Package 'BurStMisc'

August 13, 2016

Type Package

Title Burns Statistics Miscellaneous

Version 1.1

Author Pat Burns

Maintainer Pat Burns <patrick@burns-stat.com>

Description Script search, corner, genetic optimization, permutation tests, write expect test.

Suggests testthat

License Unlimited

LazyData TRUE

NeedsCompilation no

Repository CRAN

Date/Publication 2016-08-13 13:56:08

R topics documented:

Cfrag.list *Write a Fragment of a C Program*

Description

Writes either the C declaration of items in a list, or initializes them with the contents of the list.

Usage

```
Cfrag.list(x, file = NULL,  item.num = c(3, 10, 5),
indent = c("\t", "\t\t"), declaration.only = FALSE,
long = FALSE, append = FALSE)
```

Arguments

x	required. A list with names.
file	either NULL or a character string. If NULL (or an empty string), then a vector of characters is returned.
item.num	length three vector giving the number of items per line for doubles, integers and characters.
indent	length two vector giving the amount to indent declarations and the items in the initialization.
declaration.only	logical flag. If TRUE, then the variables are declared but not given initial values.
long	logical flag. If TRUE, then integers are declared to be "long". If FALSE, then integers are declared to be "int".
append	logical flag. If TRUE, then the file is appended to; otherwise it is overwritten if it exists.

Value

if file is a non-empty character string, then the name of the file that is written. Otherwise, a character vector of the declarations – each element representing a different line.

Side Effects

if file is a non-empty character string, then the file is created, overwritten or appended.

Details

The type to declare is dependent on the storage mode of the component of x. You may need to coerce components to get them to be declared the correct type.

Note

Rcpp now probably makes most uses of this function obsolete.

See Also

.C, storage.mode, as.double, as.integer, as.character, cat.

Examples

```
test.list <- list(adoub=as.double(rnorm(20)), anint=as.integer(92:109),
achar=c("aaa", "bbbb", "ccccc"))

Cfrag.list(test.list, file="")

## Not run:
Cfrag.list(test.list, file="test.c")

Cfrag.list(test.list[1], file="test.c", dec=TRUE)
Cfrag.list(test.list[-1], file="test.c", dec=FALSE, append=TRUE)

## End(Not run)
```

---

corner *Portion of an Array or Data Frame*

Description

Returns an array (or data frame) of the same number of dimensions as the input, but generally with smaller dimensions.

Usage

```
corner(x, corner = "tlffff", n = 6)
```

Arguments

x	an array (which includes matrices) or data frame.
corner	a single character string which codes the particular corner of the array that is desired. The first character is either "t" (top) or "b" (bottom). The second character is either "l" (left) or "r" (right). Remaining characters should be either "f" (front) or "b" (back).
n	a vector of positive integers which generally should have length at least that of the dim of the array – it is replicated if not. This states (the maximum of) each of the dimensions of the result.

Value

an array similar to the input x, but with smaller dimensions in general.

See Also

head, tail.

Examples

```
corner(freeny.x)
corner(freeny.x, "br", c(2,3))

# with three-dimensional arrays:
corner(iris3)
corner(array(1:1000, rep(10, 3)), "brf", 2:4)
```

---

genopt *Genetic Optimization*

Description

Approximately minimizes the value of a function using a simple heuristic optimizer that uses a combination of genetic and simulated annealing optimization.

Usage

```
genopt(fun, population, lower = -Inf, upper = Inf,
       scale = dcontrol["eps"], add.args = NULL,
       control = genopt.control(...), ...)
```

Arguments

`fun`	function that returns a numerical value.
`population`	a matrix or a list. If a matrix, then the rows correspond to the parameters and the columns are different parameter vectors. If a list, then it should have a component named "population" and optionally a component named "objective" – in particular, it can be the result of a call to `genopt`. If the "objective" component exists, then it is assumed to be the function values of the population and the function will not be evaluated on the members of the population.
`lower`	vector giving the lower bound for parameter values. This is replicated to be as long as the number of parameters.
`upper`	vector giving the upper bound for parameter values. This is replicated to be as long as the number of parameters.
`scale`	vector of scales to use when doing local search with a solution. This is replicated to be as long as the number of parameters.
`add.args`	list of additional arguments to `fun`.
`control`	an object like the output of `genopt.control` containing control parameters for the optimization.
`...`	arguments for `genopt.control` if the `control` argument is not given explicitly.

Details

There is a summary method for class genopt which shows the call, a summary of the set of objectives found, and the best solution (set of parameters).

Value

a list of class genopt with the following components:

population	a matrix of the same size as the input population matrix, but generally with different values in it.
objective	a vector with length equal to the number of columns of population containing the function value for the corresponding parameter vector in population.
funevals	the number of function evaluations performed. If population is given as a list that contains a "funevals" component, then that number is added to the number of evaluations done in the current call.
random.seed	the random seed at the start of the call – given so that you can reproduce the computations.
call	an image of the call that created this object.

References

The original version of this function appeared in "S Poetry".

See Also

genopt.control, summary.genopt.

Examples

```
# two parameters, population size 5
go1 <- genopt(function(x, other) sum(x, other),
              population=matrix(rexp(10), nrow=2, ncol=5),
              lower=0, add.arg=list(other=3), trace=FALSE)
summary(go1)

go2 <- genopt(function(x, other) sum(x, other), population=go1,
              lower=0, add.arg=list(other=3), trace=FALSE)
```

---

genopt.control *Control parameters for genopt*

Description

Returns a list suitable as the control argument of the genopt function.

Usage

```
genopt.control(births = 100, random.n = 0, jitters.n = 3,
               trace = TRUE, eps = 0.1, prob = 0.4, scale.min = 1e-12,
               maxeval = Inf)
```

Arguments

births	the number of times two parents are combined to create a new solution.
random.n	the number of random solutions created (and evaluated) before the genetic phase starts.
jitters.n	the number of times a newly successful solution is changed (in the simulated annealing phase) in an attempt to find a nearby better one.
trace	logical value: should progress be printed?
eps	the default value for the scale of the jittering in simulated annealing.
prob	in the selection of parameters between two parents, the probability of each coming from the first parent.
scale.min	the minimum value allowed in the scale argument of genopt.
maxeval	the maximum number of function evaluations allowed. This takes previous function calls into account if funevals is a component of the population argument of genopt.

Value

a list with components:

icontrol	vector of the control parameters that are logically integer (or logical).
dcontrol	vector of the control parameters that are logically real-valued.

See Also

genopt.

Examples

```
goc1 <- genopt.control(random.n=200, births=1000, trace=FALSE)

go1 <- genopt(function(x, other) sum(x, other),
              matrix(rexp(10), nrow=2, ncol=5), lower=0, add.arg=list(other=3),
              control=goc1)

goc2 <- genopt.control(random.n=0, births=1000, trace=FALSE)

go2 <- genopt(function(x, other) sum(x, other), go1,
              lower=0, add.arg=list(other=3), control=goc2)
```

ntile *Membership of ntile groups*

Description

Creates groups where the groups each have as close to the same number of members as possible.

Usage

```
ntile(x, ngroups, na.rm = FALSE, result = "list",
    reverse = FALSE, checkBleed = TRUE)
```

Arguments

x	a numeric vector.
ngroups	a positive integer giving the number of groups to create.
na.rm	logical value: if TRUE, then missing values are removed before forming the groups.
result	a character string specifying the form of the resulting value. This must be (an abbreviation of) one of: "list", "numeric", "factor"
reverse	logical value: if TRUE, then the biggest values are in group 1.
checkBleed	logical value: if TRUE, then a warning is issued if the same value appears in more than one group.

Value

the form of the value depends on the result argument.

The "list" result has the values of the input x grouped into ngroups components.

The "numeric" result is a vector of integers from 1 to ngroups indicating which group the corresponding element of x is in.

The "factor" result is an ordered factor version of the "numeric" result.

Note

A more primitive version of this function appeared in a blog post called "Miles of iles" on the Portfolio Probe website. A bit of comparison with the alternative functions in See Also may be found there.

Author(s)

Pat Burns

See Also

cut, cut2, quantcut.

Examples

```
ntile(setNames(state.area, state.name), 10)

ntile(Loblolly$height, 5, result="factor", reverse=TRUE)

ntile(c(-10:10, rep(0, 7)), 4)
```

`permutation.test.discrete`

Random Permutation Test

Description

Performs a random permutation test on the relationship between two discrete variables, or by using a function.

Usage

```
permutation.test.discrete(x, y = NULL, scores, alternative = "greater",
trials = 1000)
permutation.test.fun(x, y = NULL, fun = function(x, y) sum(x * y),
alternative = "greater", trials = 1000)
```

Arguments

`x`	either a two-column matrix or data frame, or a vector. When this has two columns, `y` should not be given.
`y`	either a zero-length object, or a vector the same length as `x`.
`scores`	a numeric matrix providing the scores for each combination of the unique values in `x` and `y`. The dimnames need to consist of these unique values.
`fun`	a function that takes two arguments and returns a single numeric value.
`alternative`	a character string that partially matches either `"greater"` or `"less"`.
`trials`	the number of random permutations to be performed.

Value

an object of class `permtstBurSt` which is a list with the following components:

`original.score`	the score (or function value) produced by the original data.
`perm.scores`	a vector of the scores (or function values) from the random permutations.
`stats`	a numeric vector that has the number of observations in the data, the number of random permutations done, the number of permutations that produced a score at least as extreme as the original, and the p-value for the test.
`alternative`	either `"greater"` or `"less"`.
`random.seed`	the random seed at the start of the call.
`call`	an image of the call that was used.

Side effects

The object `.Random.seed` is either created or updated.

Details

There are `print` and `plot` methods for this class of object. The print method merely describes the object and shows the p-value of the test (rounded, by default, to 4 digits).

A simple version of the p-value is the number of random permutations that are at least as extreme as the original divided by the total number of random permutations. The value computed, which is more correct, has 1 added to both numerator and denominator. In general the difference is of no consequence. However, there are cases where it does matter, for example when independent p-values are combined.

These functions are related to "Permuting Super Bowl Theory" which can be found in the working papers section of http://www.burns-stat.com. The paper explains permutation tests via a discussion of the Super Bowl indicator of the stock market.

See Also

plot.permtstBurSt.

Examples

```
winner <- c('N', 'N', 'A', 'N', 'A', 'N')
market <- c('+', '-', '-', '+', '+', '+')
smat <- diag(2)
dimnames(smat) <- list(c('N', 'A'), c('+', '-'))

pt1 <- permutation.test.discrete(winner, market, smat)
print(pt1)
plot(pt1)

pt2 <- permutation.test.fun(ToothGrowth[, -2], fun=cor)
print(pt2)
plot(pt2)


smat2 <- matrix(c(-3, -.5, 3, -1, 1, 0, 0, 1, -1, 3, -.5, -3),
3, 4, dimnames=list(c('Up', 'Neut', 'Down'),
c('Q1', 'Q2', 'Q3', 'Q4')))
my.results <- data.frame(results=sample(c('Up', 'Neut', 'Down'),
       100, replace=TRUE), quartile=sample(
       c('Q1', 'Q2', 'Q3', 'Q4'), 100, replace=TRUE))
permutation.test.discrete(my.results[, c("results", "quartile")],
score=smat2)
```

`plot.permtstBurSt` *Plot Distribution from a Permutation Test*

Description

Shows the distribution of the scores of the random permutations relative to the score from the actual data.

Usage

```
## S3 method for class 'permtstBurSt'
plot(x, col = c("black", "red"), width = 10,
uniqlim = 10, main = "", xlab = "Scores", ...)
```

Arguments

`x`	an object of class `permtstBurSt`, which is likely the result of: `permutation.test.discrete` or `permutation.test.fun`.
`col`	the colors to use in the plot – mainly for the bar-type plot when there are only a few scores in the permutation distribution. The first color is the one used for the insignificant part of the distribution, the second color is for the significant part of the distribution. The second color is used to identify the original score when a histogram is used.
`width`	the width of the bars.
`uniqlim`	the value that determines which type of plot to use. If the number of unique values in the scores from the permutations is bigger than `uniqlim`, then a histogram is drawn. Otherwise, bars are drawn.
`main`	character string giving the main title for the plot.
`xlab`	character string giving the label for the x-axis.
`...`	additional graphics parameters may be given.

Side effects

a plot is produced.

Details

Two styles of plot are possible. A histogram is more appropriate when there are more than a few values in the permutation distribution. When there are only a few distinct values, then a bar-type plot is more informative. The `uniqlim` argument controls the definition of "a few".

These functions are related to "Permuting Super Bowl Theory" which can be found in the working papers section of http://www.burns-stat.com. The paper explains permutation tests via a discussion of the Super Bowl indicator of the stock market.

See Also

permutation.test.discrete.

Examples

```
## Not run:
pt1 <- permutation.test.discrete(winner, market, smat)
plot(pt1, col=c("blue", "gold"))

## End(Not run)
```

---

scriptSearch *Search R scripts for text*

Description

Returns a list of text matches in the scripts in a directory.

Usage

```
scriptSearch(pattern, path = ".", subdirs = TRUE, suffix = "\\.[rR]$",
      commentsIncluded = FALSE, ..., verbose = FALSE)
```

Arguments

pattern	character string containing a regular expression (this is the argument of the same name for grep).
path	a character string giving the (relative) path of the directory holding the R scripts.
subdirs	logical value: if TRUE, then subdirectories of path are also searched.
suffix	a character string restricting the files to be searched.
commentsIncluded	logical value: if FALSE, then lines beginning with the hash symbol are ignored.
...	additional arguments to grep may be given – the most likely possibilities are ignore.case, perl and fixed.
verbose	logical value: if TRUE, then the names of the scripts being searched are printed. The full name of this argument needs to be given – no abbreviation is allowed.

Details

The default suffix argument restricts the search to R scripts. An R script is considered to be a file that ends in dot-R, where the "R" can be lowercase or uppercase.

Value

a list: the names are equal to the file names containing matches, and each component is a character vector of the whitespace-trimmed lines in the file that match the pattern.

Limitations

Using `commentsIncluded=FALSE` does not preclude a match being declared when it is in a comment not starting the line.

Author(s)

Pat Burns

See Also

`grep`

Examples

```
## Not run:
scriptSearch("list.files")
scriptSearch("garch", "~/../burns-stat3/webpages/blog")

# search for an assignment
# ' *' means zero or more spaces
scriptSearch("specialObject *<- ", "~/myScriptDirectory", subdirs=FALSE)

# search COBOL files
scriptSearch("blah", suffix="\\.cbl$")


## End(Not run)
```

---

summary.genopt *Summary of genopt object*

Description

The call, best solution and summary of objectives in the final population.

Usage

```
## S3 method for class 'genopt'
summary(object, ...)
```

Arguments

`object`	the result of a call to `genopt`.
`...`	currently unused.

Value

a list with components:

call	image of the call to genopt.
summary.objectives	summary of the objectives of the solutions in the final population.
best.solution	the vector of parameters with the best solution in the population.

See Also

genopt.

Examples

```
# two parameters, population size 5
go1 <- genopt(function(x, other) sum(x, other),
              population=matrix(rexp(10), nrow=2, ncol=5),
              lower=0, add.arg=list(other=3), trace=FALSE)
summary(go1)
```

---

writeExpectTest *Text for a testthat test*

Description

Automatically writes the text of a testthat::expect_equal test of an expression.

Usage

```
writeExpectTest(expr, filename = "", ...)
```

Arguments

expr	an expression to be tested.
filename	where should the results go to? The empty string means that it goes to the console.
...	additional arguments to cat may be given to control the output. The most likely choice is append.

Value

NULL

Side effects

text is written to the console or to a file.

Note

Some functions are hard to test, like `scriptSearch`. It seems like this one would be too, but in fact it is self-testing. If the test it writes doesn't pass in the same environment, then it must have done something wrong.

See Also

`expect_equal`.

Examples

```
writeExpectTest(head(1:10))
writeExpectTest(head(cars))
```

Index