Lusi Lab Library

```
#Enter either into an R console to load the functions:

source("https://www.depts.ttu.edu/psy/lusi/files/lusi.txt")
source("http://tiny.cc/lusilab")
```

Functions:

Analysis

parapower: simple parametric power analysis and effect size related functions.
bspmed: permute or bootstrap simple mediations.
medfig: make simple mediation figures.
repcv: cross-validate linear regression models.
afa/afa.model: automatically extract factors from measure items.

Data

reddit: download comments from reddit.
alz.posts: download posts from alzconnected.org.
scripts: download movie scripts from imsdb.
subtitles: download subtitles from opensubtitles.
move_reviews: download reviews from rottentomatoes.
movie_ratings_links: download metadata from imdb and rottentomatoes.

Curios

fractal: generate fractal images.
montyhall: simulate the Monty Hall problem.
schelling: run versions of Schelling's segregation model.

parapower

Description

Parametric effect size related functions for tests along the lines of balanced ANOVAs (e.g., certain t-tests, correlations, and regressions).

convert_effect takes an effect of a given type, and converts it to all other available types.

critical_effect takes a total sample size, and outputs effect sizes that would be significant at the given alpha level.

parapower conducts different types of simple parametric power analyses. There are 4 main components, and each can be estimated by entering all but the desired parameter:

- **N**: Before data are collected (*a priori*), usually you'd want a sample size necessary for a given effect size, alpha, and power. For example, `parapower(.25)` will find the *n* required to see a Cohen's *f* of .25, significant under an alpha (*p*-value) of .05, with a power of .8.
- **effect**: Maybe you have a maximum sample size, or one already collected, and you'd want to know what effect size you would be able to detect with a given alpha and power (sometimes called sensitivity analysis): `parapower(N = 50)`.
- **power**: Maybe you'd want to know what power you'd have to detect a given effect size given a sample size and alpha (sometime called post-hoc analysis): `parapower(.25, N = 50)`.
- **alpha**: Finally, maybe you'd want an alpha representing a specified power for a given effect and sample size (sometimes called criterion analysis): `parapower(.25, N = 50, power = .8)`.

Calculating *n* may be useful for planning studies, though this will be similar to using rules-of-thumb unless you have a good guess at an effect size. The other components may be requested for reporting (for whatever reason), but note that they add no new information if an effect size and sample size have also been reported.

Usage

```
convert_effect(effect, type = 'r', N = 3, k = 2, adjust_f = TRUE, display_f = FALSE,
               print = TRUE)

critical_effect(N, k = 2, alpha = .05)

parapower(effect = NULL, type = 'f', N = NULL, k = 2, alpha = .05, power = .8, range)
```

Arguments

effect	Effect size of the specified type.
type	Type of effect entered. Available types are Pearson's *r* ('r'), Cohen's *d* ('d'), Area Under the Curve (AUC; 'a'), Odds Ratio ('o'), η^2 (eta-squared; R^2; 'e'), or Cohen's *f* ('f').
N	Total sample size.
k	Number of groups or cells, with 2 being the default and most generally appropriate.
adjust_f	Logical indicating whether the entered effect of type 'f' should be converted from F to Cohen's *f* based on N and k: `sqrt((k - 1) * F / (N - k))`.
display_f	Logical; if TRUE, calculates F and an associated *p*-value based on N and k.
print	Logical; if FALSE, results are only invisibly returned.
alpha	Alpha level (*p*-value; theoretical type I error / false positive).
power	Power (1 − β; inverse theoretical type II error / false negative).
range	Interval of the estimated component for the optimizer to try. Defaults to 2-10,000 for N, 0-100 for effect, and 0-1 for power or alpha. If you get an error from uniroot (like "f() values at end points not of opposite sign"), try increasing the range (e.g., smaller low end for large effects, and larger high end for small effects).

Examples

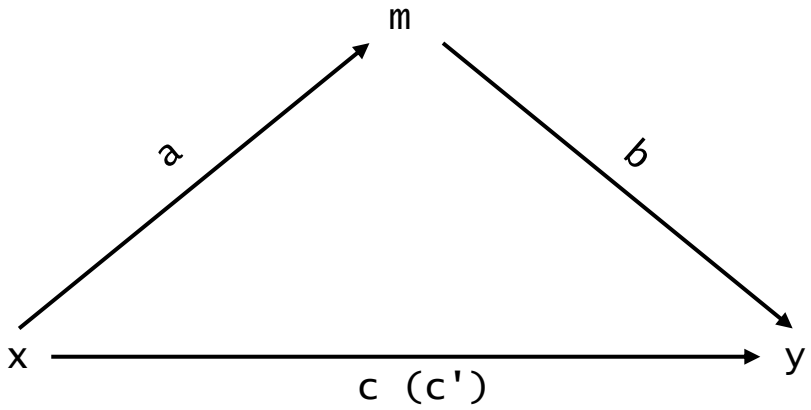
```
# a priori power analysis (looking for n given an effect size, alpha, and power)
# with no arguments, this defaults to effect = c(.3, .5, .7), type = 'd'
parapower()

# sensitivity analysis (looking for an effect size given n, alpha, and power)
parapower(N = 50)

# post-hoc power analysis (looking for power given an effect size, n, and alpha)
# since 'f' is the default type, this is power given Cohen's f = .3
parapower(.3, N = 50)

# criterion analysis (looking for alpha given an effect size, n, and power)
parapower(.3, N = 50, power = .8)


# This type of power analysis becomes increasingly difficult with more complex models.
# In those cases, simulation is very handy (though more difficult to abstract in terms
# of specification) -- if you can reasonably simulate the data you have in mind,
# you can reasonably estimate power for any model.

#
# Monte Carlo power analysis
#

# simulate a simple effect: continuous normal x and related y
N = 20
x = rnorm(N)
y = x * .4 + rnorm(N)
summary(lm(y ~ x))

## here, we can input the correlation to get other effect sizes
## k defaults to 2, which is appropriate in this case
convert_effect(cor(x, y), N = N)

## we want to know what N gives us .8 power to detect this effect at .05 alpha
## we can set up a function to easily try different Ns
sim1 = function(N, beta = .4, batches = 100, alpha = .05, power = .8){

  # first, set up a vector of 0s, which we'll fill with results from each run
  op = numeric(batches)

  # now we repeat the simulation batches times
  # (with i tracking the index of op)
  for(i in seq_len(batches)){

    # reroll the simulated variables
    x = rnorm(N)
    y = x * beta + rnorm(N)

    # set the current op entry to the p-value of the test
    op[i] = summary(lm(y ~ x))$coef[2, 4]

  }

  # observed power is percent of p-values under alpha
  # output of the function is observed power - set power for optimization
  # (where the target is 0; closest observed to set power)
  mean(op < alpha) - power
}

## uniroot will try different values of N (between 50 and 100 here),
## and output the one that gets closest to the set power
uniroot(sim1, c(50, 100))$root

## setting power to 0 will give the observed power
## increase batches for more stable estimates
sim1(55, batches = 5000, power = 0)

## compare with a binary x
```

```r
sim1b = function(N, beta = .4, batches = 100, alpha = .05, power = .8){
  op = numeric(batches)
  for(i in seq_len(batches)){
    x = rep_len(c(0, 1), N)
    y = x * beta + rnorm(N)
    op[i] = summary(lm(y ~ x))$coef[2, 4]
  }
  mean(op < alpha) - power
}

uniroot(sim1b, c(100, 300))$root
sim1b(200, batches = 5000, power = 0)

# this is what the most basic parametric power analysis is getting at
parapower(.4, 'd')

# simulate a more complicated effect: two binary variables with an interaction effect
c1 = sample(c(0, 1), N, TRUE)
c2 = sample(c(0, 1), N, TRUE)
y = c2 * c1 * .8 + rnorm(N)
(m = summary(lm(y ~ c1 * c2)))

## R-squared is eta^2, and k is 4 (2x2)
convert_effect(m$r.squared, 'eta^2', N, 4)

## same kind of function as before, only difference is the simulated data and test
## also have to recalculate the omnibus p-value since it's not returned
sim2 = function(N, beta = .8, batches = 100, alpha = .05, power = .8){
  op = numeric(batches)
  for(i in seq_len(batches)){
    c1 = sample(c(0, 1), N, TRUE)
    c2 = sample(c(0, 1), N, TRUE)
    y = c2 * c1 * beta + rnorm(N)
    f = summary(lm(y ~ c1 * c2))$fstatistic
    op[i] = pf(f[1], f[2], f[3], lower.tail = FALSE)
  }
  mean(op < alpha) - power
}

uniroot(sim2, c(50, 300))$root
sim2(100, batches = 5000, power = 0)

## note here that we're looking at the omnibus test
## if you're actually interested in, say, the interaction effect, that may affect power
sim2i = function(N, beta = .8, batches = 100, alpha = .05, power = .8){
  op = numeric(batches)
  for(i in seq_len(batches)){
    c1 = sample(c(0, 1), N, TRUE)
    c2 = sample(c(0, 1), N, TRUE)
    y = c2 * c1 * beta + rnorm(N)
    op[i] = summary(lm(y ~ c1 * c2))$coef[4, 4]
  }
  mean(op < alpha) - power
}

uniroot(sim2i, c(100, 300))$root
sim2i(200, batches = 5000, power = 0)
```

bspmed

Description

Displays the results of Sobel and permutation/bootstrapping tests for simple mediation.
Bootstrap confidence intervals indicate significance when they do not include zero (as they are confidence intervals of the effect). Permutation confidence intervals indicate significance when they do not include the indirect effect (as they are confidence intervals of the null effect).



Usage

```
bspmed(x, y, m, data=NULL, cov=NULL, random=NULL, su=NA, i=1000, method='perm',
       alpha=.05)
```

Arguments

x	x variable: can be the name of a variable in the environment or data.
y	y variable: can be the name of a variable in the environment or data.
m	mediating variable: can be the name of a variable in the environment or data.
data	data frame from which to pull variable, with the global environment as a fallback.
cov	a character vector of covariates to include in each path. Only accepted if data is also specified.
random	variable to be included as a random intercept in an lme model: `lme(y ~ x + m, random=~1|random)`
su	subset applied across variables, e.g., `su=x<2&m>0`.
i	the number of iterations to be processed for bootstrapping. Default is 1000.
method	character specifying the test type. Anything starting with 'b' will perform bootstrapping, otherwise will performs the non-iterative, permutation-of-residuals approach as described in Taylor and MacKinnon (2012; Eqs. 9 and 10).
alpha	the alpha level for the bootstrapping confidence interval. Default is .05 (i.e., 2.5% and 97.5%).

Examples

```
#This displays the mediation of weight on the relationship
#between displacement and miles per gallon.
bspmed(disp, mpg, wt, scale(mtcars))
```

medfig

Description

Plot simple mediation figures, maybe displaying indirect effects and confidence intervals of some sort, and potentially saving them.

Usage

```
medfig(mat, ci=NULL, ci.type=NULL, xlab='x variable', ylab='y variable', mlab='mediating
       variable', title=FALSE, note=TRUE, box=TRUE, digits=3, note.sig=TRUE,
       fin.length=.5, cords.box=list(), cords.arrow=list(), padding.box=.05,
       padding.arrow=.02, padding.number=.02, padding.title=.1, padding.note=.1,
       arrow.scale=.01, box.fill=NA, center.adj=.1,
       color = c(line=NA, box=NA, label=NA, number=NA, title=NA, note=NA),
       weight = c(line=NA, box=NA, label=NA, number=NA, title=2, note=.9),
       save=FALSE, format=pdf, name='mediation', ...)
```

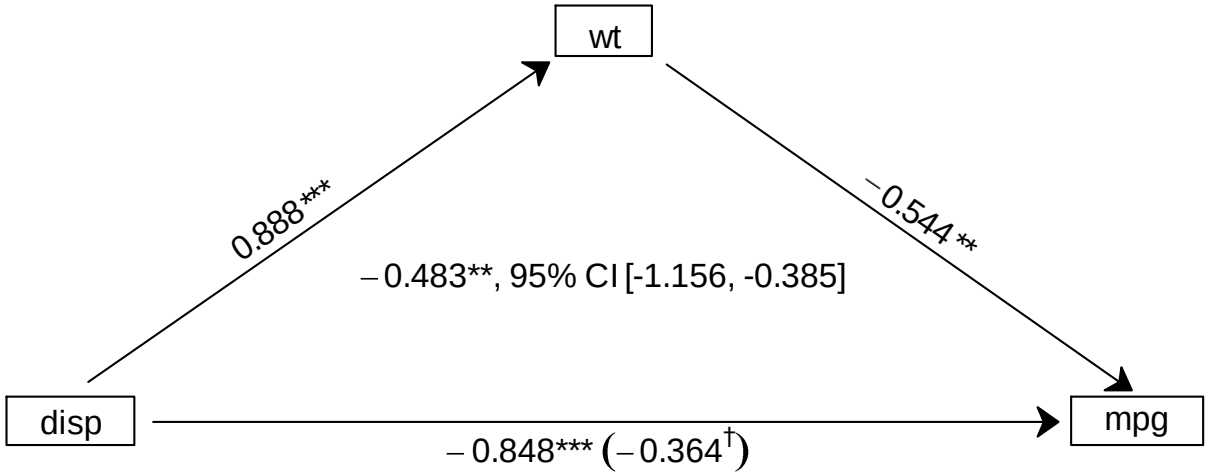
Arguments

`mat`	Output from the bspmed function, or a matrix with two rows and 4-5 columns (see example).
`ci`	A vector of confidence intervals (one lower and one upper). If the entries are named, these will be read as numbers and added together for the displayed percent (see example).
`ci.type`	Text to be displayed in the note (i.e., ci.type confidence intervals).
`xlab, ylab, mlab`	Text to be displayed in each variable box.
`title, note, box`	Logicals indicating whether to draw each element.
`digits`	Number of digits to round to.
`note.sig`	Logical indicating whether asterisks corresponding to p-values should be added to numbers.
`fin.length`	Length of the arrowhead fins, back from the line intersection (0 would be a flat base, for a triangular arrowhead).
`cords.box`	Center position of each box. Should be a list with at least one coordinate pair (a vector with values for y and x position). If entries aren't named, these x, y, and m are assigned as names; coordinates pairs can be entered positionally or named. Default is `list(x=c(.1,.1),m=c(.5,.9),y=c(.9,.1))`.
`cords.arrow`	Position of each arrow. This behaves in the same way as cords.box, but entries are names a, b, and c, and each set of coordinates should be 4 values specifying start and end positions. Any unspecifies coordinates are calculated based on box positions.
`padding.box, padding.arrow, padding.number, padding.title, padding.note`	Specifies the padding around each of the elements; box adjust space around texts in each box, arrow adjust distance from boxes, and number adjusts space between numbers and lines.
`arrow.scale`	Size of each arrowhead.
`box.fill`	Color of the box backgrounds.
`center.adj`	Amount to shift the center text downward (positive values making center text closer to the bottom).
`color`	A vector with named entries corresponding to any of the elements (line, box, label, number, title, note). Box refers to the color of the line (whereas box.fill specifies the background). Single unnamed values are applied to all elements.

weight	The weight of arrow lines or box outlines, or size of numbers or texts. Behaves the same as color.
save, format, name	Information used when saving the figure to a file; if any are specified, an image is saved. Format refers to the graphics engine; default is pdf. Some elements may not show up properly using other engines (such as cairo_pdf). Other engines are included in the base grDevices package. Another option is emf from the devEMF package (used to create the example image).
...	Additional arguments passed to par (parameters of the plot frame).

Examples

```
#medfig without any arguments will display the example from the bspmed function:
medfig()

#you can also save the output from a bspmed call and plot a figure for it later;
```



Permutation confidence intervals; p < 0.1†, p < 0.05*, p < 0.01**, p < 0.001***

```
#note that bspmed will show a figure by default.
med = bspmed(disp, mpg, wt, scale(mtcars), figure=FALSE)
medfig(med)

#Or you can enter a matrix, with confidence intervals in the second position,
#and what type they are in the third.
mat = matrix(
  c(.89, .0001, -.54, .001, -.85, .0001, -.36, .05, -.48, .001)
  , 2, dimnames=list(c('b', 'p'), c('a', 'b', 'c', "c'", 'i'))
)
medfig(mat, c('2.5'=-.38,'97.5'=.37), 'Permutation')
```

repcv

Description

Repeated cross-validation of linear regression models.

In each repetition (`reps`), the model is refit within each fold (`folds`) of the dataset. Across folds, if `permute` is `TRUE`, the smallest absolute estimate is checked against the average permuted estimate (`mean(null_b) > min(abs(b))`). Otherwise, the average *p*-value is checked against alpha (`mean(p) > alpha`). These are averaged across repetitions, resulting in something like a punished *p*-value (chance that the effect was not significant across all folds).

Usage

```
repcv(model, reps = 1000, folds = 3, data, group = NULL, permute = TRUE, alpha = .05,
      cores = parallel::detectCores() - 2, packages = NULL)
```

Arguments

model	an `lm`, `lme`, or `lmer` model object.
reps	the number of times the process should be repeated
folds	the number of random folds the data should be split into each repetition.
data	the data to split and refit the model to. If missing, the data is extracted from the model object.
group	a single grouping variable to sample rather than single data rows when splitting the data. If this is missing and the model is a mixed-effects model (fit with `lme` or `lmer`), it will be extracted from the model (only the first if there is more than one).
permute	if `TRUE` (default) the smallest absolute estimate between folds is compared to the average permuted estimate in each repetition. Otherwise, the average *p*-value across folds is compared to alpha in each repetition. The permutation method may be less conservative, and takes more time to process.
alpha	the value to check p against when `permute` is `FALSE`.
cores	the number of cores to use in parallel. Uses `foreach` for parallel processing. Set to less than 2 to use base functions.
packages	packages to load into each instance when multiple cores are being used. The `nlme` or `lme4` packages are loaded automatically when model appears to be from one of them.

Examples

```
# example with simulated data
# note that the ability to detect effects will
# depend in part on the size of each fold (n / folds)
x = rnorm(100)
y = x * .5 + rnorm(100)
repcv(lm(y ~ x))
```

afa/afa.model

Description

The goal of these functions is to automatically identify reliable item and factor structures for one or more measures.

afa is the main function which decides on items and factor structures through exploratory factor analyses (EFA), then reports a confirmatory factor analysis (CFA).

afa.model is the part that performs the EFAs. For each iteration (`iter`), data is split into a number of folds (depending on `size` if it is set, and/or the number of items), within each fold, the number of factors to use is determined (by parallel analysis [default] or a CFI cutoff rule, depending on `criteria`), and EFAs are run. Factors within the EFA are decided on by factor loadings (only items with a loading over `cutoff` are retained, then each item's loading is used to decide its factor). Factors are compared between folds, and a single factor is decided for each item based on where it most often lands. A count is then added for all of the items in each factor. Once this is done for all iterations, the final factor structures are decided by `cutoff.percent`; items appearing together more often than the cutoff are considered part of a final factor.

Usage

```
afa(items, data, fitinds=c('srmr','rmsea','tli','cfi','chisq','df','pvalue'), ...)

afa.model(x, nfactors, samples, size, criteria='parallel', cfi.change=.01, cutoff=.35,
          cutoff.percent, minvars=2, iter=100, rotation='promax', replace=TRUE,
          complete=FALSE, print=TRUE)
```

Arguments

`items`	A list with names as measure names, and vectors as items names.
`data`	A data frame in which the items can be found
`fitinds`	The results to be reported, potentially including any of `'srmr'`, `'rmsea'`, `'tli'`, `'cfi'`, `'chisq'`, `'df'`, and `'pvalue'`.
`...`	Passes additional arguments to afa.model
`x`	A matrix-like object with items as columns
`nfactors`	The maximum number of factors to consider; uses the number of items to decide if missing.
`samples`	The number of folds to use in each iteration. Decided by size if missing.
`size`	The size of each fold. Decided by the number of rows in x if missing.
`criteria`	How the number of factors is determined, matching either 'parallel' or 'cfi'.
`cfi.change`	The CFI cutoff if criteria='cfi'.
`cutoff`	The minimum factor loading size for items to be retained.
`cutoff.percent`	The percent of iteration items must appear together to be considered a factor. Defaults to something like chance, based on the number of factors (e.g., around .45 if there are two factors).

minvars	The minimum number of items that for a factor to be returned. Default is 2. If you set it to 1, single items appearing more often than the cutoff percent will be included in the output.
iter	The number of iterations. Defaults to 100. You may want to increase the number of iterations if you see items move around between runs, but note that the number of iterations is a fraction of the number of models being run (which is iter * folds * n factors before cutoff)
rotation	The rotation to use in the EFAs. See the factanal function for more.
replace	Determines if observations (rows) are replaces when sampling folds. The default is TRUE, which allows for more variation between resamples. Samples are always independently drawn, so there's always a chance of repeated observations. As samples become smaller (as determined by `size`), the potential effect of replacement is reduced. If `size` is equals the number of rows in `x`, if `replace = TRUE`, folds become bootstrap samples, and if `replace = FALSE`, folds become redundant.
complete	Logical; if TRUE, EFAs are run with one through nfactors number of factors. Otherwise, EFAs stop being run when the criteria is met.
print	Logical; if FALSE, iteration counting is suppressed.

Examples

```
#load some data
data = read.csv('https://osf.io/bq76u/?action=download')

#define items
measures = list(
  ESQ = c('EQpredict', 'EQawk', 'EQintuit', 'EQtune', 'EQmask',
    'SQmach', 'SQbuilding', 'SQtrain', 'SQroad', 'SQriver'),
  sexism = c('BSXrefined', 'BSXsupport', 'BSXcherish', 'HSXoffence',
    'HSXadvantage', 'HSXpower')
)

#process
res = afa(measures, data)
```

reddit

Description

Essentially a wrapper for the RedditExtractoR package; pulls comments from specified subreddits or searches.

Usage

```
reddit(topics, search=NULL, sort='hot', filename='reddit.csv', write=TRUE, lim=100,
       filter='\\[removed\\]', ...)

reddit.userdata(users, filename, subreddits, lim=100, type='comments')

reddit.karma(users)

reddit.lsm(data)
```

Arguments

topic	A string or vector of strings corresponding to subreddit names (e.g., 'trees' referring to reddit.com/r/trees/). Only the first value is used if search is specified.
search	Passed to RedditExtractoR::reddit_urls as the search_terms argument. If this is specified, if topic is specified, the first topic value will be added as the subreddit argument (which will restrict search to that subreddit).
sort	How to sort initial comments. Only applies if search is not specified. Default is 'hot', with 'new', 'rising', 'top', 'gilded', and 'ads', as options.
filename	Name of the file to be saved in the current working directory. This will currently always be a csv file.
write	Logical: if FALSE, data will not be save to a file (they will just be stored as objects if you've named your reddit call).
lim	Numeric: sets the number of posts to pull per topic. The max is 100 if search is not specified. If search is specified, lim will be used to set page_threshold (`round(lim/25)`).
filter	Passed to grepl. A pattern used to filter posts by the content of their comments. default is '\\[removed\\]' to filter out those comments that have been deleted.
...	Passed additional arguments to `RedditExtractoR::reddit_urls` if search is specified.
users	A vector of user names (as in reddit.com/user/**username**; such as those in the user column from the reddit function). Information is never gathered twice for the same user; `reddit.usercomments` simply removes duplicate user names (i.e., `unique(users)`), whereas `reddit.karma` will return karma scores in the same order as the input, filling in the same information for duplicate users.
subreddits	A vector of subreddits to filter for; only comments within the specified subreddits are returned. Should exactly match the subreddit_name_prefix field (e.g., `'r/Anxiety'` for https://www.reddit.com/r/Anxiety/), including 'r/' before each subreddit name, though this will be added if missing.
type	Type of user data to download; either 'comments' or 'submissions' (posts).
data	The data frame returned from a reddit call (e.g., `comments = reddit('trees'); reddit.lsm(comments)`)

Examples

```
#these will all save a file called 'reddit.csv' to the current working directory.
#pulls from a single, depression related subreddit:
reddit('depression')

#pull from a few subreddits, also saving the data as an object ('comments'):
topics = c('trees', 'Meditation')
comments = reddit(topics)

#pull comments from a search
reddit(search = 'politics')

#calculate language style matching between each comment and the comment it's replying to
#within the first thread of the trees subreddit
thread_lsm = reddit.lsm(comments[comments$title==comments$title[1],])

#download the 5 most recent comments from 10 users who commented in the trees subreddit
user_comments = reddit.usercomments(comments$user[1:5], lim=5)
```

alz.posts

Description

Scrapes forum posts from alzconnected.org.
Topics refers to original threads created by users within each forum, and *posts* refers to replies within those threads.

Usage

```
alz.forums(source=c('discussion','archive'))

alz.topics(forums, topic_pages=1, sort=NULL, ...)

alz.posts(topics, su=replies!=0, post_pages=1, ..., verbose=TRUE)
```

Arguments

source	Where to pull from; `'discussion'`, `'archive'`, or a vector with both.
forums	The output from an `alz.forums` call, or a vector of either forum names (partially matched) or IDs (those in the url for each forum).
topic_pages	The number of pages of topics to read within each forum. Default is 1.
sort	How topics should be sorted. Default is to sort by time (most recent first). `'time'` or `'-'` will sort by time in descending order (oldest topics first). Other options include `'title'` (alphabetically), `'starter'` (original poster), and `'views'` (most viewed first).
topics	The output from an `alz.topics` call, or a vector of either forum names or IDs (passed to `alz.topics`).
su	A filter for topics, based on the `alz.topics` data frame. Default is to exclude posts with no replies (`replies!=0`). Can filter based on any of the other `alz.topics` columns (`colnames(alz.topics())`).
post_pages	The number of pages of posts to read from each topic (in canonical order). Default is 1.
...	Passes arguments between functions. `alz.posts` will call `alz.topics` if topics is not a `data.frame` or is missing, which will call `alz.forums`. `alz.topics` arguments included in `alz.posts` are only applied if topics is a vector or is missing.
verbose	Logical; if `FALSE`, the thread count will no longer be displayed as they are being read.

Examples

```
#If no arguments are included, the first page of replies for each topic in the first
#page of topics in each forum will be returned.
posts = alz.posts()

#Could include all posts from all topics, but this would take several hours:
posts = alz.posts(topic_pages=Inf, post_pages=Inf)

#The same thing can be done separately
forums = alz.forums()
topics = alz.topics(forums)
posts = alz.posts(topics)

#These can be subset; e.g., this would only read forums with over 1000 posts:
topics = alz.topics(forums[forums$topics > 1000,])
```

scripts

Description

Scrapes movie scripts from imsdb.com.

Usage

```
scripts(genre, range=NULL, namesOnly=FALSE, dir=NULL, redownload=FALSE, pdf.dpi=400)
```

Arguments

genre	A string: either the name of a script (as it appears in the url, e.g., '12-Monkeys' from http://www.imsdb.com/scripts/12-Monkeys.html), a vector of such names, or the name of a genre, at least partially matching one of the accepted genres: 'action', 'adventure', 'all', 'animation', 'comedy', 'crime', 'drama', 'family', 'fantasy', 'film-noir', 'horror', 'musical', 'mystery', 'romance', 'sci-fi', 'short', 'thriller', 'war', or 'western'.
range	The scripts you actually want to download, based on the genre's list (e.g., range=1:10 for the first 10 scripts returned).
namesOnly	Logical: if TRUE, returns the list of script names without downloading them.
dir	Path to the folder in which to save downloaded scripts. By default, a folder in the current working directory will be created for each genre (e.g., "drama scripts", or "listed scripts" when names are entered instead of genres).
redownload	Logical: if TRUE, previously downloaded scripts will be overwritten. Otherwise (by default), scripts with associated files found in the output directory will be skipped.
pdf.dpi	Resolution of the .png images saved by pdftools::pdf_convert, to be processed by tesseract::ocr (which is only done when a plain text file is not found or is too short, and pdftools::pdf_text fails or returns empty).

Examples

```
#this will save all of the scripts included in the genre "family"
#to a folder called "family scripts" in the current working directory.
scripts('family')

#this will return a vector of script names.
movieTitles = scripts('drama', namesOnly=TRUE)

#then save a specified set of those scripts
scripts(movieTitles[30:40])
```

subtitles

Description

Downloads subtitle (.srt) files from opensubtitles.org based on IMDB ID or movie title.

Usage

```
subtitles(ids, zip = 'subtitles_zip', out = 'subtitles', names = ids, rate_limit = 1,
          user_agent = 'TemporaryUserAgent', langid = 'eng', redownload = FALSE, unzip =
          TRUE, rename = TRUE, cleanup = TRUE)

subtitles_clean(file, out = 'subtitles_clean', save = FALSE, overwrite = FALSE)
```

Arguments

ids	a vector of IMDB IDs (the number at the end of an IMDB URL, e.g., imdb.com/title/tt**0105665**), or movie titles to be searched for.
zip	path to a folder in which to save the download zip files. If the folder does not exist, it will be created.
out	path to a folder in which to save the extracted .srt (or cleaned .txt) files. If the folder does not exist, it will be created.
names	a vector corresponding to the ids vector, used to name the zip and extracted .srt files. This will default to the ids vector.
rate_limit	average number of seconds to wait between requests. Opensubtitles' limit is 40 requests every 10 seconds.
user_agent	user agent string used to identify yourself to opensubtitles. These can be requested from opensubtitles: trac.opensubtitles.org/projects/opensubtitles/wiki/DevReadFirst.
langid	language code (ISO 939-2). Defaults to 'eng'.
redownload	if TRUE, downloads and overwrite zip files it finds in the zip directory. By default, these will be skipped.
unzip	if FALSE, .srt files will not be extracted from the downloaded zip files.
rename	if FALSE, the original names of the .srt files will be preserved. Otherwise, these are renamed to the corresponding name entry.
cleanup	if FALSE, downloaded zip files that could not be unzipped will be retained. Otherwise, these will be removed.
file	path to a .srt file.
save	if TRUE, a new .txt files will be written. Otherwise, the cleaned text will just be invisibly returned.
overwrite	if TRUE, a .txt file it finds to already exist will be overwritten.

Examples

```
# creates a 'subtitles_zip' and 'subtitles' folder in your current working directory
subs = subtitles(c('north sea texas', 'one & two', 'the dirties'))

# creates a 'subtitles_clean' folder containing the cleaned subtitle files
for(file in list.files('subtitles', full.names = TRUE))
  subtitles_clean(file, save = TRUE)
```

movie_reviews

Description

Scrapes user reviews from rottentomatoes.com.

Usage

```
movie_reviews(titles, pages, write=TRUE, path=getwd(), filename)
```

Arguments

titles	A movie title, or vector of titles as they appear in the rottentomatoes url (e.g., 'marshal' from 'https://www.rottentomatoes.com/m/**marshall**/')
pages	The number of pages to pull from. There are up to 20 reviews per page.
write	Logical: if FALSE, the reviews are not written to a file.
path	Path to where the file should be saved. Default is the current working directory.
filename	What the file should be called, excluding the extension, as it is always .csv.

Examples

```
# save reviews from up to 10 pages of reviews for two movies:
movie_reviews(c('syriana','marshall'),10)
```

movie_ratings_links

Description

Search for imdb and rottentomatoes links based on movie titles. Then retrieve ratings and metadata from each site.

Usage

```
movie_ratings_links(titles, add = '', retry = TRUE, search_source =
                      'https://search.yahoo.com/search?q=')

rottentomatoes_meta(urls)

imdb_meta(urls)
```

Arguments

titles	a character vector of movie titles.
add	discriminating information to add to the query string, such as the year or director of each title; a character or vector of characters the same length as titles.
retry	logical; if FALSE, failed searches will not be retried.
search_source	URL to use for the search; each entry in titles is appended to this, with consecutive spaces, hyphens, and underscores replaced with '+', and '+movie+review' added. Yahoo is default because their rate limit seems more generous, but you could use Google, for example, with search_source = 'https://www.google.com/search?q='.
urls	a list of URLs for the appropriate site. This could be from the output of movie_ratings_links; see examples.

Examples

```
# search for ratings links
sites = movie_ratings_links(c('north sea texas', 'one & two', 'the dirties'))

# get metadata from imdb
imdb = imdb_meta(sites$links$imdb)

# get metadata from rottentomatoes
rottentomatoes = rottentomatoes_meta(sites$links$rottentomatoes)
```

fractal

Description

Make fractal images, and optionally save them as animated gifs.

Usage

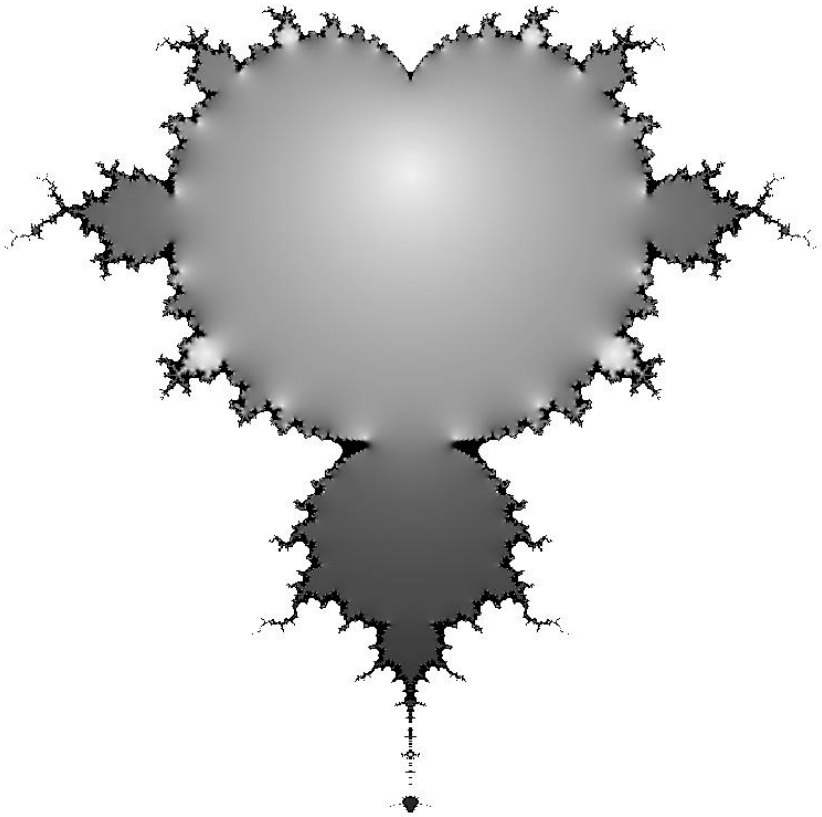
```
fractal(proc=z^2+c, e=25, size=min(dev.size(units='px')), rr=c(-1.8,0.6), ir=c(-1.2,1.2),
        col=grey(0:255/270), save=FALSE, name='fractal', delay=30, full=FALSE, ...)
```

Arguments

proc	The recursive process which evolves the complex. c is the initial complex, and z is the most recent state of the complex. z^2+c is the default, which produces a Mandelbrot set.
e	Number of times the complex should be evolved. Also corresponds to the number of frames in a gif, if a gif is saved. Default is 25.
size	Size of the images in pixels. Should be a single value, which will be both the height and width of the image. Takes the smallest dimension of the plot window by default.
rr	Range of the real part of the complex: default is c(-1.8, .6).
ir	Range of the imaginary part of the complex: default is c(-1.2, 1.2).
col	Colors to use in the image. By default, uses a set of 256 shades of grey. Could also use something like cm.colors(256) or rainbow(256) for colorful pictures.
save	Logical: if TRUE, a gif will be saved in the current working directory.
name	Name of the image file, if one is to be saved. 'fractal' by default. A '.gif' is always appended.
delay	The amount of time each frame is displayed in a gif in hundredths of a second. default is 30.
full	Logical: if TRUE, each evolution step is added to an array. If save is TRUE, or name or delay is specified, this will be set to TRUE as it is necessary to create gifs. The total memory requirement is a function of size and e; if full is FALSE, each evolution overwrites the last, so size can be much larger, but if full is TRUE, size and e will limit one another, and you are more likely to see memory allocation errors.
...	Passes additional arguments to write.gif from the caTools package.

Examples

```
#standard Mandelbrot set
fractal()
```



```
#cosine of the tangent of z squared plus the sine of c squared,
#and slightly more evolved?
fractal(cos(tan(z^2) + sin(c^2)), e=30)
```

montyhall

Description

Simulation of the Monty Hall problem, with tweakable elements.

Usage

```
montyhall(doors=3, choice=sample(options,1), judge='random', nopen=1, iter=10000)
```

Arguments

doors	Number of doors to choose from. Default and minimum is 3.
choice	Initial door you choose. Default is random.
judge	How the judge chooses which door to open, excluding the door you chose, and the wining door. Defaults to random. Can set the judge to be biased toward the left- or right-most available doors (e.g., `judge='left'`).
nopen	Number of doors the judge opens. Default is 1. Maximum is doors-2.
iter	Number of times to run the simulation. Default is 10000.

Examples

```
#Standard example
montyhall()

#output:
#
#Doors: 3
#Doors opened: 1
#Initial Choice: random
#Judge: randomly chooses which available door to open
#Iterations: 10000
#                                      win %
#always stay                          0.3341
#always switch                        0.6659
#randomly choose after door is opened  0.4945
#choose the left-most available door   0.5077
#choose the right-most available door  0.4923


#setup with biased judge and consistent initial choices
montyhall(choice=2, judge='left')

#output:
#
#Doors: 3
#Doors opened: 1
#Initial Choice: 2
#Judge: has a bias toward the left-most available door
#Iterations: 10000
#                                      win %
#always stay                          0.3347
#always switch                        0.6653
#randomly choose after door is opened  0.5024
#choose the left-most available door   0.6546
#choose the right-most available door  0.3454
```

schelling

Description

An implementation of Schelling's segregation model with adjustable parameters.

Usage

```
schelling(n = 138, dims = c(13, 16), probs = c(o = .5, '#' = .5), range = 1, ideals = .5,
          tols = .1, dist = Inf, concern = 'own', ..., distweight = .1, idealtype =
          'min', shuffle = TRUE, maxepochs = 99, moveto = 'nearest', record = FALSE, plot
          = TRUE, mfrow = c(1, 2), colors = NULL, gif = FALSE, delay = 20, magnify = 60)
```

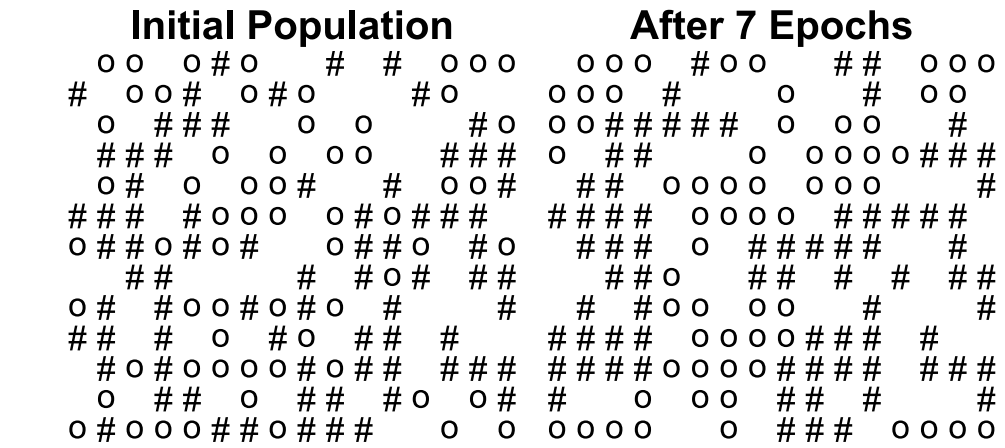
Arguments

`n`	determines the number of filled cells. If less than 1, treated as a percent of total cells, otherwise as a count.
`dims`	a vector setting the number of rows and columns of the area.
`probs`	sets the probability of each group being samples (the number of agents in each class). Can range between 0 and 1. If all entries are named, these will be taken as the class values, and as the template for `ideals, tols, range, dist, distweight, and idealtype`; any of these can alternatively specify classes – the longest named vector is taken as the template, with shorter vectors being recycled to match.
`range`	sets the range each group considers their neighborhood; 1 means all surrounding cells.
`ideals`	sets the ideal number of `concern` neighbors for each group. If less than 1, treated as a percent of occupied cells within range, otherwise treated as a count.
`tols`	when `idealtype` is `'specific'`, this sets how sensitive the agent will be – how off from their ideal can they be without feeling the need to move.
`dist`	sets how far each agent can move. `Inf` will allow agents to move anywhere in the space.
`concern`	specifies which class each group is concerned with. `'own'` sets concern to the group's name.
`...`	passes additional arguments to plot if `plot` is `TRUE`. The only special argument is `type`, which determines whether characters (`'ch'`) or colors (`'co'`) are plotted.
`distweight`	sets how much each group considers distance when moving; 0 will allow agents to choose the best option within their `dist`, otherwise farther cells are penalized.
`idealtype`	determines how the set ideals are treated for each group. If `'min'`, agents will move if there are fewer than their ideal number/percent of `concern` neighbors. If `'max'`, agents will move if there are ideal or more `concern` neighbors (being uncomfortable with too many `concern` neighbors). If `'specific'`, agents will move if the number or percent of `concern` neighbors is more than `tols` different from their ideal.
`shuffle`	logical; if `FALSE`, agents are checked and moved in a random order each epoch, otherwise they are checked and moved from the top left corner, proceeding down each column.
`maxepochs`	sets the maximum number of times all agents can be checked. Early stopping occurs if the space is identical to its previous state (no agents were moved in the last epoch).
`moveto`	sets the agent's moving strategy; either `'nearest'` (default), which looks for the closes cell that meets their requirements, or `'best'`, which looks for the best of all options within their `dist`.
`record`	logical; if `TRUE`, the space after each epoch is returned. Otherwise only the first and final space is returned.
`plot`	logical; if `TRUE`, the first and final state is plotted.

`mfrow`, `colors`	arguments passed to `par` and `plot`.
`gif`, `delay`, `magnify`	If any of these are specified, `gif` will be set to `TRUE` (a gif will be saved, assuming the `caTools` package is installed). `delay` is the amount of time each frame is shown in hundredths of a second. `magnify` enlarges the matrix (each cell is a pixel without magnification, and is repeated `magnify` times).

Examples

```
# standard example
schelling()
```




```
# 3, uneven groups
schelling(probs = c('#' = .2, '^' = .6, '~' = .2))
```


```
# everyone only cares about os
schelling(concern = 'o')
```