical, ruv_supervised, ruv_residuals, or pca. |
| `surrogates` | Choose a method for getting the number of surrogates, be or leek, or a number. |
| `expt_state` | Current state of the expt object (to check for log2, cpm, etc) |
| `...` | Parameters fed to arglist. |

## Value

List including the adjustments for a model matrix, a modified count table, and 3 plots of the known batch, surrogates, and batch/surrogate.

## See Also

**Biobase sva EDASeq RUVseq edgeR**

## get_ncbi_taxonid *Use taxize to get ncbi taxon IDs*

### Description

taxize looks like it might be awesome, but it is also pretty annoying

### Usage

```
get_ncbi_taxonid(species = "Leishmania major")
```

### Arguments

| | |
|---|---|
| species | Human readable species name |

### Value

potential NCBI taxon IDs

### See Also

**taxize**

### Examples

```
## Not run: 
 taxonid <- get_ncbi_taxonid(species="Trypanosoma cruzi")

## End(Not run)
```

## 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**

---

## `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, fc = 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. |
| `fc` | 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`

gff2df *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

```
gff2df(gff, type = NULL, id_col = "ID", second_id_col = "locus_tag",
  try = 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. |
| second_id_col | Second column to check. |
| try | Give your own function call to use for importing. |

## Value

Dataframe of the annotation information found in the gff file.

## See Also

**rtracklayer** **GenomicRanges** import.gff

## Examples

```
## Not run:
 funkytown <- gff2df('reference/gff/saccharomyces_cerevsiae.gff.xz')

## End(Not run)
```

## 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 gff2df(), but returns data suitable for getSet()

### Value

Iranges! (useful for getSeq().)

### See Also

**rtracklayer** gff2df **Biostrings** `import.gff`

### Examples

```
## Not run:
 library(BSgenome.Tcruzi.clbrener.all)
 tc_clb_all <- BSgenome.Tcruzi.clbrener.all
 cds_ranges <- gff2irange('reference/gff/tcruzi_clbrener.gff.xz', type='CDS')
 cds_sequences <- Biostrings::getSeq(tc_clb_all, cds_ranges)

## End(Not run)
```

## ggplot2_heatmap *Taken from https://plot.ly/ggplot2/ggdendro-dendrograms/*

### Description

Check out the following link for a neat dendrogram library. http://www.sthda.com/english/wiki/beautiful-dendrogram-visualizations-in-r-5-must-known-methods-unsupervised-machine-learning

### Usage

```
ggplot2_heatmap(some_df)
```

### Arguments

| | |
|---|---|
| `some_df` | A data frame to heatmap using ggplot2. |

### Value

putatively a heatmap!

## 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 oriented with the
 ## > 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 clusterProfiler.*

## 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(de_genes, mf_over, bp_over, cc_over, mf_under, bp_under, cc_under,
  goid_map = "id2go.map", score_limit = 0.01, goids_df = NULL,
  overwrite = FALSE, selector = "topDiffGenes", pval_column = "adj.P.Val")
```

### Arguments

| | |
|---|---|
| `de_genes` | Some differentially expressed genes. |
| `mf_over` | Mfover data. |
| `bp_over` | Bpover data. |
| `cc_over` | Ccover data. |
| `mf_under` | Mfunder data. |
| `bp_under` | Bpunder data. |
| `cc_under` | Ccunder expression data. |
| `goid_map` | Mapping of IDs to GO in the Ramigo expected format. |
| `score_limit` | Maximum score to include as 'significant'. |
| `goids_df` | Dataframe of available goids (used to generate goid_map). |
| `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 = NULL, ma = NULL, ...)
```

### 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 |
| `...` | 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)
 ## good luck, you are going to be waiting a while for the ma plots to print!

## End(Not run)
```

---

`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 = 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`

## 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 inaction, check out the vignettes: `browseVignettes(package = 'hpgltools')`

## 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 |

### Details

Note that I did this two months ago and haven't touched it since...

### 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_combatMod *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

```
hpgl_combatMod(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 = hpgl_combatMod(df, batches, model)

## 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().

### 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_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_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(df, ...)
```

### Arguments

| | |
|---|---|
| df | 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)
```

## kegg_get_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

```
kegg_get_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 = kegg_get_orgn('Canis')
 ## >     Tid     orgid      species                   phylogeny
 ## >  17 T01007   cfa Canis familiaris (dog) Eukaryotes;Animals;Vertebrates;Mammals

## End(Not run)
```

## kegg_to_ensembl *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

```
kegg_to_ensembl(kegg_ids)
```

### Arguments

| | |
|---|---|
| `kegg_ids` | List of KEGG identifiers to be mapped. |

### Value

Ensembl IDs as a character list.

### See Also

**KEGGREST** `keggGet`

### Examples

```
## Not run:
ensembl_list <- kegg_to_ensembl("a")

## End(Not run)
```

## kmeans_testing — *This is for the moment just a code dump of some arbitrarily chosen kmeans clustering stuff*

### Description

Fill this in asap with real code for Ginger's search of gene sets which have similar profiles over time.

### Usage

```
kmeans_testing(gene_ids = get0("gene_ids"))
```

### Arguments

| | |
|---|---|
| `gene_ids` | A set of gene IDs to query. |

| | |
|---|---|
| `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 = TRUE,
  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 makeContrasts() 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)
```

---

## `limma_scatter` *Plot arbitrary data from limma as a scatter plot.*

---

### Description

Extract the adjusted abundances for the two conditions used in the pairw

### Usage

```
limma_scatter(all_pairwise_result, first_table = 1, first_column = "logFC",
  second_table = 2, second_column = "logFC", type = "linear_scatter", ...)
```

### Arguments

| | |
|---|---|
| `all_pairwise_result` | Result from calling balanced_pairwise(). |
| `first_table` | First table from all_pairwise_result$limma_result to look at (may be a name or number). |
| `first_column` | Name of the column to plot from the first table. |
| `second_table` | Second table inside all_pairwise_result$limma_result (name or number). |
| `second_column` | Column to compare against. |
| `type` | Type of scatter plot (linear model, distance, vanilla). |
| `...` | Use the elipsis to feed options to the html graphs. |

### Value

plot_linear_scatter() set of plots comparing the chosen columns. If you forget to specify tables to compare, it will try the first vs the second.

### See Also

**limma** `plot_linear_scatter`

### Examples

```
## Not run:
 compare_logFC <- limma_scatter(all_pairwise, first_table="wild_type", second_column="mutant",
                                first_table="AveExpr", second_column="AveExpr")
 compare_B <- limma_scatter(all_pairwise, first_column="B", second_column="B")

## End(Not run)
```

---

## `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)
```

## load_host_annotations *Load organism annotation data (mouse/human).*

### Description

Creates a dataframe gene and transcript information for a given set of gene ids using the OrganismDbi interface.

### Usage

```
load_host_annotations(orgdb, gene_ids = NULL, keytype = "ENSEMBL",
  fields = c("TXCHROM", "GENENAME", "TXSTRAND", "TXSTART", "TXEND"),
  biomart_dataset = "hsapiens_gene_ensembl")
```

### Arguments

| | |
|---|---|
| `orgdb` | OrganismDb instance. |
| `gene_ids` | Gene identifiers for retrieving annotations. |
| `keytype` | a, umm keytype? I need to properly read this code. |
| `fields` | Columns to include in the output. |
| `biomart_dataset` | Name of the biomaRt dataset to query for gene type. |

### Value

a table of gene information

### See Also

**AnnotationDbi dplyr biomaRt** `select` `keytypes`

### Examples

```
## Not run:
 host <- load_host_annotations(org, c("a","b"))

## End(Not run)
```

## `load_kegg_pathways` *Creates a KEGG pathway/description mapping dataframe.*

### Description

Use AnnotationDbi to map descriptions of KEGG pathways to gene IDs.

### Usage

```
load_kegg_pathways(orgdb, gene_ids, keytype = "ENSEMBL")
```

### Arguments

| | |
|---|---|
| `orgdb` | OrganismDb instance. |
| `gene_ids` | Identifiers of the genes to retrieve annotations. |
| `keytype` | as per the previous functions, I don't know what this does yet |

### Value

Character list of pathways.

### See Also

**AnnotationDbi**

### Examples

```
## Not run:
 pathnames <- load_kegg_pathways(org, c("a","b","c"))

## End(Not run)
```

## `load_orgdb_annotations` *Load organism annotation data (parasite).*

### Description

Creates a dataframe gene and transcript information for a given set of gene ids using the OrganismDbi interface.

### Usage

```
load_orgdb_annotations(orgdb, gene_ids = NULL, include_go = FALSE,
  keytype = "ENSEMBL", fields = NULL, sum_exons = FALSE)
```

### Arguments

| | |
|---|---|
| orgdb | OrganismDb instance. |
| gene_ids | Gene identifiers for retrieving annotations. |
| include_go | Ask the Dbi for gene ontology information? |
| keytype | mmm the key type used? |
| fields | Columns included in the output. |
| sum_exons | 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 can pass it 'all'.

### Value

Table of geneids, chromosomes, descriptions, strands, types, and lengths.

### See Also

**AnnotationDbi GenomicFeatures BiocGenerics** `columns` `keytypes` `select` `exonsBy`

### Examples

```
## Not run:
 one_gene <- load_annotations(org, c("LmJF.01.0010"))

## End(Not run)
```

---

## 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, 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.

### See Also

**AnnotationDbi GO.db magrittr** `select` `tbl_df`

### Examples

```
## Not run:
 go_terms <- load_go_terms(org, c("a","b"))

## End(Not run)
```

---

## `load_orgdb_kegg` *Creates a gene/KEGG mapping dataframe.*

---

### Description

In much the same way AnnotationDbi provides GO data, it also provides KEGG data.

### Usage

```
load_orgdb_kegg(orgdb, gene_ids = NULL, keytype = "ENSEMBL",
  columns = c("KEGG_PATH"))
```

### Arguments

| | |
|---|---|
| `orgdb` | OrganismDb instance. |
| `gene_ids` | Identifiers of the genes to retrieve annotations. |
| `keytype` | The keytype, eg. the primary key used to query the orgdb. |
| `columns` | Columns to extract. |

### Details

Tested in test_45ann_organdb.R Perhaps this function should be merged with the GO above?

### Value

Df of kegg mappings

### See Also

**AnnotationDbi** **dplyr** `select` `tbl_df`

### Examples

```
## Not run:
 kegg_data <- load_kegg_mapping(org, c("a","b"))

## End(Not run)
```

---

`load_parasite_annotations`
*I see no reason to have load_host_annotations and load_parasite_annotations.*

---

### Description

Thus I am making them both into aliases to load_annotations.

### Usage

```
load_parasite_annotations(...)
```

### Arguments

| | |
|---|---|
| `...` | Arguments to be passed to load_annotations. |

---

`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_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", goids_df = NULL,
  overwrite = FALSE)
```

### Arguments

| | |
|---|---|
| `goid_map` | TopGO mapping file. |
| `goids_df` | 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 for them 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(data, adjust = "BH", n = 0, coef = NULL,
  annot_df = NULL)
```

### Arguments

| | |
|---|---|
| `data` | Output from 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. |

### 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_xls` `topTable`

## Examples

```
## Not run:
 finished_comparison = eBayes(limma_output)
 table = make_limma_tables(finished_comparison, adjust="fdr")

## End(Not run)
```

---

`make_organ` *Create an organismDbi object by joining a txdb and orgdb together.*

---

## Description

This function is a bit more fragile than I would like.

## Usage

```
make_organ(txdb, keytype = NA, orgdb = NA)
```

## Arguments

| | |
|---|---|
| `txdb` | Txdb input to merge |
| `keytype` | When merging to an orgdb, what key to use? |
| `orgdb` | The orgdb to help create the OrganismDbi instance. |

## Value

An OrganismDb instance

## See Also

**S4Vectors GenomicFeatures AnnotationDbi OrganismDbi** `makePackageName`

## Examples

```
## Not run:
 orgdbi <- make_organ(Tcruzi_txdb, orgdb=Tcruzi_orgdb)

## End(Not run)
```

## make_organismdbi *Create an OrganismDbi for a species at the TriTrypDb*

### Description

OrganismDbi instances are pretty neat, they pull together OrgDb and TxDb. With any luck, this function provides the ability to pull together all the data from the TriTrypDb, GO.db, and KEGGREST in order to accomplish these peculiar tasks.

### Usage

```
make_organismdbi(id = "lmajor_friedlin", cfg = NULL,
  output_dir = "organdb/tritryp", ...)
```

### Arguments

| | |
|---|---|
| `id` | Unique tritrypdb identifier. |
| `cfg` | A configuration dataframe, when null it will be replaced by reading a csv file in inst/extdata. |
| `output_dir` | The directory into which to put the various intermediate files, including downloads from the TriTrypdb, the created OrgDb and TxDb instances, and the final OrganismDbi. |
| `...` | Extra arguments including a boolean for whether to include kegg. |

### Value

A path, some data files, and a kitty!

### See Also

**AnnotationForge OrganismDbi**

### Examples

```
## Not run:
 crazytown <- make_organismdbi()  ## wait a loong time

## End(Not run)
```

## make_orgdb — *Make an orgDb object from some information provided by make_orgdb_info()*

### Description

An orgDb object should provide some useful annotation data including fun stuff like gene ontology, kegg, etc. In the case of the species at the TriTrypDb, much of this information is available in the species .txt file. This function takes that data and collates it into the final orgDb objects using AnnotationForge. It then makes some attempts to ensure that the resulting material created in the filesystem conforms to specifications which allow one to have multiple strains, etc. Finally, if everything goes according to plan, it calls devtools::install() and installs the resulting package.

### Usage

```
make_orgdb(orgdb_info, id = "lmajor_friedlin", cfg = NULL, kegg = TRUE,
  output_dir = "organismdbi", ...)
```

### Arguments

| | |
|---|---|
| `orgdb_info` | List of data frames generated by make_orgdb_info() |
| `id` | Human readable species identifier, keys off the cfg data frame. |
| `cfg` | Configuration data extracted either from inst/eupath_configuration.csv or provided by the user. |
| `kegg` | Attempt adding kegg data? |
| `output_dir` | Base output directory for the resulting packages. |
| `...` | Args to pass through. |

### Value

List of the resulting package name(s) and whether they installed.

### See Also

**AnnotationForge devtools** `makeOrgPackage`

### Examples

```
## Not run:
 orgdb_installedp <- make_orgdb(id="tcruzi_clbrener")

## End(Not run)
```

## make_orgdb_info *Generate the (large) set of data frames required to make functional OrgDb/TxDb/OrganismDbi objects.*

### Description

This function should probably be split into a few more pieces as it is pretty unwieldy at the moment.

### Usage

```
make_orgdb_info(gff, txt, kegg = TRUE)
```

### Arguments

| | |
|---|---|
| `gff` | File to read gff annotations from. |
| `txt` | File to read txt annotations from. |
| `kegg` | Boolean deciding whether to try for KEGG data. |

### Value

List containing gene information (likely from the txt file), chromosome information (gff file), gene types (gff file), gene ontology information, and potentially kegg information.

### See Also

**rtracklayer GenomicRanges**

### Examples

```
## Not run:
 orgdb_data <- make_orgdb_info(gff="lmajor.gff", txt="lmajor.txt")

## 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 = 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. |

## 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_report` *Make a knitr report with some defaults set a priori.*

---

## Description

I keep forgetting to set appropriate options for knitr. This tries to set them.

## Usage

```
make_report(name = "report", type = "pdf")
```

### Arguments

| | |
|---|---|
| `name` | Name the document! |
| `type` | Html or pdf reports? |

### Value

Dated report file.

### See Also

**knitr rmarkdown knitrBootstrap**

---

## `make_tooltips` *Create a simple df from a gff which contains tooltips usable in googleVis graphs.*

---

### Description

The tooltip column is a handy proxy for more thorough anontations information when it would otherwise be too troublesome to acquire.

### Usage

```
make_tooltips(annotations, desc_col = "description", type = "gene",
  id_col = "ID", ...)
```

### Arguments

| | |
|---|---|
| `annotations` | Either a gff file or annotation data frame (which likely came from a gff file.). |
| `desc_col` | Gff column from which to gather data. |
| `type` | Gff type to use as the master key. |
| `id_col` | Which annotation column to cross reference against? |
| `...` | Extra arguments dropped into arglist. |

### Value

Df of tooltip information or name of a gff file.

### See Also

**googleVis** `gff2df`

### Examples

```
## Not run:
 tooltips <- make_tooltips('reference/gff/saccharomyces_cerevisiae.gff.gz')

## End(Not run)
```

## make_txdb — *Create a TxDb object given data provided by make_orgdb_info()*

### Description

Much like make_orgdb() above, this uses the same data to generate a TxDb object.

### Usage

```
make_txdb(orgdb_info, cfg_line, gff = NULL, from_gff = FALSE,
  output_dir = "organismdbi", ...)
```

### Arguments

| | |
|---|---|
| `orgdb_info` | List of data frames generated by make_orgdb_info(). |
| `cfg_line` | Configuration data frame as per make_orgdb. |
| `gff` | File to read |
| `from_gff` | Use a gff file? |
| `output_dir` | Place to put rda intermediates. |
| `...` | Extra arguments to pass through. |

### Value

List of the resulting txDb package and whether it installed.

### See Also

**GenomicFeatures** **Biobase** **devtools** `createPackage`

### Examples

```
## Not run:
 txdb <- make_txdb(orgdb_output)

## End(Not run)
```

## mdesc_table *Get the description of a microbesonline genomics table*

### Description

This at least in theory is only used by get_microbesonline, but if one needs a quick and dirty SQL query it might prove useful.

### Usage

```
mdesc_table(table = "Locus2Go")
```

### Arguments

| | |
|---|---|
| `table` | Choose a table to query. |

### Value

Data frame describing the relevant table

### See Also

**DBI** `dbSendQuery` `fetch`

### Examples

```
## Not run:
 description <- mdesc_table(table="Locus2Go")

## End(Not run)
```

## 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")
```

## Arguments

| | |
|---|---|
| `data` | Data frame, presumably of counts. |
| `fact` | Factor describing the columns in the data. |

## 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`

## mytaxIdToOrgDb *Create an orgdb from an taxonID*

### Description

This function is a bit more fragile than I would like. I am not completely sold on AnnotationHub yet.

### Usage

```
mytaxIdToOrgDb(taxid)
```

### Arguments

| | |
|---|---|
| `taxid` | TaxonID from AnnotationHub |

### Value

An Orgdb instance

### See Also

**AnnotationHub S4Vectors**

### Examples

```
## Not run:
 orgdbi <- mytaxIdToOrgDb(taxid)

## End(Not run)
```

## 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.

---

## `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, norm = "raw", ...)
```

### Arguments

| | |
|---|---|
| `data` | Matrix of count data. |
| `design` | Dataframe describing the experimental design. (conditions/batches/etc) |
| `norm` | Normalization to perform: 'sf\|quant\|qsmooth\|tmm\|upperquartile\|tmm\|rle' I keep wishy-washing on whether design is a required argument. |
| `...` | More arguments might be necessary. |

### 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 = 5, low_to_zero = FALSE, thresh = 2,
  min_samples = 2, p = 0.01, A = 1, k = 1, cv_min = 0.01,
  cv_max = 1000, ...)
```

### 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(). |
| ... | 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_idmap` *Load organism annotation data (mouse/human).*

---

## Description

Creates a dataframe gene and transcript information for a given set of gene ids using the OrganismDbi interface.

## Usage

```
orgdb_idmap(orgdb, gene_ids = NULL, mapto = c("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

## See Also

**AnnotationDbi** `select` `keytypes`

## Examples

```
## Not run:
 host <- load_host_annotations(org, c("a","b"))

## End(Not run)
```

---

`parse_gene_go_terms` *TriTrypDB gene information table GO term parser*

---

## Description

TriTrypDB gene information table GO term parser

## Usage

```
parse_gene_go_terms(filepath, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `filepath` | Location of TriTrypDB gene information table. |
| `verbose` | Whether or not to enable verbose output. |

## Value

Returns a dataframe where each line includes a gene/GO terms pair along with some addition information about the GO term. Note that because each gene may have multiple GO terms, a single gene ID may appear on multiple lines.

## Author(s)

Keith Hughitt

## parse_gene_info_table *TriTrypDB gene information table parser*

### Description

An example input file is the T. brucei Lister427 gene information table available at: http://tritrypdb.org/common/downloads/C
5.0_TbruceiLister427Gene.txt

### Usage

```
parse_gene_info_table(file, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `file` | Location of TriTrypDB gene information table. |
| `verbose` | Whether or not to enable verbose output. |

### Value

Returns a dataframe of gene info.

### Author(s)

Keith Hughitt

## parse_go_terms *EuPathDB gene information table GO term parser*

### Description

Note: EuPathDB currently includes some GO annotations corresponding to obsolete terms. For example, the L. major gene LmjF.19.1390 (http://tritrypdb.org/tritrypdb/showRecord.do?name=GeneRecordClasses.GeneRecor
includes the term "GO:0003702" on the website and txt data file. The term has been deprecated, and does not have a category associated with it on the website. These will not be included in the final database.

### Usage

```
parse_go_terms(filepath)
```

### Arguments

| | |
|---|---|
| `filepath` | Location of TriTrypDB gene information table. |

## Value

Returns a dataframe where each line includes a gene/GO terms pair along with some addition information about the GO term. Note that because each gene may have multiple GO terms, a single gene ID may appear on multiple lines.

## Author(s)

Keith Hughitt

---

`parse_interpro_domains` *EuPathDB gene information table InterPro domain parser*

---

## Description

EuPathDB gene information table InterPro domain parser

## Usage

```
parse_interpro_domains(filepath)
```

## Arguments

| | |
|---|---|
| `filepath` | Location of TriTrypDB gene information table. |

## Value

Returns a dataframe where each line includes a gene/domain pairs.

## Author(s)

Keith Hughitt

---

`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 = "locus_tag")
```

### 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? |

### Value

Data frame of gene names and number of times the pattern appears/gene.

### See Also

**Biostrings Rsamtools Rsamtools** `FaFile` `getSeq` `PDict` `vcountPDict`

### Examples

```
## Not run:
 num_pattern = pattern_count_genome('mgas_5005.fasta', 'mgas_5005.gff')

## End(Not run)
```

---

## `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(df = NULL, conditions = NULL, batches = NULL, n = 20)
```

### Arguments

| | |
|---|---|
| `df` | a dataframe of (pseudo)counts |
| `conditions` | a factor or character of conditions in the experiment. |
| `batches` | a factor or character of batches in the experiment. |
| `n` | the number of genes to extract. |

### 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 together 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_data, expt_design = NULL, expt_factors = c("condition",
  "batch"), num_components = NULL, plot_pcas = FALSE)
```

## Arguments

| | |
|---|---|
| `expt_data` | the data to analyze (usually exprs(somedataset)). |
| `expt_design` | a dataframe describing the experimental design, containing columns with useful information like the conditions, batches, number of cells, whatever... |
| `expt_factors` | a character list of experimental conditions to query for R^2 against the fast.svd of the data. |
| `num_components` | a 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. |

### 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)
```

---

## pcRes — *Compute variance of each principal component and how they correlate with batch and cond*

---

### Description

This was copy/pasted from cbcbSEQ https://github.com/kokrah/cbcbSEQ/blob/master/R/explore.R

### Usage

```
pcRes(v, d, condition = NULL, batch = NULL)
```

### Arguments

| | |
|---|---|
| `v` | from makeSVD |
| `d` | from makeSVD |
| `condition` | factor describing experiment |
| `batch` | factor describing batch |

### Value

A dataframe containig variance, cum. variance, cond.R-sqrd, batch.R-sqrd

## See Also

plot_pca

---

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**

`pkg_cleaner` *Cleans up illegal characters in packages generated by make_organismdbi(), make_orgdb(), and make_txdb(). This attempts to fix some of the common problems therein.*

## Description

OrganismDbi instances are pretty neat, they pull together OrgDb and TxDb. With any luck, this function provides the ability to pull together all the data from the TriTrypDb, GO.db, and KEGGREST in order to accomplish these peculiar tasks.

### Usage

```
pkg_cleaner(path, removal = "-like", replace = "")
```

### Arguments

| | |
|---|---|
| `path` | Location for the original Db/Dbi instance. |
| `removal` | String to remove from the instance. |
| `replace` | What to replace removal with, when necessary. |

### Value

A new OrgDb/TxDb/OrganismDbi

### Examples

```
## Not run:
 crazytown <- make_organismdbi()  ## wait a loong time

## End(Not run)
```

---

## `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, batch_column = "batch", factor_type = "factor")
```

### Arguments

| | |
|---|---|
| `expt` | Experiment from which to acquire the design, counts, etc. |
| `svs` | Set of surrogate variable estimations from sva/svg or batch estimates. |
| `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, names = NULL, title = NULL,
  scale = NULL, ...)
```

### Arguments

| | |
|---|---|
| `data` | Expt or data frame set of samples. |
| `colors` | Color scheme, if not provided will make its own. |
| `names` | Another version of the sample names for printing. |
| `title` | A title! |
| `scale` | Whether to log scale the y-axis. |
| `...` | 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_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, ...)
```

### 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. |
| `...` | 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_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, sample_names = NULL,
  position = "identity", fill = NULL, title = NULL, scale = NULL,
  colors_by = "condition")
```

## Arguments

| | |
|---|---|
| `data` | Expt, expressionset, or data frame. |
| `colors` | Color scheme to use. |
| `sample_names` | Names of the samples. |
| `position` | How to place the lines, either let them overlap (identity), or stack them. |
| `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 |

## 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 description 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, ...)
```

## 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. |
| `...` | 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)
```

### 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. |

### 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.

### Usage

```
plot_essentiality(file)
```

### Arguments

| | |
|---|---|
| `file` | a file created using the perl script ’essentiality_tas.pl’ |

### 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. |

### 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 = 12,
  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_gvis_ma` *Make an html version of an MA plot: M(log ratio of conditions) / A(mean average).*

### Description

A fun snippet from wikipedia: "In many microarray gene expression experiments, an underlying assumption is that most of the genes would not see any change in their expression therefore the majority of the points on the y-axis (M) would be located at 0, since Log(1) is 0. If this is not the case, then a normalization method such as LOESS should be applied to the data before statistical analysis. If the median line is not straight, the data should be normalized.

## Usage

```
plot_gvis_ma(df, tooltip_data = NULL, filename = "html/gvis_ma_plot.html",
  base_url = "", ...)
```

## Arguments

| | |
|---|---|
| `df` | Data frame of counts which have been normalized counts by sample-type, which is to say the output from voom/voomMod/hpgl_voom(). |
| `tooltip_data` | Df of tooltip information (gene names, etc). |
| `filename` | Filename to write a fancy html graph. |
| `base_url` | String with a basename used for generating URLs for clicking dots on the graph. |
| `...` | more options are more options! |

## Value

NULL, but along the way an html file is generated which contains a googleVis MA plot. See plot_de_ma() for details.

## See Also

**googleVis** `plot_ma_de`

## Examples

```
## Not run:
 plot_gvis_ma(df, filename="html/fun_ma_plot.html",
              base_url="http://yeastgenome.org/accession?")

## End(Not run)
```

---

`plot_gvis_scatter` *Make an html version of a scatter plot.*

---

## Description

Given an arbitrary scatter plot, we can make it pretty and javascript-tacular using this function.

## Usage

```
plot_gvis_scatter(df, tooltip_data = NULL,
  filename = "html/gvis_scatter.html", base_url = "", trendline = NULL)
```

### Arguments

| | |
|---|---|
| `df` | Df of two columns to compare. |
| `tooltip_data` | Df of tooltip information for gvis graphs. |
| `filename` | Filename to write a fancy html graph. |
| `base_url` | Url to send click events which will be suffixed with the gene name. |
| `trendline` | Add a trendline? |

### Value

NULL, but along the way an html file is generated which contains a googleVis scatter plot. See plot_scatter() for details.

### See Also

**googleVis** `gvisScatterChart`

### Examples

```
## Not run:
 gvis_scatter(a_dataframe_twocolumns, filename="html/fun_scatter_plot.html",
              base_url="http://yeastgenome.org/accession?")

## End(Not run)
```

---

## `plot_gvis_volcano` *Make an html version of an volcano plot.*

---

### Description

Volcano plots provide some visual clues regarding the success of a given contrast. For our data, it has the -log10(pvalue) on the y-axis and fold-change on the x. Here is a neat snippet from wikipedia describing them generally: "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_gvis_volcano(toptable_data, fc_cutoff = 0.8, p_cutoff = 0.05,
  tooltip_data = NULL, filename = "html/gvis_vol_plot.html",
  base_url = "", ...)
```

### Arguments

| | |
|---|---|
| `toptable_data` | Df of toptable() data. |
| `fc_cutoff` | Fold change cutoff. |
| `p_cutoff` | Maximum p value to allow. |
| `tooltip_data` | Df of tooltip information. |
| `filename` | Filename to write a fancy html graph. |
| `base_url` | String with a basename used for generating URLs for clicking dots on the graph. |
| `...` | more options |

### Value

NULL, but along the way an html file is generated which contains a googleVis volcano plot.

### See Also

**googleVis** `plot_volcano`

### Examples

```
## Not run:
 plot_gvis_volcano(voomed_data, toptable_data, filename="html/fun_ma_plot.html",
                    base_url="http://yeastgenome.org/accession?")

## End(Not run)
```

---

## `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, ...)
```

### 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. |
| `...` | 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(fundata)
```

### Arguments

| | |
|---|---|
| `fundata` | A data frame to plot. |

`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,
  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. |
| `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_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, names = NULL,
  text = TRUE, title = NULL, yscale = NULL, ...)
```

### Arguments

| | |
|---|---|
| `data` | Expt, dataframe, or expressionset of samples. |
| `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:
 libsize_plot <- plot_libsize(expt=expt)
 libsize_plot  ## ooo pretty bargraph

## End(Not run)
```

`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, first = NULL, second = NULL, base_url = NULL,
  pretty_colors = TRUE, color_high = NULL, color_low = 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. |
| `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. |
| `first` | First column to plot. |
| `second` | Second column to plot. |
| `base_url` | Base url to add to the plot. |
| `pretty_colors` | Colors! |
| `color_high` | Chosen color for points significantly above the mean. |
| `color_low` | Chosen color for points significantly below the mean. |
| `...` | 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",
  pval_cutoff = 0.05, alpha = 0.4, logfc_cutoff = 1,
  label_numbers = TRUE, size = 2, tooltip_data = NULL,
  gvis_filename = NULL, invert = FALSE, ...)
```

## 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. |
| `pval_cutoff` | Name of the pvalue column to use for cutoffs. |
| `alpha` | How transparent to make the dots. |
| `logfc_cutoff` | 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. |
| `invert` | Invert the ma plot? |
| `...` | 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, toptable_data, 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)
```

### 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? |

### 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` | a list of plots |
| `file` | a file to write to |
| `cols` | the number of columns in the grid |
| `layout` | set the layout specifically |

## Value

a multiplot!

---

`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, labels = NULL,
  title = NULL, ...)
```

## Arguments

| | |
|---|---|
| `data` | Expt, expressionset, or dataframe. |
| `design` | Eesign matrix. |
| `colors` | Color scheme. |
| `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. |
| `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)
 nonzero_plot  ## ooo pretty

## 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, p_column = "limma_adjp", fc_column = "limma_logfc",
  bins = 100, constant_p = 0.05, constant_fc = 0)
```

### Arguments

| | |
|---|---|
| `table` | DE table to examine. |
| `p_column` | Column in the DE table defining the changing p-value cutoff. |
| `fc_column` | Column in the DE table defining the changing +/- log fold change. |
| `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:
 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 = 16)
```

### 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. |

### 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 ggplot PCA plot describing the samples' clustering.*

### Description

Make a ggplot PCA plot describing the samples' clustering.

### Usage

```
plot_pca(data, design = NULL, plot_colors = NULL, plot_labels = NULL,
  plot_title = NULL, plot_size = 5, size_column = NULL, ...)
```

## Arguments

| | |
|---|---|
| `data` | an expt set of samples. |
| `design` | a design matrix and. |
| `plot_colors` | a color scheme. |
| `plot_labels` | add labels? Also, what type? FALSE, "default", or "fancy". |
| `plot_title` | a title for the plot. |
| `plot_size` | size for the glyphs on the plot. |
| `size_column` | use an experimental factor to size the glyphs of the plot |
| `...` | arglist from elipsis! |

## Value

a list containing the following:

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_pcs *A quick and dirty PCA plotter of arbitrary components against one another.*

---

### Description

A quick and dirty PCA plotter of arbitrary components against one another.

### Usage

```
plot_pcs(pca_data, first = "PC1", second = "PC2", variances = NULL,
  design = NULL, plot_title = TRUE, plot_labels = NULL, plot_size = 5,
  size_column = NULL, ...)
```

### Arguments

| | |
|---|---|
| `pca_data` | a 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` | a list of the percent variance explained by each component. |
| `design` | the experimental design with condition batch factors. |
| `plot_title` | a title for the plot. |
| `plot_labels` | a parameter for the labels on the plot. |

| | |
|---|---|
| `plot_size` | The size of the dots on the plot |
| `size_column` | an experimental factor to use for sizing the glyphs |
| `...` | extra arguments dropped into arglist |

### Value

a ggplot2 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_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? |

### 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_qq_all_pairwise` *Perform qq plots of every column against every other column of a dataset.*

### Description

This function is stupid, don't use it. It makes more sense to just use plot_qq, however I am not quite read to delete this function yet.

### Usage

```
plot_qq_all_pairwise(data)
```

### Arguments

| | |
|---|---|
| `data` | Dataframe to perform pairwise qqplots with. |

### Value

List containing the recordPlot() output of the ratios, logs, and means among samples.

### See Also

**Biobase**

## `plot_qq_plot` *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_qq_plot(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_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.

## 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_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, names = NULL,
  title = NULL, Rowv = 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). |
| `names` | Alternate samples names. |
| `title` | Title of the plot! |
| `Rowv` | Reorder the rows by expression? |
| `...` | 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",
  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. |
| `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_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.

## Usage

```
plot_sm(data, colors = NULL, method = "pearson", names = NULL,
  title = NULL, ...)
```

## Arguments

| | |
|---|---|
| `data` | Expt, expressionset, or data frame. |
| `colors` | Color scheme if data is not an expt. |
| `method` | Correlation or distance method to use. |
| `names` | Use pretty names for the samples? |
| `title` | Title for the graph. |
| `...` | 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_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, chosen_factor = "snpcategory",
  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. |
| `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 = 12,
  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. |

### Value

List of MF/BP/CC pvalue plots.

### See Also

**topgo clusterProfiler**

## `plot_volcano` *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(toptable_data, tooltip_data = NULL, gvis_filename = NULL,
  fc_cutoff = 0.8, p_cutoff = 0.05, size = 2, alpha = 0.6,
  xaxis_column = "logFC", yaxis_column = "P.Value", ...)
```

## Arguments

| | |
|---|---|
| `toptable_data` | Dataframe from limma's toptable which includes log(fold change) and an adjusted p-value. |
| `tooltip_data` | Df of tooltip information for gvis. |
| `gvis_filename` | Filename to write a fancy html graph. |
| `fc_cutoff` | Cutoff defining the minimum/maximum fold change for interesting. This is log, so I went with +/- 0.8 mostly arbitrarily as the default. |
| `p_cutoff` | Cutoff defining significant from not. |
| `size` | How big are the dots? |
| `alpha` | How transparent to make the dots. |
| `xaxis_column` | Column from the data to use on the x axis (logFC) |
| `yaxis_column` | Column from the data to use on the y axis (p-value) |
| `...` | 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(toptable_data, 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)
```

---

## pp *png() shortcut*

---

## Description

I hate remembering my options for png()

## Usage

```
pp(file, width = 9, height = 9, res = 180)
```

## Arguments

| | |
|---|---|
| `file` | a filename to write |
| `width` | How wide? |
| `height` | How high? |
| `res` | The chosen resolution. |

## Value

a png 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_xls.

## See Also

`combine_de_tables`

## 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**

## 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.

---

## 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`

---

## require.auto

*Automatic loading and/or installing of packages.*

---

### Description

Load a library, install it first if necessary.

### Usage

```
require.auto(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/

## Value

0 or 1, whether a package was installed or not.

## See Also

**BiocInstaller** `biocLite` `install.packages`

## Examples

```
## Not run:
 require.auto("ggplot2")

## End(Not run)
```

---

`rex` *Resets the display and xauthority variables to the new computer I am using so that plot() works.*

---

## Description

This function assumes a line in the .profile which writes the DISPLAY variable to $HOME/.displays/$(hostname).last

## Usage

```
rex(display = ":0")
```

## Arguments

| | |
|---|---|
| `display` | DISPLAY variable to use, if NULL it looks in ~/.displays/$(host).last |

## 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 = 4, filename = "Rdata.rda.xz")
```

### Arguments

| | |
|---|---|
| directory | Directory to save the Rdata file. |
| backups | How many revisions? |
| filename | Choose a filename. |

### Value

Command string used to save the global environment.

### See Also

save pipe

### Examples

```
## Not run:
 saveme()

## End(Not run)
```

## 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(de_list, min_copies = 2, semantic = c("mucin",
  "sialidase", "RHS", "MASP", "DGF", "GP63"), semantic_column = "1.tooltip")
```

## Arguments

| | |
|---|---|
| `de_list` | List of sets of genes deemed significantly up/down with a column expressing approximate count numbers. |
| `min_copies` | Keep only those genes with >= n putative copies. |
| `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_filter

---

`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(de_list, max_copies = 2, use_files = FALSE,
  semantic = c("mucin", "sialidase", "RHS", "MASP", "DGF", "GP63"),
  semantic_column = "1.tooltip")
```

## Arguments

| | |
|---|---|
| `de_list` | 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? |
| `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

---

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 = "locus_tag")
```

## 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

```
## Not run:
 num_pattern = sequence_attributes('mgas_5005.fasta', 'mgas_5005.gff')

## End(Not run)
```

## `set_expt_batch` *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_batch(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_condition`

### Examples

```
## Not run:
 expt = set_expt_batch(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_condition` `set_expt_batch`

## 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_condition` *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_condition(expt, fact = NULL, ids = NULL, ...)
```

## Arguments

| | |
|---|---|
| `expt` | Expt to modify |
| `fact` | Conditions to replace |
| `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_batch create_expt

## Examples

```
## Not run:
 expt = set_expt_condition(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_condition set_expt_batch

## Examples

```
## Not run:
 expt = set_expt_factors(big_expt, condition="column", batch="another_column")

## 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_condition` `set_expt_batch`

### Examples

```
## Not run:
 expt = set_expt_samplenames(expt, c("a","b","c","d","e","f"))

## End(Not run)
```

## `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, fc_cutoffs = c(0, 1, 2),
  fc_column = "limma_logfc", p_type = "adj", invert = FALSE, p = 0.05,
  z = NULL, order = NULL, maximum = NULL, ...)
```

### Arguments

| | |
|---|---|
| `combined` | Result from combine_de_tables and/or extract_significant_genes(). |
| `fc_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. |
| `fc_column` | The column in the master-table to use for FC cutoffs. |
| `p_type` | Adjusted or not? |
| `invert` | Reverse the order of contrasts for readability? |
| `p` | Chosen p-value cutoff. |
| `z` | Choose instead a z-score cutoff. |
| `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:
 ## Damn I wish I were smrt enough to make this elegant and easily comprehendable, 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, all_genes, orgdb = "org.Dm.eg.db",
  orgdb_from = "FLYBASE", orgdb_to = "ENTREZID", internal = TRUE,
  go_level = 3, pcutoff = 0.05, qcutoff = 0.1, fc_column = "logFC",
  updown = "up", permutations = 100, min_groupsize = 5,
  kegg_prefix = "Dmel_", mings = 5, kegg_organism = "dme",
  categories = 12, parallel = TRUE)
```

## Arguments

| | |
|---|---|
| `sig_genes` | Dataframe of genes deemed 'significant.' |
| `all_genes` | 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. |
| `internal` | Used by the 'use_internal_data' flag. |
| `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? |
| `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. |
| `mings` | What is the minimum ontology group's size? |
| `kegg_organism` | Choose the 3 letter KEGG organism name here. |
| `categories` | How many categories should be plotted in bar/dot plots? |
| `parallel` | Perform slow operations in parallel? |

## 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, goids_df = NULL)
```

### Arguments

| | |
|---|---|
| `sig_genes` | Set of 'significant' genes as a table. |
| `de_table` | All genes from the original analysis. |
| `goids_df` | 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", ...)
```

## 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, qvalue = 0.1,
  length_keytype = "transcripts", go_keytype = "ENTREZID",
  goseq_method = "Wallenius", padjust_method = "BH",
  bioc_length_db = "ensGene", ...)
```

## 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.' |
| `qvalue` | Minimum qvalue 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? |
| `...` | 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, gff, goids_df, universe_merge = "id",
  second_merge_try = "locus_tag", species = "fun", pcutoff = 0.1,
  direction = "over", conditional = FALSE, categorysize = NULL,
  gff_type = "cds", ...)
```

## Arguments

| | |
|---|---|
| `sig_genes` | Input list of differentially expressed genes. |
| `gff` | Annotation information for this genome. |
| `goids_df` | Set of GOids, as before in the format ID/GO. |
| `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. |
| `direction` | Under or over represented categories. |
| `conditional` | Perform a conditional search? |
| `categorysize` | Category size below which to not include groups. |
| `gff_type` | Gff column to use for creating the universe. |
| `...` | 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", 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")
```

### Arguments

| | |
|---|---|
| `sig_genes` | Guess! The set of differentially expressed/interesting genes. |
| `species` | Organism supported by gprofiler. |
| `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! |

### 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_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_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", goids_df = 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, parallel = FALSE, ...)
```

### 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. |
| `goids_df` | 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? |
| `parallel` | Perform some operations in parallel to speed this up? |
| `...` | Other options which I do not remember right now! |

### Value

Big list including the various outputs from topgo

### See Also

**topGO**

## `sm` *Silence, m...*

### Description

Some libraries/functions just won't shut up. Ergo, silence, peasant! This is a simpler silence peasant.

### Usage

```
sm(...)
```

### Arguments

| | |
|---|---|
| `...` | Some code to shut up. |

### Value

Whatever the code would have returned.

## snp_add_file — *Add a new snp table to a set of comparisons for clustering.*

### Description

This is used by expt_snp to read input files and relatively quickly merge them.

### Usage

```
snp_add_file(sample, input_dir = "preprocessing/outputs",
  file_suffix = "_parsed_ratio.txt")
```

### Arguments

| | |
|---|---|
| `sample` | A text snp summary for 1 sample. |
| `input_dir` | Location of the data. |
| `file_suffix` | Suffix to use when finding the file(s). |

## subset_expt — *An alias to expt_subset, because it is stupid to have something start with verbs and others start with nouns.*

### Description

This just calls expt_subset.

### Usage

```
subset_expt(...)
```

### Arguments

| | |
|---|---|
| `...` | All arguments are passed to expt_subset. |

## subset_ontology_search — *Perform ontology searches on up/down subsets of differential expression 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, clusterProfiler, 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_exons *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_exons(data, gff = NULL, annotdf = NULL, parent = "Parent",
  child = "row.names")
```

### Arguments

| | |
|---|---|
| data | Count tables of exons. |
| gff | Gff filename. |
| annotdf | Dataframe of annotations (probably from gff2df). |
| 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.

### See Also

**rtracklayer** gff2df

### Examples

```
## Not run:
summed <- sum_exons(counts, gff='reference/xenopus_laevis.gff.xz')

## End(Not run)
```

## `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(file)
```

### Arguments

| | |
|---|---|
| `file` | a file created using the perl script 'essentiality_tas.pl' |

### Value

A plot and some numbers

### See Also

**ggplot2**

## `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 = "classic", ranksof = "classic")
```

### 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, transform = "raw",
  base = NULL, ...)
```

### Arguments

| | |
|---|---|
| `count_table` | A matrix of count data |
| `design` | Sometimes the experimental design is also required. |
| `transform` | 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)
```

---

## `translate_ids_querymany` *Use mygene's queryMany to translate gene ID types*

---

### Description

Juggling between entrez, ensembl, etc can be quite a hassel. This hopes to make it easier.

### Usage

```
translate_ids_querymany(queries, email = "abelew@gmail.com", fields = "all",
  species = "all")
```

### Arguments

| | |
|---|---|
| `queries` | Gene IDs to translate. |
| `email` | E-mail address to send with the querymany query. |
| `fields` | Set of fields to request, pass null for all. |
| `species` | Human readable species for translation (Eg. 'human' instead of 'hsapiens'.) |

## Details

Tested in test_40ann_biomart.R This function really just sets a couple of hopefully helpful defaults. When I first attempted to use queryMany, it seemed to need much more intervention than it does now. But at the least this function should provide a reminder of this relatively fast and useful ID translation service.

## Value

Df of translated IDs/accessions

## See Also

**mygene** queryMany

## Examples

```
## Not run:
 data <- translate_ids_querymany(genes)

## End(Not run)
```

---

`tritryp_downloads` *Download the various data files from http://tritrypdb.org/*

---

## Description

The tritrypdb nicely makes their downloads standardized!

## Usage

```
tritryp_downloads(version = "27", species = "lmajor", strain = "friedlin",
  dl_dir = "organdb/tritryp", quiet = TRUE)
```

## Arguments

| | |
|---|---|
| `version` | What version of the tritrypdb to use? |
| `species` | Human readable species to use. |
| `strain` | Strain of the given species to download. |
| `dl_dir` | Directory into which to download the various files. |
| `quiet` | Print download progress? |

## Value

List of downloaded files.

### Examples

```
## Not run:
 filenames <- tritryp_downloads(species="lmajor", strain="friedlin", version="28")

## 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.

---

## varpart — *Use variancePartition to try and understand where the variance lies in a data set.*

---

### Description

variancePartition is the newest toy introduced by Hector.

### Usage

```
varpart(expt, predictor = "condition", factors = c("batch"), cpus = 6,
  genes = 40, parallel = 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 |
| `cpus` | Number cpus to use |
| `genes` | Number of genes to count. |
| `parallel` | use doParallel? |

### Value

partitions List of plots and variance data frames

### See Also

**doParallel** **variancePartition**

---

## 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_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_xls

## Examples

```
## Not run:
 finished_comparison = deseq_pairwise(expressionset)
 data_list = write_deseq(finished_comparison)

## End(Not run)
```

`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 for them.

## Usage

```
write_de_table(data, type = "limma", ...)
```

## Arguments

| | |
|---|---|
| `data` | Output from results(). |
| `type` | Which DE tool to write. |
| `...` | 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_xls`

## Examples

```
## Not run:
 finished_comparison = eBayes(deseq_output)
 data_list = write_deseq(finished_comparison, workbook="excel/deseq_output.xls")

## 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_xls

### Examples

```
## Not run:
 finished_comparison <- edger_pairwise(expressionset)
 data_list <- write_edger(finished_comparison)

## 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 = FALSE, convert = "cpm", transform = "log2", batch = "sva",
  filter = "cbcb")
```

### Arguments

| | |
|---|---|
| expt | An expressionset to print. |
| excel | Filename to write. |
| norm | Normalization to perform. |
| violin | Include violin plots? |
| convert | Conversion to perform. |
| transform | Transformation used. |
| batch | Batch correction applied. |
| filter | Filtering method used. |

### 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_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, excel = "excel/goseq.xlsx", wb = NULL,
  add_trees = TRUE, pval = 0.1, add_plots = TRUE, height = 15,
  width = 10, ...)
```

### Arguments

| | |
|---|---|
| `goseq` | 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? |
| `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** **goseq**

## 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_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", add_plots = TRUE, height = 15,
  width = 10, ...)
```

### 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. |
| `add_plots` | Add some pvalue plots? |
| `height` | Height of included plots? |
| `width` | And their width. |
| `...` | More options, not currently used I think. |

### Value

A prettyified table in an xlsx document.

### See Also

**openxlsx gProfiler**

## 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. |

### Details

Tested in test_21limma.R

### See Also

`write_de_table`

### Examples

```
## Not run:
 finished_comparison = limma_pairwise(expressionset)
 data_list = write_limma(finished_comparison)

## End(Not run)
```

---

## 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,
  table_style = "TableStyleMedium9", ...)
```

### 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? |
| `table_style` | The chosen table style for excel |
| `...` | 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_xls` *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_xls(data = "undef", wb = NULL, sheet = "first", rownames = TRUE,
  start_row = 1, start_col = 1, ...)
```

## Arguments

| | |
|---|---|
| `data` | Data frame to print. |
| `wb` | Workbook to which to write. |
| `sheet` | Name of the sheet 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. |
| `...` | 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_xls(dataframe, sheet="hpgl_data")
  xls_coords <- write_xls(another_df, 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
  try_results <- xlsx_plot_png(fun_plot)

## End(Not run)
```

---

## `ymxb_print` *Print a model as y = mx + b just like in grade school!*

---

### Description

Because, why not!?

### Usage

```
ymxb_print(model)
```

### Arguments

| | |
|---|---|
| `model` | Model to print from glm/lm/robustbase. |

### Value

a string representation of that model.

# Index