# Package: cgwtools (via r-universe)

September 14, 2024

**Type** Package

**Title** Miscellaneous Tools

**Version** 4.1

**Date** 2023-10-20

**Author** Carl Witthoft

**Maintainer** Carl Witthoft <cellocgw@gmail.com>

**Description** Functions for performing quick observations or evaluations of data, including a variety of ways to list objects by size, class, etc. The functions 'seqle' and 'reverse.seqle' mimic the base 'rle' but can search for linear sequences. The function 'splatnd' allows the user to generate zero-argument commands without the need for 'makeActiveBinding' . Functions provided to convert from any base to any other base, and to find the n-th greatest max or n-th least min. In addition, functions which mimic Unix shell commands, including 'head', 'tail' ,'pushd' ,and 'popd'. Various other goodies included as well.

**License** LGPL-3

**Imports** methods, gmp

**LazyData** true

**Depends** R (>= 3.5.0)

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2023-10-20 21:30:05 UTC

## Contents

---

`cgwtools-package` *A collection of tools that the author finds handy*

---

## Description

Most of these tools are small functions to do simple tasks or provide filtered views of the current environment. In addition the function `splatnd` is provided primarily as a piece of example code to show how to write zero-argument operators. It's based on the code in the package `sos` , and avoids the need to use `makeActiveBinding` (as in ,e.g., `pracma::ans` )

## Author(s)

Carl Witthoft, with attributions as noted in the individual help pages

Maintainer:Carl Witthoft carl@witthoft.com

approxeq *Do "fuzzy" equality and return a logical vector.*

## Description

This function compares two vectors (or arrays) of values and returns the near-equality status of corresponding elements. As with `all.equal()` , the intent is primarily to get around machine limits of representation of floating-point numbers. For integer comparison, just use the base == operator.

## Usage

```
approxeq(x, y, tolerance = .Machine$double.eps^0.5, ...)
```

## Arguments

| | |
|---|---|
| x, y | The two input items, typically vectors or arrays of data. |
| tolerance | Set the precision to which `abs(x[j] - y[j])` will be compared. The default argument provided is the R-standard value for floats. |
| ... | Not used at this time. |

## Details

If x and y are of different lengths, the shorter one is recycled and a warning issued.

## Value

A vector of the same length as the longer of x or y, consisting of TRUE and FALSE elements, depending on whether the corresponding elements of x and y are within the approximate equality precision desired.

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

all.equal, Comparison, identical

## Examples

```
x<-1:10
y<-x+runif(10)*1e-6
approxeq(x,y) #all FALSE
approxeq(x,y,tolerance=1e-5) #mostly TRUE, probably
```

## ascarr *Banner Versions of Characters*

### Description

This data set contains the "banner" arrays of characters to be used by the function banvax.

### Usage

```
ascarr
```

### Format

The array must be of dimension [6,8,N] for N characters. The dimnames for the third dimension must be the character matching each such layer.

## askrm *Interactive application of selected function a list of objects.*

### Description

This function was originally written to do the same as the unix rm -i command. The user supplies a list of items and the name of a function which is optionally applied to each item in turn.

### Usage

```
askrm(items = ls(parent.frame()), fn = "rm", ask = TRUE)
```

### Arguments

| | |
|---|---|
| items | A character vector of names of the objects to be acted upon (such as ls() returns). The default is all objects in the parent working environment. |
| fn | The name of the function to be applied, supplied as a character string. Possible future upgrades may allow function names to be entered without quotes. |
| ask | If TRUE , the user is prompted for "y/n" before performing the function on each object in the list. Be cautious about setting to FALSE for obvious reasons. Note that the only accepted positive response is **exactly** "y" so, e.g. "yes" will be treated as "no." |

### Value

A list with three elements.

| | |
|---|---|
| func | Echo back the input function, for archival reference. |
| selected | All the items from the input list to which the function fn was applied. In the default case, these are the items deleted from the environment. |
| evout | A list of the value(s) returned by the function, if any, each time it was executed. |

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

When interactive prompting is not desired, sapply or its brethren are recommended.

## Examples

```
# get rid of junky objects left around from testing
foo<-1
afoo<-c(foo,2)
foob <- c('a','b','d')
askrm(ls(pattern="foo") )

x<- rep(1,10)
y<- runif(10)
askrm(c('x','y'),'sd',ask=FALSE)
```

---

banvax *Function print a text banner in the style of the original DEC VAX "banner" function.*

---

## Description

This function takes a character string and writes the same characters as large block-letters into a text file.

## Usage

```
banvax(msg, file = 'banner.txt',  linewid = 80, bandat = cgwtools::ascarr)
```

## Arguments

| | |
|---|---|
| msg | A character string. See the Details section for the behavior for nonmatching characters. |
| file | A character string identifying an output file, or a connection as described in the help page for cat . |
| linewid | Defines the max number of characters per line. Change from the default value of 80 depending on the display or paper size in use. |
| bandat | The data array to use for the banner elements. In general, must be a 3-dimensional array with dimensions [x,y,N] where each n-th layer contains the banner form of one character. See the Details section. |

### Details

The supplied data file `ascarr` contains all letters and numerals and a bunch of punctuation marks such as " ;" , "," "\" , etc. If a character in the input is not found in the file, a "box" is used in its place. If a user-supplied file is specified, note that each layer of the [x,y,N] array must have a name equal to the character to be invoked. Here `x` specifies the number of columns and `y` the number of rows in each banner-element.

The function `cat` generates the output, so setting the argument `file` to "" will direct the output to the console. Quoting from the help page for `cat`, ' If "" (the default), cat prints to the standard output connection, the console unless redirected by `sink`. If it is "|cmd", the output is piped to the command given by cmd , by opening a pipe connection. '

### Value

Nothing is returned from the function. The output is a file or whatever `connection` is specified by the `file` argument.

### Author(s)

Author and Maintainer:Carl Witthoft `<carl@witthoft.com>`

### References

https://www0.mi.infn.it/~calcolo/OpenVMS/ssb71/6015/6017p041.htm#index_x_2757

---

## base2base *Function to convert any base to any other base (up to 36).*

---

### Description

This function now resides in the package `base2base`

---

## binit *Create histogram bins for each unique value in a sample.*

---

### Description

This is a Q&D way to create Pareto / histogram bins of a dataset when you want a separate bin for each value and don't want to deal with the 'breaks' or equivalent arguments in `hist` or other histogram functions in R packages.

### Usage

```
binit(samps,roundPrec=NULL)
```

### Arguments

| | |
|---|---|
| samps | A vector or array of data to be binned. |
| roundPrec | The number of digits to round samps to. Highly recommended when the data are floats |

### Details

binit sorts the input data and feeds the result to rle. This effectively produces histogram-like results. If you want a strict Pareto order (most common first), just sort the list elements lengths and values by the magnitudes in lengths.

### Value

A list containing two elements lengths: the number of items in each bin values: the data value associated with each bin

### Author(s)

Carl Witthoft, <carl@witthoft.com>

### See Also

rle , hist,

### Examples

```
x <- sample(1:100, 1000, rep=TRUE)
xbin <- binit(x)
plot(xbin$values,xbin$lengths, type = 'h')
# without rounding, will just be grass
x <- rnorm(1000)
 xbin <- binit(x,2)
plot(xbin$values,xbin$lengths, type = 'h')
```

---

## cumfun — *Function calculate the cumulative result of any function along an input vector.*

---

### Description

Calculates the cumulative value of almost any function in the same manner as cumsum

### Usage

```
cumfun(frist, FUN = sum, ...)
```

### Arguments

| | |
|---|---|
| frist | A vector of numerical or string values |
| FUN | The function to use. Can be either the function name (as is usually done) or a single character string naming the function. Default is sum. |
| ... | Additional arguments, if any, required for the chosen FUN function. See Details. |

### Details

If additional arguments are of length one, they are applied to each value in frist. If they are longer but not equal in length to length(frist) they will either be truncated or recycled to match. The builtin functions cumsum and cumprod generally are much faster than applying sum or prod to this function.

### Value

A list of the cumulative calculations generated.

### Author(s)

Carl Witthoft, <carl@witthoft.com>

### See Also

cumsum , cumall, rowCumsums

### Examples

```
foo <- 'abcdefcghicklmno'
grepfoo <-function(x,y) grep(y,x)
cumfoo <- cumfun(unlist(strsplit(foo,'')), FUN=grepfoo, y = 'c')
bar <- rnorm(1000,1)
cumsd <- cumfun(bar,FUN=sd)
plot(unlist(cumsd),type='l')
# compare with input std dev
lines(c(0,1000),c(1,1),lty=2,col='red')
```

---

## dim *Function to return dimensions of arguments, or lengths if dim==NULL.*

---

### Description

Simple overload to return dim when it's sensible and length otherwise

### Usage

```
dim(item)
```

### Arguments

| | |
|---|---|
| item | The object whose dimensions are to be determined |

### Value

Either a single value as returned by length or a vector of integers indicating the magnitude of each dimension as returned by dim

### Author(s)

Carl Witthoft, <carl@witthoft.com>

### See Also

dim , length,

### Examples

```
x1<-1:10
x2<-matrix(1,3,4)
dim(x1)
dim(x2)
```

---

## dirdir *Wrapper function around dir() which returns only directories found in the specified location(s).*

---

### Description

For those times when you only want to know the local directories available, use this instead of struggling through myriad arguments to dir . All arguments are the same as for plain old "dir" and are passed to dir .

### Usage

```
dirdir(path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE,
 recursive = FALSE, ignore.case = FALSE, include.dirs = FALSE, no.. = FALSE)
```

### Arguments

| | |
|---|---|
| path | a character vector of full path names; the default corresponds to the working directory, getwd(). Tilde expansion (see path.expand) is performed. Missing values will be ignored. Elements with a marked encoding will be converted to the native encoding (and if that fails, considered non-existent). |
| pattern | an optional regular expression. Only file names which match the regular expression will be returned. |

| | |
|---|---|
| all.files | a logical value. If FALSE, only the names of visible files are returned (following Unix-style visibility, that is files whose name does not start with a dot). If TRUE, all file names will be returned. |
| full.names | a logical value. If TRUE, the directory path is prepended to the file names to give a relative file path. If FALSE, the file names (rather than paths) are returned. |
| recursive | logical. Should the listing recurse into directories? |
| ignore.case | logical. Should pattern-matching be case-insensitive? |
| include.dirs | logical. Should subdirectory names be included in recursive listings? (They always are in non-recursive ones). |
| no.. | logical. Should both "." and ".." be excluded also from non-recursive listings? |

## Value

Note: this is directly quoted from the man page for `dir`. A character vector containing the names of the files in the specified directories (empty if there were no files). If a path does not exist or is not a directory or is unreadable it is skipped.

The files are sorted in alphabetical order, on the full path if full.names = TRUE.

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

dir

---

# findpat *Function to locate patterns ( sequences of strings or numerical values) in data vectors.*

---

## Description

Finds the location of a specified sequence either of numbers or strings in the source data item. If desired, for numerical data, both the source and the named pattern can be rounded to specified number of digits.

## Usage

```
findpat(datavec, pattern, roundit = NULL)
```

## Arguments

| | |
|---|---|
| datavec | A vector of numerical or string values |
| pattern | A vector containing the sequence to search for |
| roundit | If not NULL, sets the precision for rounding numbers. Ignored if datavec are strings |

### Details

If datavec and pattern are of different types, R will automagically convert to a common type and then compare the values. This is a result of the coercion rules identified in the man page for Comparison, "If the two arguments are atomic vectors of different types, one is coerced to the type of the other, the (decreasing) order of precedence being character, complex, numeric, integer, logical and raw." Use of the roundit argument is recommended whenever working with doubles (floats) to avoid the well-known and often overlooked pain of binary precision errors.

### Value

If the first element of pattern isn't found, a message is posted and an empty integer vector is returned. Otherwise, a vector of the indices of datavec where the desired pattern is located is returned. These are the indices of the start of the pattern.

### Author(s)

Carl Witthoft, <carl@witthoft.com>

### See Also

which , nth_number_after_mth, strfind

### Examples

```
fooc <- letters[c(1:15,4:9,12:26)]
 findpat(fooc,c('d','e','f'))
# 4 16
fooi <- c(1:50,5:9,60:80)
findpat(fooi,6:8)
# 6 52
findpat(fooi,c('6','7','8'))
# also 6 52
```

---

## getstack *Returns the current directory stack that pushd and popd manipulate*

---

### Description

getstack goes into the separate environment where pushd and popd operate and returns the current stack of directories.

### Usage

```
getstack()
```

### Arguments

none

## Details

Allowing a function to modify an object in the GlobalEnvironment is frowned upon by CRAN (and most programmers), so to maintain a directory stack a separate environment is established by pushd. Since this environment is not visible at the console level, `getstack` allows the user to check on the current status of the stack.

## Value

The current directory stack is returned as a vector of strings.

## Author(s)

Carl Witthoft <carl@witthoft.com>

## See Also

popd , pushd, setwd

## Examples

```
## depends on your local directory structure and permissions
getwd()
getstack() #empty, probably
pushd('..')
getstack()
pushd('.')
getstack()
popd()
getstack()
popd()
getstack()
getwd() #back where we started
```

---

`inverse.seqle` *Inverse of* `seqle`

---

## Description

As with `inverse.rle`, this function reverses the compression performed with `seqle` so long as you know the `incr` value used to generate the compressed data.

## Usage

```
inverse.seqle(x, incr = 1L)
```

### Arguments

| | |
|---|---|
| x | An object of class rle |
| incr | The increment between elements used to generate the compressed data object. Note that this can be either integer or float. For floating-point sequences, the reconstruction of the original series may differ at the level of floating-point precision used to generate the input object. |

### Value

a vector of values identical (or nearly so, for floats) to the original sequence. Note: Since the concept of "increment" has no reliable meaning when dealing with characters or char strings, when x is non-numeric the argument incr is ignored and the function reverts to base::inverse.rle....

### Note

The bulk of the code is taken directly from base::inverse.rle . Thanks to "flodel" on StackOverflow for suggesting code to handle floating-point increments.

### Author(s)

Carl Witthoft, <carl@witthoft.com>

### See Also

seqle , inverse.rle

### Examples

```
x<- c(2,2,2,3:8,8,8,4,4,4,5,5.5,6)
y<-seqle(x,incr=0)
inverse.seqle(y,0)
y <- seqle(x,incr=1)
inverse.seqle(y)
inverse.seqle(y,2) # not what you wanted
```

---

## lsclass *Q&D function to list all objects with the specified* class attribute .

---

### Description

This is one of the author's collection of ls* Q&D functions. Since anyone can define a new class at any time, there is no predefined set of legal or illegal class names. Remember that an object can have multiple classes. This function only allows searching for a single class name in a given call.

### Usage

```
lsclass(type = "numeric")
```

### Arguments

| | |
|---|---|
| type | The name of the class you're looking for. |

### Value

A vector of character strings containing the names of matching objects (as would be returned by the base function ls ) .

### Author(s)

Carl Witthoft carl@witthoft.com

### See Also

typeof, class , lstype

### Examples

```
xyzzy<-structure(vector(),class='grue')
lsclass('integer')
lsclass('grue')
```

---

## lsdata *List all objects in an .Rdata file.*

---

### Description

This function opens an .Rdata file, lists the contents, and cleans up after itself.

### Usage

```
lsdata(fnam = ".Rdata")
```

### Arguments

| | |
|---|---|
| fnam | the name of the datafile to be examined. |

### Value

The output of ls applied to the objects loaded from the specified data file.

### Author(s)

Carl Witthoft carl@witthoft.com

### References

Various people have published similar code on Stack Overflow.

## See Also

load, resave

## Examples

```
##not run because of complaints about detritus
# xblue<-1
# yblue<-2
# save(xblue,yblue,file='blue.Rdata')
# lsdata('blue.Rdata')
```

---

`lssize` *List the sizes of all selected objects.*

---

## Description

Just a toy to list the number of elements or optionally the bytesize as produced with `object.size` of a specified selection of objects. I find it handy when I want to rid an environment of large (or empty) objects. In the default case, `byte=FALSE` , lists and S4 objects are "taken apart" down to the lowest level so all individual elements are counted.

## Usage

```
lssize(items, byte = FALSE)
```

## Arguments

| | |
|---|---|
| `items` | A vector of character strings identifying the objects of interest as would be returned by, e.g. `ls(pattern="foo")` or `lstype("double")` . |
| `byte` | If TRUE, calculate the number of bytes taken up by an object. If FALSE, calculate the total number of elements of an object. |

## Value

A vector of the object sizes, with the object names as names for the elements

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## References

Many thanks to Martin Morgan of bioconductor.org who provided the recursive function for deconstructing an S4 Object. See http://stackoverflow.com/questions/14803237/ for the original question and answer.

## See Also

lstype , object.size, length

## Examples

```
x1<-runif(100)
x2<-runif(1000)
x3<-runif(2000)
lssize(ls(pattern='x[1-3]'))
lssize(ls(pattern='x[1-3]'),byte=TRUE)
#depending on what you have in your environment:
lssize(lstype('integer'))
```

---

# lstype *List all objects of the specified type.*

---

## Description

This is a Q&D tool to list all objects in the current environment of a specified type. As discussed in the base R documentation, these types are the vector types "logical", "integer", "double", "complex", "character", "raw" and "list", "NULL", "closure" (function), "special" and "builtin" (basic functions and operators), "environment", "S4" (some S4 objects).

## Usage

```
lstype(type = "closure")
```

## Arguments

| | |
|---|---|
| type | Any valid variable type, or "function," which is redirected to "closure." |

## Value

A vector of character strings as is returned by the base function ls .

## Author(s)

Carl Witthoft carl@witthoft.com

## See Also

ls , lssize, lsclass

## Examples

```
lstype('integer') #if you have any such in your environment.
```

maxn *Functions to find the n-th maximum or minimum of a vector or array.*

## Description

These functions behave similarly to `min, max` and `which(x == max/min (x)` to find the n-th greatest max or min of a dataset.

## Usage

```
maxn(x,nth=1)
minn(x,nth=1)
which.maxn(x, nth = 1, arr.ind = FALSE, useNames = TRUE)
which.minn(x, nth = 1, arr.ind = FALSE, useNames = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | A vector or array of numerical or string values. Note: ordering of string values may not be what is expected. See Details. |
| `nth` | Which lesser(greater, for min functions) to find, with default nth==1 being identical to `max, min`. |
| `arr.ind` | Same meaning as for `which`, a logical value indicating whether to return the array indices if `x` is an array. |
| `useNames` | Same meaning as for `which`, a logical value indicating if the value of arrayInd() should have (non-null) dimnames at all |

## Details

Quoting the help page for `max` : Character versions are sorted lexicographically, and this depends on the collating sequence of the locale in use: the help for Comparison gives details. The max/min of an empty character vector is defined to be character NA. (One could argue that as "" is the smallest character element, the maximum should be "", but there is no obvious candidate for the minimum.)

## Value

For `maxn, minn`, a single value which is the nth- max or min.

For the `which.min,max` functions, quoting from `which`: If arr.ind == FALSE (the default), an integer vector, or a double vector if x is a long vector, with length equal to sum(x), i.e., to the number of TRUEs in x. If arr.ind == TRUE and x is an array (has a dim attribute), the result is arrayInd(which(x), dim(x), dimnames(x)), namely a matrix whose rows each are the indices of one element of x

## Author(s)

Carl Witthoft, `<carl@witthoft.com>`

## See Also

which , max

## Examples

```
set.seed(17) # for repeatability
foo <- matrix(sample(1:10,20,replace=TRUE),5,4)
maxn(foo,3)
which.minn(foo,4)
```

---

minRow *Functions which mimic* max.col *to find for minimum or maximum of rows or columns.*

---

## Description

These are Q&D wrappers around max.col to make it easy to get the positions of max or the min of either rows or columns of an array. The description of the base function is, for comparison, "Find the maximum position for each row of a matrix".

## Usage

```
maxRow(mat,ties.method = c("random", "first", "last") )
minRow(mat,ties.method = c("random", "first", "last") )
minCol(mat,ties.method = c("random", "first", "last") )
maxCol(mat,ties.method = c("random", "first", "last") )
```

## Arguments

| | |
|---|---|
| mat | A 2-D matrix, same rules as for max.col |
| ties.method | Specify how to deal with ties, using same internal rules as max.col |

## Value

For each of these functions, same as for max.col : index of a max or min value for each row or column, an integer vector of length nrow(mat) or ncol(mat).

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

max.col

mystat *Calculate and display basic statistics for an object.*

## Description

This function calculates the min, max, median, mean, standard deviation, skew and kurtosis for the specified object and displays the results in a semi-tabular form. An option is provided to set the number of digits displayed for the returned values. Note: see the help pages in this package for theskew and thekurt for information on those implementations.

## Usage

```
mystat(x, numdig = 3, na.rm = TRUE, printit = TRUE)
```

## Arguments

| | |
|---|---|
| x | A vector or vectorizable object. |
| numdig | How many digits to the right of the decimal point to display (when printit is TRUE. |
| na.rm | Does the user desire NA values to be removed. Rare is the need to set this to FALSE. |
| printit | Set to TRUE to see the results, nicely formatted, in the console. |

## Value

A data frame with scalar elements matching their names:

| | |
|---|---|
| min | minimum |
| max | maximum |
| mean | mean value |
| median | median |
| sdev | standard deviation |
| skew | skew |
| kurtosis | kurtosis |

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

theskew , thekurt

## Examples

```
x <- runif(100)
mystat(x)
mystat(x,numdig=6)
```

---

`polyInt` *Function to find intersection points of two polygons.*

---

## Description

This is a Q&D tool to find the locations where two polygons, in a plane only (not 3D space), intersect.

## Usage

```
polyInt(poly1,poly2, stopAtFirst = FALSE, plotit = FALSE, roundPrecision = 10, ...)
```

## Arguments

| | |
|---|---|
| `poly1` | An Nx2 or 2xN matrix with X-values in the first row/column and Y-values in the second. |
| `poly2` | An Nx2 or 2xN matrix with X-values in the first row/column and Y-values in the second. |
| `stopAtFirst` | Boolean: if TRUE, then just return the first intersection point found. Useful time-saver when the user only needs to know if the polygons intersect. |
| `plotit` | Boolean: if TRUE, (and stopAtFirst is FALSE), the two polygons are plotted and the intersection points marked on the graph. |
| `roundPrecision` | Number of digits that data should be rounded to. This is necessary to avoid the usual floating-point precision problems when checking for possible duplicated intersection points. |
| `...` | Arguments to be passed along to the internal `line` call when `plotit` is TRUE. |

## Details

The function loops over all pairs of segments (one from `poly1` and one from `poly2`), calling `segSegInt` to see if they intersect. After all pair-combinations are tested, the collected intersection points, if any, are reduced to the unique collection. This avoids repetition when an intersection point is a vertex of one (or both) of the polygons. It is not necessary to "close" the supplied set of vertices, i.e. repeat the initial vertex at the end of the array as is needed to generate a complete line-plot of a polygon. The function will add that repeated vertex if it's not present in the input polygon(s).

Note: The supporting function `segSegInt` returns NA when two segments are parallel. However, when two polygons in fact have an overlapping (and thus parallel) couple of edges, the adjoining edges of one or both polygons will not be parallel to these parallel edges and will intersect one or both, so the vertex which lies on the other polygon's edge will be reported.

## Value

A matrix of the x and y coordinates of all intersection points, or, if `stopAtFirst` is TRUE, the first intersection point found. If no intersections exist, NULL is returned.

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

There are many tools which are far faster and more flexible. I wrote this one because it uses only base functions and doesn't require converting polygon vertices into a special class variable. Here are two common packages. `intersect` , `st_intersection`

## Examples

```
sqone <- cbind(c(0,1,1,0),c(0,0,1,1))
sqtwo <- sqone + 0.5
foo <- polyInt(sqone, sqtwo, plotit = TRUE)
```

---

# popd *Performs equivalent of* bash *command with same name*

---

## Description

popd is based on the cygwin bash manpages' description of these commands.

## Usage

```
popd(dn=FALSE, pull=0)
```

## Arguments

| | |
|---|---|
| dn | Determines whether a stack "pop" is to be performed. This is the equivalent of the first argument in `bash:popd` . If `dn` is `FALSE` and `pull` is zero, then set the new directory to the value at the top of the stack. If `dn` is `TRUE` then do not change directory, and look to `pull` for modifying the stack. See details for why the conditions are set this way. |
| pull | Equivalent of the latter `n` arguments in `bash`. Removes the stack entry corresponding to the `pull`'s value ; can be positive or negative. Note that there may be some inconsistency in how this is handled in different implementations of `bash`. |

## Details

Recommend reading `man bash` for full details of the operations. This implementation will not change the working directory if `dn` is `TRUE` The directory history is stored in a file in the function's environment (not console environment).`dirhist`, typically first created with `pushd` .

## Value

A status value: 0 for success or 1 if there is no stack file (.dirhist). Future upgrades may include other codes for other failure mechanisms, but for now error messages will have to suffice.

## Author(s)

Carl Witthoft <carl@witthoft.com>

## See Also

pushd , setwd

## Examples

```
## depends on your local directory structure and permissions
getwd()
pushd("~/..")
getwd()
popd()
getwd()
```

---

pushd *Performs equivalent of* bash *command with same name*

---

## Description

pushd is based on the cygwin bash manpages' description of these commands.

## Usage

```
pushd(path, dn=FALSE,rot=0)
```

## Arguments

| | |
|---|---|
| path | The directory to move into. |
| dn | Equivalent of the dir argument in bash . When TRUE, adds the current directory to the stack. |
| rot | Equivalent of the n argument in bash. Rotates the existing stack by the value of rot ; can be positive or negative. Note that there may be some inconsistency in how this is handled in different implementations of bash . |

## Details

Recommend reading man bash for full details of the operations. This implementation should do nothing more than change the working directory (and store directory history in a file in the function's environment (not console environment) .dirhist ).

## Value

A status value, which is always `0` for success. A future upgrade may implement a `trycatch` for conditions such as an inaccessible directory, but for now error messages will have to suffice.

## Author(s)

Carl Witthoft <carl@witthoft.com>

## See Also

popd , setwd

## Examples

```
## depends on your local directory structure and permissions
getwd()
pushd("~/..")
getwd()
popd()
getwd()
```

---

# ratRoot *Function to find the rational roots of any polynomial (when they exist).*

---

## Description

.

## Usage

```
ratRoot(Anum, Adenom = rep(1,times=length(Anum)) )
```

## Arguments

| | |
|---|---|
| `Anum` | A vector of the polynomial coefficients' numerators, starting with the highest power. Values can be numeric or bigz. |
| `Adenom` | A vector of the polynomial coefficients' denominators, starting with the highest power. Values can be numeric or bigz. Default is all ones, indicating that all input coefficients are integers. |

## Details

The code makes use of the 'Rational Root Theorem,' which states that all real, rational roots p/q meet two criteria. Given a polynomial of the form `An *x^n + A_(n-1) * x^(n-1) + . . . + A0 = 0`, all coefficients adjusted to be integers, 'p' must be a factor of the zero-power polynomial coefficient A0, and 'q' must be a factor of the maximum-power polynomial coefficient An. If any of the input `Adenom` are not '1', the function will make this adjustment.

## Value

A vector of bigq fractions representing all real rational roots found. If empty, all roots are irrational (or perhaps complex).

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## References

https://en.wikipedia.org/wiki/Rational_root_theorem

## Examples

```
ratRoot(c(6, -15, 261), c(1,1,49))
ratRoot(c(1,0,0,0,-1))
ratRoot(c(1,-2,1))
```

---

# resave *Add some objects to an existing* .Rdata *- type file.*

---

## Description

Take an existing myfile.Rdata data file and add the specified objects to it. This is achieved by opening the data file in a local environment, "dumping" the new objects into that environment, and re-saving everything to the same file name.

## Usage

```
resave(..., list = character(), file, overwrite = TRUE, loadverbose = FALSE )
```

## Arguments

| | |
|---|---|
| ... | Names of objects to save. Same usage as in base::save |
| list | A list of names of the objects to save. Can be used with or without any named arguments in ... . Same usage as in base::save |
| file | The name of the file to open and add items to. |
| overwrite | If TRUE (the default), items currently in the selected file will be overwritten with items of the same name in the console environment. If FALSE, they will not be overwritten; a warning message will be issued. |
| loadverbose | Default is FALSE. If TRUE, then the names of all objects currently in the file are echoed to the console |

## Value

Nothing is returned. This function is used solely to put objects into the file.

### Note

Code is essentially the same as that provided by "flodel" on StackOverflow, with some enhancements based on "save_append" by "nehartj" on GitHub

### Author(s)

Carl Witthoft <carl@witthoft.com>

### See Also

lsdata , save, load

### Examples

```
## not run to avoid creating detritus
# foo<-1:4
# bar<-5:8
# save(foo,file='foo.Rdata')
# resave(bar,file='foo.Rdata')
# #check your work
# lsdata('foo.Rdata')
```

---

## segSegInt *Function to find intersection point between two line segments (NOT lines).*

---

### Description

This function finds the intersection of the two lines containing the provided line segments, then checks whether the intersection is contained within both segments. This is an implementation of the equations presented, among many other sources, at https://paulbourke.net/geometry/pointlineplane/ . Bourke comments there, that "The equations apply to lines, if the intersection of line segments is required then it is only necessary to test if ua and ub lie between 0 and 1. Whichever one lies within that range then the corresponding line segment contains the intersection point. If both lie within the range of 0 to 1 then the intersection point is within both line segments."

### Usage

```
segSegInt(seg1, seg2=NULL, plotit=FALSE, probParallel = 1e-10,  ...)
```

### Arguments

| | |
|---|---|
| seg1 | An 2x2 or 4x2 matrix with X-values in the first column and Y-values in the second column. If 4x2, the lower 2 rows are assigned to seg2 and the input argument seg2 is ignored. |
| seg2 | An 2x2 or matrix with X-values in the first column and Y-values in the second column. Ignored if seg1 is 4x2. |

| | |
|---|---|
| plotit | Boolean: if TRUE, (and stopAtFirst is FALSE), the two line segments are plotted and, if one exists, the intersection point marked. |
| probParallel | A numeric value, typically quite small, used to identify line segments which probably are parallel. This basically is checking for identical slopes. |
| ... | Not used at present |

## Details

The function runs a check for parallelism so as not to return an infinity of intersection points, then basically checks for intersection of the lines to which the line segments belong, and finally verifies said intersection belongs to both line segments. This is an implementation of the equations presented at, among many other sources, https://paulbourke.net/geometry/pointlineplane/ . Bourke comments there, that "The equations apply to lines, if the intersection of line segments is required then it is only necessary to test if ua and ub lie between 0 and 1. Whichever one lies within that range then the corresponding line segment contains the intersection point. If both lie within the range of 0 to 1 then the intersection point is within both line segments."

## Value

A matrix of the x and y coordinates of the intersection points. If no intersection exists, `c(NA,NA)` is returned. This is done so the return value is always a 2-element vector.

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

polyInt

---

# seqle *Extends* rle *to find and encode linear sequences.*

---

## Description

The function `rle`, or "run-length encoder," is a simple compression scheme which identifies sequences of repeating values in a vector. `seqle` extends this scheme by allowing the user to specify a sequence of values with a common "slope," or delta value, between adjacent elements. `seqle` with an increment of zero is the same as `rle`.

## Usage

```
seqle(x, incr = 1L, prec = .Machine$double.eps^0.5)
```

## Arguments

| | |
|---|---|
| x | The input vector of values. |
| incr | The desired increment between elements which specifies the sequences to search for. Note that this can be either integer or float. For floating-point sequences, see the prec argument for determining what level of precision is used to determine whether elements continue a sequence or not. |
| prec | Defines the required precision to which elements are compared when determining whether they are part of a sequence. Note that for integer inputs, this value is more or less meaningless. |

## Details

Note: the returned value is assigned the class "rle". So far as I can tell, this class has only a print method, i.e. defining what is returned to the console when the user types the name of the returned object. Since the concept of "increment" has no reliable meaning when dealing with characters or char strings, when x is non-numeric the argument incr is ignored and the function reverts to base::rle.

## Value

| | |
|---|---|
| lengths | a vector of the lengths (1 or greater) of all sequences found. |
| values | a vector of the initial value for each sequence. For example, if incr ==1 a values of 5 associated with a lengths of 3 represents the sequence 5,6,7. |

## Note

The bulk of the code is taken directly from base::rle . Thanks to "flodel" on StackOverflow for suggesting code to handle floating-point increments.

## Author(s)

Carl Witthoft, <carl@witthoft.com>

## See Also

rle inverse.seqle

## Examples

```
x<- c(2,2,2,3:8,8,8,4,4,4,5,5.5,6)
seqle(x,incr=0)
seqle(x,incr=1)
seqle(x,incr=1.5)
```

short *Returns a small sample of the specified data set.*

## Description

The user specifies both the number of elements to display and the number of elements at the start and end of the vector to ignore ('skip') when selecting elements. The results are displayed in a nice tabular form. There are options to set the value of N as well as the number of values to "skip" before selecting the values. `short` is similar to a combination of the unix `head` and `tail` functions.

## Usage

```
short(x = seq(1, 20), numel = 4, skipel = 0, ynam = deparse(substitute(x)), dorows=FALSE)
```

## Arguments

| | |
|---|---|
| `x` | The data vector to be examined. |
| `numel` | How many elements to display. Note that `numel` elements of the beginning and of the end of the vector are returned. |
| `skipel` | If desired, skip the first `skipel` elements before returning `numel` elements. |
| `ynam` | Not normally changed by the user. `ynam` retrieves the name of the object in question, to be used in the output table formatting. |
| `dorows` | For matrices only, return the "numel" number of rows rather than elements. `dorows` is ignored with a warning if the input `x` has higher dimensionality. |

## Details

If the argument `x` happens to be a list, `short` unlists everything, so the first `numel` values will be taken from the first list element, going on to the second element as needed, etc.

## Value

Nothing is returned of interest. The function is called to provide what is printed directly to the console, which is a formatted table of the lead and tail values selected, with column labels identifying their location in the input vector object.

## Author(s)

Carl Witthoft <carl@witthoft.com>

## See Also

`head` , `tail`

## Examples

```
foo<-matrix(runif(100),nrow=20)
short(foo)
short(foo,numel=6,skipel=10)
short(foo,numel=6,skipel=10,dorows=TRUE)
```

---

splatnd *Execute simple zero-argument functions*

---

## Description

Execute simple zero-argument functions without having to type the "()" , and without having to go through the bother of `makeActiveBinding`. This code is provided primarily to allow the user to build his own set of command "shortcuts" by modifying the set of arguments to the `switch` function in the function body. The bulk of the code is copied from the excellent package `sos` . The name `splatnd` cannot be called directly, and doesn't even exist after being sourced. It serves to define a variety of operators `![your_string_here]` . If the string after `!` is not in the switch-list, the function defaults to the normal splat operator, i.e. NOT[your_string_here] .

## Arguments

none

## Details

The strings supported, with their associated functions, as delivered are: 'newdev' = dev.new(width=4.5, height= 4.5, restoreConsole=T), 'qapla' = cat('batlh tIn chav'), 'quitn' = quit('no'), 'quity' = quit('yes'), There's an obvious risk of undesired results should there exist an object in the environment with the same name as one of the items in the `switch` options. The workaround is to enclose the object name in parentheses. See the example.

## Value

The returned value is the result of whatever function or operator was invoked.

## Note

The bulk of the code is taken directly from the `sos` package.

## Author(s)

Carl Witthoft, `<carl@witthoft.com>`

## See Also

The R manuals on creating operators, `findFn` in the package `sos` , normally invoked as ???

## Examples

```
# based on the default items in splatnd.R
qapla <- 1:5
!qapla
!(qapla)
```

---

`thekurt` *Calculates the kurtosis of the input data set.*

---

## Description

Kurtosis is the next moment after skew (which is the moment after the variance). This function is provided primarily to support the function `mystat` . It uses the algorithm provided in the R package `e1071`

## Usage

```
thekurt(x)
```

## Arguments

| | |
|---|---|
| `x` | A vector or vectorizable data set. |

## Value

A single scalar, the kurtosis of the data provided.

## Author(s)

Carl Witthoft, `<carl@witthoft.com>`

---

`theskew` *Calculates the skew of a dataset.*

---

## Description

This function is included primarily to support `mystat` . Skew is the next moment after the variance. The algorithm used here is taken from the R package `e1071` .

## Usage

```
theskew(x)
```

## Arguments

| | |
|---|---|
| `x` | A vector of data to be evaluated |

## Value

A single scalar, the skew of the dataset

## Author(s)

Carl Witthoft, <carl@witthoft.com>

# Index