# Package 'codeutils'

June 22, 2023

**Title** Contains a Series of Utility Functions for File Handling and Programming

**Version** 0.0.4

**Date** 2023-02-18

**Maintainer** <malcolm.haddon@gmail.com>

**Description** Contains a Series of Utility Functions for File Handling, Programming, etc. Contains codeutilities, programming utilities, utils for Rmd files, and for diagrams.

**License** GPL-3

**Imports** knitr

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**NeedsCompilation** no

**Author** Malcolm Haddon [aut, cre]

## R topics documented:

## cart2pol *cart2pol converts cartesian coordinates into the polar angle*

### Description

cart2pol as a step in converting cartesian coordinates into polar coordinates this calculates the angle, in degrees, from x y values

### Usage

```
cart2pol(x)
```

### Arguments

| | |
|---|---|
| x | either a vector of two values of a matrix of pairs of values |

### Value

a single angle of vector of angles

### Examples

```
## Not run:
  cart2pol(c(3,3))  # should be 45
  dat <- matrix(c(3,4,5,7),nrow=2,ncol=2,byrow=TRUE)
  print(dat)
  cart2pol(dat)     # should be 36.8699 twice.

## End(Not run)
```

## circle *circle draws a circle with a given origin and radius*

### Description

circle provides the means of drawing a circle of a given radius and origin within a diagram ready for the addition of text.

### Usage

```
circle(origx = 50, origy = 50, radius = 10, col = 1, lwd = 1)
```

### Arguments

| | |
|---|---|
| origx | the final x origin |
| origy | the final y origin |
| radius | the radius of the circle |
| col | the col of the circle |
| lwd | the line width of the circle |

## Value

the matrix of x and y values invisibly

## Examples

```
makecanvas()
circle(origx=35,origy=70,radius=30,lwd=2,col=1)
circle(origx=65,origy=60,radius=30,lwd=2,col=2)
circle(origx=45,origy=40,radius=30,lwd=2,col=4)
```

---

`classDF` *classDF - tabulate the class of each column in a data.frame*

---

## Description

classDF - tabulate the class of each column in a data.frame.

## Usage

```
classDF(dataframe)
```

## Arguments

| | |
|---|---|
| `dataframe` | • the input data.frame for examination |

## Value

generates paired column names with their classes

## Examples

```
data(ChickWeight)
classDF(ChickWeight)
```

---

`codeutils` *codeutils: a set of functions to assist with code development*

---

## Description

The codeutils package provides an array of utility functions, these include documentation functions and summary functions. plot functions are to be found in the hplot package

**codeutilities**

**countgtOne** counts values >1 in a vector

**countgtzero** counts values >0 in a vector

**countNAs** count the numbr of NA values in a vector

**countones** count the number of values = 1 in a vector

**countzeros** count the number of values = 0 in a vector

**codeBlock** generates a comment section ready to copy into code

**classDF** Tabulates the class of each column in a dataframe

**getname** returns the name of a variable as character

**properties** characterizes properties of data.frame columns

**fileutils**

**listFunctions** lists all functions in a given R file

**classDF** identifies the class of each column in a data.frame

**rmdutils**

**lininterpol** linearly replaces NA values in a vector

**greplow** a case ignoring 'grep'

**printV** prints a vector as a column with index numbers

**quants** used in apply to estimate quantiles across a vector

**which.closest** finds closest value in a vector to a given number

**diagrams**

**canvas** sets up a plotting area ready for use

**circle** draws a circle with a given origin and radius

---

`countgtOne` *countgtOne used in apply to count the number > 1 in a vector*

---

## Description

countgtOne used in apply to count the number > 1 in a vector

## Usage

```
countgtOne(invect)
```

## Arguments

| | |
|---|---|
| `invect` | vector of values |

## Value

A single value of zero or the number of NAs

## Examples

```
## Not run:
  set.seed(12346)
  x <- trunc(runif(10)*10)
  x
  countgtone(x)  # should be 7

## End(Not run)
```

---

countgtzero *countgtzero used in apply to count the number >0 in a vector*

---

## Description

countgtzero used in apply to count number >0 in a vector

## Usage

```
countgtzero(invect)
```

## Arguments

| | |
|---|---|
| invect | vector of values |

## Value

A single value of number of values > 0

## Examples

```
## Not run:
  set.seed(12346)
  x <- trunc(runif(10)*10)
  x
  countgtzero(x)  # should be 9

## End(Not run)
```

---

countNAs *countNAs used in apply to count the number of NAs in a vector*

---

## Description

countNAs used in apply to count number of NAs in a vector

## Usage

```
countNAs(invect)
```

### Arguments

| | |
|---|---|
| invect | vector of values |

### Value

A single value of zero or the number of NAs

### Examples

```
## Not run:
  set.seed(12346)
  x <- trunc(runif(10)*10)
  x[c(3,7)] <- NA
  countNAs(x)  # should be 2

## End(Not run)
```

---

## countones *countones used in apply to count the number of ones in a vector*

---

### Description

countones used in apply to count number of ones in a vector

### Usage

```
countones(invect)
```

### Arguments

| | |
|---|---|
| invect | vector of values |

### Value

A single value of zero or the number of ones

### Examples

```
## Not run:
  set.seed(12346)
  x <- trunc(runif(10)*10)
  x
  countones(x)  # should be 2

## End(Not run)
```

## countzeros *countzeros used in apply to count the number of zeros in a vector*

### Description

countzeros used in apply to count zeros in a vector

### Usage

```
countzeros(invect)
```

### Arguments

| | |
|---|---|
| invect | vector of values |

### Value

A single value of zero or the number of zeros

### Examples

```
## Not run:
  set.seed(12346)
  x <- trunc(runif(10)*10)
  x
  countzeros(x)  # should be 1

## End(Not run)
```

## describefunctions *describefunctions lists all R functions in a set of files*

### Description

describefunctions lists all the R functions in a set of R files along with their syntax, the linenumber in each file, the filename, the function name, and the functions within the set of R files that each function calls. In addition, there is now a crossreference column, which identifies which functions call each function. If just the indir is provided then all R files in that directory will be examined. .Rmd files will not be considered but any other file type starting with .R may cause trouble until I find a fix!

### Usage

```
describefunctions(indir, files = "", outfile = "")
```

### Arguments

| | |
|---|---|
| indir | the directory in which to find the R files |
| files | a vector of filenames, as character, within which to search for functions, default="", which means all R files in indir will be used |
| outfile | the full path and name of the CSV file to which the results should be saved. default="", which means the output will only be returned invisibly. If outfile has a fullpath csv filename then it will also be written to that file as well as retunred invisibly |

### Value

It can produce a csv file but also returns the results invisibly

### Examples

```
filen <- tempfile("test",fileext=".R")
txt <- c("# this is a comment",
 "#' @title ...",
 "dummy <- function() {",
 "  out <- anotherdummy()",
 "  return(out)",
 "}",
 "# a possibly confusing use of function",
 "#' @title ...",
 "anotherdummy <- function() {",
 "  return(NULL)",
 "}",
 "  ")
 write(txt,file=filen)
 usedir <- paste0(tempdir(),"//")
 filename <- tail(unlist(strsplit(filen,"\\",fixed=TRUE)),1)
 x <- describefunctions(indir=usedir,files=filename,outfile="")
 x
```

---

## detibble *detibble if the input is a tibble it converts it back to a data.frame*

---

### Description

detibble is used to ensure that if an input object is a tibble then it is returned as a base R data.frame. This is sometimes needed as tibbles can upset some base R functionality.

### Usage

```
detibble(indat)
```

### Arguments

| | |
|---|---|
| indat | any matrix like data.frame or tibble |

### Value

a data.frame

## Examples

```
## Not run:
# syntax
detibble(tibble)

## End(Not run)
```

---

`diagrams` *diagrams provides the syntax of functions for making diagrams*

---

## Description

diagrams provides the syntax of functions for making diagrams

## Usage

```
diagrams()
```

## Value

nothing but it write syntax for diagram functions to the console

## Examples

```
diagrams()
```

---

`digitsbyrow` *digitsbyrow a helper function for knitr, to specify formats by row*

---

## Description

digitsbyrow is a solution obtained from StackOverFlow, suggested by Tim Bainbridge in 11/12/19. knitr formats table columns as a whole, which can be a problem if one wants to mix integers with real numbers in the same columns. This first transposes the data.frame/matrix being printed, fixes the formats, and then transposes it back. In knitr one then needs to use the align argument to fix the alignment. In may version I have conserved both rownames and colnames for both data.frames and matrices (the original only did so for data.frames but I often print matrices). digitsbyrow converts all entries to character so knitr becomes necessary for printing.

## Usage

```
digitsbyrow(df, digits)
```

## Arguments

| | |
|---|---|
| `df` | the data.frame or matrix to be printed by knitr |
| `digits` | a vector of the digits wanted for each row of the df or matrix |

## Value

a formatted data.frame or matrix depending on input

## Examples

```
x <- matrix(c(rnorm(5,mean=5,sd=1),seq(1,10,1)),nrow=3,ncol=5,byrow=TRUE,
            dimnames=list(1:3,1:5))
digitsbyrow(x, c(3,0,0))
# needs knitr to use kable
# kable(digitsbyrow(x, c(3,0,0)),align='r',row.names=TRUE)
```

---

`extractpathway` *extractpathway traces the sequence of functions calls within a function*

---

## Description

extractpathway is used when documenting the sequence of function calls within a set of functions within a package one is developing. It needs to know the location of the R directory (indir) for the package, the starting functions at the beginning of a particular algorithm, and a listing from the rutilsMH function describefunctions. Then it traces the sequential usage of all known package functions, ignoring base R functions. The final output is a vector of function names, starting with the top-level function.

## Usage

```
extractpathway(indir, infun, allfuns)
```

## Arguments

| | |
|---|---|
| `indir` | the R directory within an R package being documented |
| `infun` | the name of the top-level function whose sequential function use is being explored |
| `allfuns` | the output of applying readLines to a text file containing R code. |

## Value

a vector of function names in the sequence in which they are used within the first names function

## Examples

```
print("Wait on complex use of tempDir; see describefunctions for an example")
```

## extractRcode *extractRcode pulls out the r-code blocks from Rmd files*

### Description

extractRcode pulls out the r-code blocks from Rmd files and saves them into a separate R file.

### Usage

```
extractRcode(indir, rmdfile, filename = "out.R")
```

### Arguments

| | |
|---|---|
| indir | the directory in which the rmd file is to be found and into which the output file will be placed. |
| rmdfile | the name of the Rmd file whose R code is to be extracted |
| filename | the name of the R file into which the r-code is to go. |

### Value

generates an R file in the working directory, otherwise returns nothing

### Examples

```
print("wait on a real example")
```

## facttonum *facttonum converts a vector of numeric factors into numbers*

### Description

facttonum converts a vector of numeric factors into numbers. If the factors are not numeric then the outcome will be a series of NA. It is up to you to apply this function only to numeric factors. A warning will be thrown if the resulting output vector contains NAs

### Usage

```
facttonum(invect)
```

### Arguments

| | |
|---|---|
| invect | vector of numeric factors to be converted back to numbers |

### Value

an output vector of numbers instead of the input factors

## Examples

```
## Not run:
 DepCat <- as.factor(rep(seq(100,600,100),2)); DepCat
 5 * DepCat[3]
 as.numeric(levels(DepCat))  # #only converts levels not the replicates
 DepCat <- facttonum(DepCat)
 5 * DepCat[3]
 x <- factor(letters)
 facttonum(x)

## End(Not run)
```

---

`findfuns` *findfuns finds references to other functions within other functions*

---

## Description

findfuns is used when developing a complex project containing many R files, each containing many R functions. Given a file that contains a set of functions (infile) and a data.frame of all functions from the project (allfuns), which is obtained using listfuns, then findfuns searches each function for references to any of the projects functions. This allows them to be cross referenced

## Usage

```
findfuns(indir, infile, allfuns)
```

## Arguments

| | |
|---|---|
| `indir` | the directory in which the file identified in 'infile' is located |
| `infile` | the filename of the R file within which to search for the functions listed in the allfuns data.frame derived from the listfuns function |
| `allfuns` | a data.frame of functions and their properties listed in the order of the sorted function names in the 'function' column |

## Value

the same data.frame except that the references column will have been populated

## Examples

```
print("wait on suitable data-set")
```

## freqMean *freqMean calculates the mean and stdev of count data*

### Description

freqMean calculates the mean and stdev of count data it requires both the values and their associated counts and return a vector of two numbers.

### Usage

```
freqMean(values, counts)
```

### Arguments

| | |
|---|---|
| values | the values for which there are counts |
| counts | the counts for each of the values empty cells can be either 0 or NA |

### Value

a vector containing the mean and st.dev.

### Examples

```
## Not run:
vals <- c(1,2,3,4,5)    values=dat[,1];counts=dat[,2]
counts <- c(3,NA,7,4,2)
freqMean(vals,counts)  # should give 3.125 and 1.258306

## End(Not run)
```

## geomean *geomean log-normal bias corrected geometric mean of a vector*

### Description

Calculates log-normal bias corrected geometric mean of a vector. NAs and zeros are removed from consideration.

### Usage

```
geomean(invect)
```

### Arguments

| | |
|---|---|
| invect | is a vector of numbers in linear space. |

### Value

The bias-corrected geometric mean of the vector

### Examples

```
## Not run:
 x <- c(1,2,3,4,5,6,7,8,9)
 geomean(x)

## End(Not run)
```

---

## getDBdir *getDBdir identifies the DropBox path*

---

### Description

getDBdir identifies the path to DropBox within the users sub-directory within the C:/users/ directory. If not present then it returns NULL and gives a warning.

### Usage

```
getDBdir()
```

### Value

the path to the DropBox directory

### Examples

```
getDBdir()
```

---

## getmax *getmax generates the upper bound for a plot*

---

### Description

getmax generates upper bound for a plot where it is unknown whether the maximum is greater than zero of not. If > 0 then multiplying by the default mult of 1.05 works well but if the outcome if < 0 then the multiplier needs to be adjusted appropriately so the maximum is slightly higher than the maximum of the data

### Usage

```
getmax(x, mult = 1.05)
```

### Arguments

| | |
|---|---|
| x | the vector of data to be tested for its maximum |
| mult | the multiplier for both ends, defaults to 1.05 (=0.95 if < 0) |

### Value

a suitable upper bound for a plot if required

## Examples

```
## Not run:
 vect <- rnorm(10,mean=0,sd=2)
 sort(vect,decreasing=TRUE)
 getmax(vect,mult=1.0)
 vect <- rnorm(10,mean = -5,sd = 1.5)
 sort(vect,decreasing=TRUE)
 getmax(vect,mult=1.0)

## End(Not run)
```

---

`getmin` *getmin generates the lower bound for a plot*

---

## Description

getmin generates lower bound for a plot where it is unknown whether the minimum is less than zero of not. If less than 0 then multiplying by the default mult of 1.05 works well but if the outcome if > 0 then the multiplier needs to be adjusted appropriately so the minimum is slightly lower than the minimum of the data

## Usage

```
getmin(x, mult = 1.05)
```

## Arguments

| | |
|---|---|
| `x` | the vector of data to be tested for its minimum |
| `mult` | the multiplier for both ends, defaults to 1.05 (=0.95 if >0) |

## Value

a suitable lower bound for a plot if required

## Examples

```
## Not run:
vect <- rnorm(10,mean=0,sd=2)
sort(vect)
getmin(vect,mult=1.0)

## End(Not run)
```

## getname *getname returns the name of a variable as character*

### Description

getname runs 'deparse(substitute(x))' to get the name of the input variable. Saves remembering the syntax

### Usage

```
getname(x)
```

### Arguments

| | |
|---|---|
| x | any variable whose name is wanted as a character string |

### Value

a character string with the name of input variable

### Examples

```
## Not run:
a_variable <- c(1,2,3,4,5,6,7,8)
getname(a_variable)

## End(Not run)
```

## getnamespace *getnamespace returns the namespace for a given function*

### Description

getnamespace searches the loaded NameSpaces and returns the name of the NameSpace or package for the input function. This is used in by 'network'. If the namespace is not loaded this will not be able to be found.

### Usage

```
getnamespace(fun)
```

### Arguments

| | |
|---|---|
| fun | the name of the function of interest. It must be of class character, which can be obtained using 'getname' |

### Value

the name of the loaded NameSpace or package within which a function can be found.

## Examples

```
## Not run:
   getnamespace(getname(lm))
   getnamespace(getname(anova))

## End(Not run)
```

---

`getseed` *getseed generates a random number seed*

---

## Description

getseed generates a seed for use within set.seed. It produces up to a 6 digit integer from the Sys.time. This Initially, at the start of a session there is no seed; a new one is created from the current time and the process ID when one is first required. Here, in getseed, we do not use the process ID so the process is not identical but this at least allows the set.seed value to be stored should the need to repeat a set of simulations arise. The process generates up to a six digit number it then randomly reorders those digits and that becomes the seed. That way, if you were to call getseed in quick succession the seeds generated should differ even when they are generated close together in time.

## Usage

```
getseed()
```

## Value

an integer up to 7 digits long

## Examples

```
useseed <- getseed()
set.seed(useseed)
rnorm(5)
set.seed(12345)
rnorm(5)
set.seed(useseed)
rnorm(5)
```

---

`gettime` *gettime calculates time in seconds passed each day*

---

## Description

gettime is a function designed to facilitate the measurement of time between intervals within R software that are expected to take a maximum of hours. It calculates the time as seconds elapsed from the start of each day. As long as the timing of events does not pass from one day to the next accurate results will be generated.

## Usage

```
gettime()
```

### Value

the time in seconds from the start of a day

### Examples

```
## Not run:
  begin <- gettime()
  for (i in 1:1e6) sqrt(i)
  finish <- gettime()
  print(finish - begin)

## End(Not run)
```

---

greplow *greplow - uses tolower in the search for the pattern*

---

### Description

greplow a grep implementation that ignores the case of either the search pattern or the object to be search. Both are converted to lower case before using grep.

### Usage

```
greplow(pattern, x)
```

### Arguments

| | |
|---|---|
| pattern | • the text to search for in x |
| x | • the vector or object within which to search for 'pattern' once both have been converted to lowercase. |

### Value

the index location within x of 'pattern', if it is present, an empty integer if not

### Examples

```
## Not run:
txt <- c("Long","Lat","LongE","LatE","Depth","Zone","Effort","Method")
greplow("zone",txt)
greplow("Zone",txt)
greplow("long",txt)

## End(Not run)
```

## halftable *halftable halves the height of a tall narrow data.frame*

### Description

halftable would be used when printing a table using kable from knitr where one of the columns was Year. The objective would be to split the table in half taking the bottom half and attaching it on the right hand side of the top half. The year column would act as the index.

### Usage

```
halftable(inmat, yearcol = "Year", subdiv = 3)
```

### Arguments

| | |
|---|---|
| inmat | the data.frame to be subdivided |
| yearcol | the column name of the year field |
| subdiv | the number of times the data.frame should be subdivided; the default is 3 but the numbers can only be 2 or 3. |

### Value

a data.frame half the height and double the width of the original

### Examples

```
## Not run:
x <- as.data.frame(matrix(runif(80),nrow=20,ncol=4))
x[,1] <- 1986:2005
x[,4] <- paste0("text",1:20)
halftable(x,yearcol="V1",subdiv=2)
halftable(x[,c(1,2,4)],yearcol="V1")
x1 <- rbind(x,x[1,])
x1[21,"V1"] <- 2006
halftable(x1,yearcol="V1",subdiv=3)

## End(Not run)
```

## identifyfuns *identifyfuns uses text from readLines to identify function beginnings*

### Description

identifyfuns is used when tracing the interactions between functions within R packages. It uses the vector of character vectors that is produced by readLines and identifies the starting lines of all functions. It ignores all functions defined within comments, as well as ignoring all functions defined internally to other functions. It does the latter by testing for a couple of spaces at the start of a line containing a function definition, which functions defined within another function should have.

### Usage

```
identifyfuns(content)
```

### Arguments

| | |
|---|---|
| content | the output of applying readLines to a text file containing R code. |

### Value

a vector of line numbers identifying the start of all functions within the content. This may be a vector of zero length if there are no functions.

### Examples

```
txt <- c("# this is a comment",
"dummy <- function() { return(NULL) }",
"# a possibly confusing use of function",
"anotherdummy <- function() { return(NULL) }")
identifyfuns(txt)
```

---

## info *info gets the dimension or length of a matrix, array, data.frame or list*

---

### Description

info gets the dimension or length of a matrix, array, data.frame or list. It is safer than dim because if the object is a list dim fails.

### Usage

```
info(invar, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| invar | an object that is either a matrix, an array, a data.frame or a list |
| verbose | should the head of the object be printed to console. default=FALSE |

### Value

the dimensions of the object

### Examples

```
x <- array(rnorm(125,mean=5,sd=1),dim=c(5,5,5))
info(x,FALSE)
x <- list(x=5,y=6,z=7)
info(x)
```

## kablerow *kablerow a replacement for knitr::kable which enables row formatting*

### Description

knitr::kable enables one to round the number of digits for each column of a table. However, sometimes one wants to format the rows and not the columns. kablerow enables that while using the kable function. It rounds the rows to the desired number of digits and then converts those rounded values to characters, which kable can then print more appropriately.

### Usage

```
kablerow(x, rowdigits, namerows = NA, namecols = NA)
```

### Arguments

| | |
|---|---|
| x | an input matrix or data.frame |
| rowdigits | the number of digits desired for each row |
| namerows | should row.names be printed; default=NA. change to TRUE for row.names printing |
| namecols | should col.names be printed; default=NA (which prints V1, V2 ,V3, ...) |

### Value

Nothing but it does use knitr::kable to print a formatted matrix

### Examples

```
x <- matrix(rnorm(25,mean=5,sd=1),nrow=5,ncol=5)
colnames(x) <- 1:5
numdig <- c(2,3,4,3,2)
rownames(x) <- c("a","b","c","d","e")
kablerow(x,rowdigits=c(2,3,4,3,2),namerows=TRUE)
```

## lininterpol *lininterpol - linearly interpolate values in a vector with NAs*

### Description

lininterpol - linearly interpolate values in a vector with NAs. A common problem when plotting up time series is where there are missing values or NAs the plotted line will have gaps, one can always plot points on top of a line to identify where there are missing values but an alternative would be to interpolate the missing values linearly and plot that line as a dashed line. This function generates those linear interpolations. The input vector cannot have missing values at the beginning or the end. If there are no missing values the original vector is returned

### Usage

```
lininterpol(invect)
```

### Arguments

| | |
|---|---|
| invect | • the vector of values including missing values |

### Value

invect but with NAs replaced with linearly interpolated values.

### Examples

```
## Not run:
 Expt <- c(20102,18465,16826,15333,14355,NA,13843.7,NA,NA,NA,15180)
 lininterpol(Expt)

## End(Not run)
```

---

## listExamples *listExamples lists all the examples in a package R file*

---

### Description

listExamples lists all the examples in a package R file. It comments out the first line number and any dontrun statements along with their following curly bracket.

### Usage

```
listExamples(infile)
```

### Arguments

| | |
|---|---|
| infile | • a character variable containing the path and filename |

### Value

Creates an R file in the working directory and prints its name to the console

### Examples

```
## Not run:
txt <- vector("character",4)
txt[1] <- "#' @examples "
txt[2] <- "#' /dontrun{"
txt[3] <- "#' print("This is an example of using listExamples")"
txt[4] <- "#' }"
infile <- textConnection(txt)
listExamples(infile)

## End(Not run)
```

## listfuns *listfuns produces a listing of all functions in an input R file*

### Description

listfuns reads in a given R file and then identifies each function header within it and pulls out the function name, its syntax, the line-number in the file, and associates that with the filename.

### Usage

```
listfuns(infile)
```

### Arguments

| | |
|---|---|
| infile | the R file to be examined |

### Value

a data.frame of syntax, function name, line number, and file name

### Examples

```
print("wait for an example")
```

## magnitude *magnitude returns the magnitude of numbers*

### Description

magnitude is useful when using an optimizer such as optim, which uses a parscale parameter. magnitude can determine the respective parscale value for each parameter value.

### Usage

```
magnitude(x)
```

### Arguments

| | |
|---|---|
| x | the vector of numbers (parameters) whose magnitudes are needed |

### Value

a vector of magnitudes

### Examples

```
## Not run:
  x <- c(0,0.03,0.3,3,30,300,3000)
  magnitude(x)

## End(Not run)
```

## makecanvas *makecanvas sets up a plotting area ready for the flowchart*

### Description

makecanvas sets up a plotting areas ready for a flowchart made up of shapes, circles, polygons, rectangles, text, and arrows

### Usage

```
makecanvas(xstart = 0, xfinish = 100, ystart = 0, yfinish = 100)
```

### Arguments

| | |
|---|---|
| xstart | x-origin value defaults = 0 |
| xfinish | maximum of x axis defaults = 100 |
| ystart | y-origin value default = 0 |
| yfinish | y-axis maximum default = 100 |

### Value

nothing but plots an empty graph ready for polygons and text

### Examples

```
## Not run:
  makecanvas(ystart=50,yfinish=93.5)
  polygon(makevx(2,27),makevy(90,6),col=0,lwd=1,border=1)

## End(Not run)
```

## makelabel *makelabel generates a label from text and values*

### Description

makelabel It is common to want a label with text and a series of values. But paste and paste0 cycles the text and the values. To avoid this makelabel first combines the values as text and then adds the input text to the front of the values

### Usage

```
makelabel(txt, vect, sep = "_", digits = 3)
```

### Arguments

| | |
|---|---|
| txt | the input text for the label, can be empty |
| vect | the series of values to be included in the label |
| sep | the separator for the components; defaults to '_' |
| digits | how many significant digits for the values; default = 3 |

## Value

a character string made up of text and values

## Examples

```
pars <- c(18.3319532,33.7935124,3.0378107,6.0194465,0.5815360,0.4270468)
makelabel("Cohort1",pars[c(1,3,5)],sep="__")
makelabel("",pars[c(1,3,5)],sep="__",digits=4)
```

---

# makelist *makelist a utility that outputs a list structure defined by its input*

---

## Description

makelist is a utility that performs the common task of making a list structure of the same length as the vector of names input. It also names each of the list components after the vector of names. The output list structure is then ready to be populated with results.

## Usage

```
makelist(scenes)
```

## Arguments

| | |
|---|---|
| scenes | an vector of character names describing different scenarios |

## Value

a list of length scenes names for the vector of names in scenes

## Examples

```
scenarios <- c("base_case","higher_M","Lower_M")
changeM <- makelist(scenes=scenarios)
changeM
```

---

# makerect *makerect draws a rectangle once a plot is available*

---

## Description

makerect draws a rectangle after canvas has been called

## Usage

```
makerect(left, xinc, top, yinc, linecol = "grey", lwd = 1, col = NULL)
```

### Arguments

| | |
|---|---|
| left | defines lefthand edge of rectangle |
| xinc | left + xinc defines right-hand edge or rectangle |
| top | defines top edge of rectangle |
| yinc | top - yincdefines bottom edge of rectangle |
| linecol | colour of line. default="grey" |
| lwd | the width of the line, default=1 |
| col | the fill colour of the polygon drawn. default=NULL so not filled |

### Value

a vector denoting the center (x,y) of the rectangle

### Examples

```
## Not run:
   canvas(ystart=50,yfinish=93.5)
   makerect(left=2,xinc=27,top=90,yinc=6)

## End(Not run)
```

---

## makeUnit *makeUnit generates a unit matrix whose diagonal can be changed*

---

### Description

makeUnit generates a unit matrix but includes the facility to alter the diagonal value away from 1.0 if desired.

### Usage

```
makeUnit(N, diagvalue = 1)
```

### Arguments

| | |
|---|---|
| N | the order of the matrix |
| diagvalue | defaults to 1.0, but otherwise can be a different constant or a vector of dimension N |

### Value

a square matrix defaulting to a unit matrix

### Examples

```
## Not run:
  makeUnit(4)
  surv <- exp(-0.2)
  makeUnit(4,surv)

## End(Not run)
```

## makevx *makevx make an x values vector*

### Description

makevx takes the left x value of a rectangle and the increment rightwards that defines a vector describing the four vertices of the rectangle topleft, topright, bottomright, bottomleft, topleft. when matched with makevy generates the descriptor for a complete rectangle.

### Usage

```
makevx(init, inc)
```

### Arguments

| | |
|---|---|
| init | x-value for the left-hand edge of a rectangle |
| inc | the x-increment added to init to define the right-hand edge |

### Value

a vector of y-values

### Examples

```
## Not run:
 plot(0:100,seq(58,93.5,length=101),type="n",xaxt="n",yaxt="n",
 xlab="",ylab="", bty="n")
 polygon(makevx(2,27),makevy(90,6),col=0,lwd=1,border=1)

## End(Not run)
```

## makevy *makevy make a y values vector*

### Description

makevy takes the top y value of a rectangle and the vertical increment downwards and defines a vector describing the four vertices of the rectangle topleft, topright, bottomright, bottomleft. topleft, when matched with makevx generates the descriptor for a complete rectangle.

### Usage

```
makevy(init, inc)
```

### Arguments

| | |
|---|---|
| init | y-value for the top edge of a rectangle |
| inc | the y-increment subtracted from init to define the lower edge |

## Value

a vector of y-values

## Examples

```
## Not run:
 canvas(ystart=50,yfinish=93.5)
 polygon(makevx(2,27),makevy(90,6),col=0,lwd=1,border=1)

## End(Not run)
```

---

outfit *outfit tidy print of output from optim, nlminb, or nlm*

---

## Description

outfit takes in the output list from either optim, nlminb, or nlm and prints it more tidily to the console, In the case of nlm it also prints the conclusion regarding the solution. It might be more effective to implement an S3 method.

## Usage

```
outfit(inopt, backtran = TRUE, digits = 5, title = "", parnames = "")
```

## Arguments

| | |
|---|---|
| inopt | the list object output by nlm, nlminb, or optim |
| backtran | a logical default = TRUE If TRUE it assumes that the parameters have been log-transformed for stability and need back-transforming |
| digits | the number of digits to round the backtransformed parameters. defaults to 5. |
| title | character string used to label the output if desired, default = empty character string |
| parnames | default="" which means the estimated parameters will merely be numbered. If a vector of names is given then this will be used instead, at least, for nlm and optim. |

## Value

nothing but it does print the list to the console tidily

## Examples

```
x <- 1:10  # generate power function data from c(2,2) + random
y <- c(2.07,8.2,19.28,40.4,37.8,64.68,100.2,129.11,151.77,218.94)
alldat <- cbind(x=x,y=y)
pow <- function(pars,x) return(pars[1] * x ^ pars[2])
ssq <- function(pars,indat) {
   return(sum((indat[,"y"] - pow(pars,indat[,"x"]))^2))
}  # fit a power curve using normal random errors
pars <- c(2,2)
best <- nlm(f=ssq,p=pars,typsize=magnitude(pars),indat=alldat)
outfit(best,backtran=FALSE) #a=1.3134, b=2.2029 ssq=571.5804
```

## pkgfuns *pkgfuns names all functions within a package*

### Description

pgkfuns when given the name of a loaded library gives the names of all functions within that library sorted in alphebetical order.

### Usage

```
pkgfuns(packname)
```

### Arguments

| | |
|---|---|
| packname | the name of the package as character |

### Value

a character vector containing the names of all functions in the named package

### Examples

```
## Not run:
  pkgfuns("graphics")
  pkgfuns("rutilsMH")

## End(Not run)
```

## plotoblong *plotoblong generates an oblong from x0,x1,y0,y1*

### Description

plotoblong generates an oblong from x0,x1,y0,y1

### Usage

```
plotoblong(x0, x1, y0, y1, border = 1, col = 0, lwd = 1)
```

### Arguments

| | |
|---|---|
| x0 | x-axis left |
| x1 | x-axis right |
| y0 | yaxis bottom |
| y1 | yaxis top |
| border | colour of the border, default=black=1 |
| col | colour of fill, default = 0 = empty |
| lwd | width of the line,default=1 |

## Value

nothing but it plots a polygon

## Examples

```
## Not run:
  canvas()
  plotoblong(1,50,1,50,lwd=3,linecol=4)

## End(Not run)
```

---

`pol2cart` *pol2cart polar to cartesian coordinates*

---

## Description

pol2cart translate polar coordinates of angles (as degrees) and a distance = radius, into cartesian coordinates of x and y. The option of using arbitrary origin coordinates is included

## Usage

```
pol2cart(angle, dist, xorig = 0, yorig = 0)
```

## Arguments

| | |
|---|---|
| `angle` | the angle in degrees, either a single number of a vector |
| `dist` | the length of the line or radius, a single number |
| `xorig` | the final xorigin |
| `yorig` | the final yorigin |

## Value

a matrix of 1 or more rows depending on length of angle

## Examples

```
## Not run:
  ans <- pol2cart(angle=seq(0,360,15),dist=20,xorig=30,yorig=30)
  print(ans)

## End(Not run)
```

## printV *printV returns a vector cbinded to 1:length(invect)*

### Description

printV takes an input vector and generates another vector of numbers 1:length(invect) which it cbinds to itself. This is primarily useful when trying to print out a vector which can be clumsy to read when print across the screen. applying printV leads to a single vector being printed down the screen

### Usage

```
printV(invect, label = c("value", "index"))
```

### Arguments

| | |
|---|---|
| invect | the input vector to be more easily visualized, this can be numbers, characters, or logical. If logical the TRUE and FALSE are converted to 1's and 0's |
| label | the column labels for vector, default is index and value |

### Value

a dataframe containing the vector 1:length(invect), and invect.

### Examples

```
## Not run:
vec <- rnorm(10,mean=20,sd=2)
printV(vec)
vec <- letters
printV(vec)
vec <- c(TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE,TRUE,TRUE)
printV(vec,label=c("index","logicstate"))

## End(Not run)
```

## properties *properties - used to check a data.frame before standardization*

### Description

properties - used to check a data.frame before standardization. The maximum and minimum are constrained to four decimal places. It allows for columns of NAs and for Posix columns. In case one uses tibbles this function now checks and internally changes the indat into a strict data.frame. This will not influence the external use but it does allow the properties to be obtained.

### Usage

```
properties(indat, dimout = FALSE)
```

### Arguments

| | |
|---|---|
| indat | the data.frame containing the data fields to be used in the subsequent standardization. It tabulates the number of NAs and the number of unique values for each variable and finds the minimum and maximum of the numeric variables |
| dimout | determines whether or noth the dimensions of the data.frame are printed to the screen or not; defaults to FALSE |

### Value

a data.frame with the rows being each variable from the input input data.frame and the columns being the number of NAs, the number of unique values, and minimum and maximum (where possible).

### Examples

```
## Not run:
 data(abdat)
 properties(abdat$fish)

## End(Not run)
```

---

## pythag *pythag calculates Pythagorus' theorum on a vector of two values*

---

### Description

pythag Pythagorus' theorum states that the length of the hypotheneuse between two lines at right angels to each other (that is in cartesian coordinates) is the sqrt of the sum of their squares.

### Usage

```
pythag(x)
```

### Arguments

| | |
|---|---|
| x | a vector of two numbers or a matrix of pairs of numbers |

### Value

a single number or a vector depending on input

### Examples

```
## Not run:
 pythag(c(3,4))  # should be 5
 dat <- matrix(c(3,4,5,7),nrow=2,ncol=2,byrow=TRUE)
 print(dat)
 pythag(dat)     # should be 5 and 10

## End(Not run)
```

## quants *quants used in apply to estimate quantiles across a vector*

### Description

quants used in 'apply' to estimate quantiles across a vector

### Usage

```
quants(invect, probs = c(0.025, 0.05, 0.5, 0.95, 0.975))
```

### Arguments

| | |
|---|---|
| invect | vector of values |
| probs | the quantiles wanted in the outputs; default = c(0.025,0.05,0.5,0.95,0.975) |

### Value

a vector of the c(0.025,0.05,0.5,0.95,0.975) quantiles or whatever is input to probs

### Examples

```
## Not run:
 x <- runif(1000)
 quants(x)
 quants(x,probs=c(0.075,0.5,0.925))

## End(Not run)
```

## removeEmpty *removeEmpty removes empty strings from a vector of strings*

### Description

removeEmpty removes empty strings from a vector of strings. Such spaces often created by spurious commas at the end of lines. It also removes strings made up only of spaces and removes spaces from inside of inidivdual chunks of text.

### Usage

```
removeEmpty(invect)
```

### Arguments

| | |
|---|---|
| invect | vector of input strings, possibly containing empty strings |

### Value

a possibly NULL vector of strings

### Examples

```
## Not run:
x <- c("1","","2","","   ","3"," ","4","","a string","end")
x
length(x)
length(removeEmpty(x))
removeEmpty(x)

## End(Not run)
```

---

## revsum *revsum generates a vector of the cumulative sum from n to 1*

---

### Description

revsum generates a vector of the cumulative sum of an input vector from n to 1 rather than from 1 - n, as in cumsum.

### Usage

```
revsum(x)
```

### Arguments

| | |
|---|---|
| x | an input vector |

### Value

a vector of cumulative values from n to 2

### Examples

```
x <- c(1,2,3,4,5)/15
print(round(cbind(x,cumsum(x),revsum(x)),3))
```

---

## rmdcss *rmdcss generates some initial css style code for HTML Rmd files*

---

### Description

rmdcss generates some initial css style code for HTML Rmd files as well as a mathjax script that will generate equation numbers for any display equations in the document. This prints the css style code and the mathjax script to the console from where it should be pasted into the Rmd file immediately following the YAML header. It now contains font sizes for the h1 heaqder and the .inline and .display math classes

### Usage

```
rmdcss()
```

### Value

nothing but it prints css style code and a mathjax script to the console

### Examples

```
rmdcss()
```

---

## setuprmd *setuprmd sets up and Rmd file ready to generate an HTML file*

---

### Description

setuprmd sets up a custom Rmd file for generating an HTML file, which better suits my own preferences

### Usage

```
setuprmd(filen = "")
```

### Arguments

| | |
|---|---|
| filen | the full path filename for the final Rmd file. Ensure its filetype = .Rmd. The default = "", which write the custom text to the console. |

### Value

nothing but it does write a file to one's hard drive in the location listed in filen

### Examples

```
setuprmd(filen="")
```

---

## splitDate *splitDate - Generates a vector of date and time components*

---

### Description

splitDate - Generates a vector of date and time components, perhaps for inclusion in filenames or other labels; helpful for keeping different run outputs seperate and identifiable.

### Usage

```
splitDate(dat = NA)
```

### Arguments

| | |
|---|---|
| dat | • a system time from Sys.time() to be broken in components; defaults to NA, whereupon the current time is used. |

## Value

a vector od characters relating to 'Year', 'Month', 'Day','Time', and a DateTime, which is a combination of all of these suitable for inclusion in a filename.

## Examples

```
## Not run:
tmp <- splitDate()
print(tmp)
print(names(tmp))
print(as.numeric(tmp[1:3]))
print(tmp["DateTime"])

## End(Not run)
```

---

`str1` *str1 a simple replacement for str(x,max.level=1)*

---

## Description

str1 an abbreviated replacement for str(x,max.level=1), which I put together because to often I make a typo when typing out the full str syntax. Hence I find str1 helpful

## Usage

```
str1(x)
```

## Arguments

| | |
|---|---|
| `x` | the object whose structure is to be listed |

## Value

str(x,max.level=1)

## Examples

```
x <- matrix(rnorm(25,mean=5,sd=1),nrow=5,ncol=5)
str1(x)
```

## str2 *str2 a simple replacement for str(x,max.level=2)*

### Description

str2 an abbreviated replacement for str(x,max.level=2), which I put together because to often I make a typo when typing out the full str syntax. For when str1 is not detailed enough.

### Usage

```
str2(x)
```

### Arguments

| | |
|---|---|
| x | the object whose structure is to be listed |

### Value

str(x,max.level=2)

### Examples

```
x <- matrix(rnorm(25,mean=5,sd=1),nrow=5,ncol=5)
str2(x)
```

## tidynames *tidynames can replace awkward data.frame names with better ones*

### Description

tidynames can replace awkward or overly long data.frame column names with better ones that are easier to use. It also permits one to maintain the same set of column names within an analysis even when the source data.frame includes alterations.

### Usage

```
tidynames(columns, replace, repwith)
```

### Arguments

| | |
|---|---|
| columns | the vector of names that should include the ones to be altered |
| replace | the names to be changed, as a vector of character strings |
| repwith | the replacement names as a vector of character strings |

### Value

a vector of new columns names

### Examples

```
print("wait")
```

## toXL *toXL copies a data.frame or matrix to the clipboard*

### Description

toXL copies a data.frame or matrix to the clipboard so one can then switch to Excel and just type ctrl + V to paste the data in place

### Usage

```
toXL(x, output = FALSE)
```

### Arguments

| | |
|---|---|
| x | a vector or matrix |
| output | • a boolean determining whether to print the object to the screen as well as the clipboard; defaults to FALSE |

### Value

Places the object 'x' into the clipboard ready for pasting

### Examples

```
datamatrix <- matrix(data=rnorm(100),nrow=10,ncol=10)
colnames(datamatrix) <- paste0("A",1:10)
rownames(datamatrix) <- paste0("B",1:10)
toXL(datamatrix,output=TRUE)
```

## which.closest *which.closest find the number closest to a given value*

### Description

which.closest finds either the number in a vector which is closest to the input value or its index value

### Usage

```
which.closest(x, invect, index = T)
```

### Arguments

| | |
|---|---|
| x | the value to lookup |
| invect | the vector in which to lookup the value x |
| index | should the closest value be returned or its index; default=TRUE |

### Value

by default it returns the index in the vector of the value closest to the input value

## Examples

```
## Not run:
vals <- rnorm(100,mean=5,sd=2)
pick <- which.closest(5.0,vals,index=TRUE)
pick
vals[pick]
which.closest(5.0,vals,index=FALSE)

## End(Not run)
```

---

`wtedmean` *wtedmean calculates the weighted mean of a set of values and weights*

---

## Description

wtedmean solves the problem of calculating a weighted mean value from a set of values with different weights. Within the aMSE this is common when trying to summarize across populations within an SAU or summarize SAU within a zone by finding a mean value weighted by the respective catch from each related population or SAU.

## Usage

```
wtedmean(x, wts)
```

## Arguments

| | |
|---|---|
| `x` | the values whose weighted mean is wanted |
| `wts` | the weights to use, often a set of catches |

## Value

a single real number

## Examples

```
saucpue <- c(91.0,85.5,88.4,95.2)
saucatch <- c(42.0,102.3,75.0,112.0)
wtedmean(saucpue,saucatch)
saucatch/sum(saucatch)  # the relative weights
```

`%ni%` *'%ni%' identifies which element in x is NOT in y*

## Description

'%ni%' identifies which element in x is NOT in y

## Usage

```
x %ni% y
```

## Arguments

| | |
|---|---|
| x | a vector of elements which can be numeric or character |
| y | a vector of elements which can be numeric or character |

## Examples

```
x <- 1:10
y <- 6:18
x %ni% y
pick <- (x %ni% y)
x[pick]
```

# Index