# Package 'crank'

October 12, 2022

**Version** 1.1-2

**Title** Completing Ranks

**Date** 2019-04-08

**Author** Jim Lemon <drjimlemon@gmail.com>,

**Maintainer** Jim Lemon <drjimlemon@gmail.com>

**Description** Functions for completing and recalculating rankings and sorting.

**Imports** stats

**License** GPL (>= 2)

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2019-04-09 04:23:01 UTC

## R topics documented:

`cats2ranks` *Ordered option selections to ranks*

## Description

Convert ordered option selections to ranks, assigning the mean of unused ranks to unselected options.

## Usage

```
cats2ranks(x,cats=NULL)
```

## Arguments

| | |
|---|---|
| `x` | A matrix or data frame of numeric or characters labels for options. Rows are interpreted as cases or respondents and columns are interpreted as the order of option selections, beginning with the highest ranking (usually something like "Most important") and descending. |
| `cats` | The range of numbers that represent options. The default is the vector of unique entries in '`x`'. |

## Details

'`cats2ranks`' converts ordered option selections to mean ranks. It is useful in the situation where a respondent is asked to select one of a number of options as the most important, another as the second most important, and so on. It counts the number of times each option code appears in each column and calculates the mean ranking of options. It is expected that there will be fewer selections available than there are options, thus creating the opportunity for biased rankings. This can occur when one or more options are not commonly chosen, but are given extreme (usually high) ranks when they are. The function calculates the mean of unallocated ranks and assigns this to all options not chosen by each respondent, correcting for this bias. The correction assumes that the respondent does not differentiate between unranked options, but these are all ranked lower than the options selected.

'`cats2ranks`' is especially useful when respondents do not select the same number of options. The mean of unallocated ranks is calculated for each respondent so that all options are entered into the calculation of mean ranks.

Note that '`cats2ranks`' interprets each value in '`x`' as a nominal level variable and its column index as the rank, while '`meanranks`' interprets values as ordinal level (ranks). Thus if a matrix or data frame of ranks is passed to '`cats2ranks`', it will not give the correct mean ranks or relative positions.

## Value

A list with four components:

| | |
|---|---|
| `ranks` | The matrix of completed ranks. |

| | |
|---|---|
| cats | The vector of options as passed or calculated. |
| ranksum | The sum of ranks for each option. |
| rankcount | The number of times each option was selected. |

## Author(s)

Jim Lemon

## See Also

muranks,meanranks

## Examples

```
# first a standard 1:m numerically coded selection
opchoice<-matrix(NA,nrow=40,ncol=5)
for(i in 1:40) opchoice[i,]<-sample(1:10,5)
opchoice
cats2ranks(opchoice)
# now a messy character choice with missing values
opchoice<-matrix(NA,nrow=40,ncol=5)
tencolors<-c("red","green","blue","yellow","magenta","cyan",
 "purple","orange","brown","pink")
for(i in 1:40) {
 nchoices<-sample(3:5,1)
 opchoice[i,1:nchoices]<-sample(tencolors,nchoices)
}
opchoice
cats2ranks(opchoice)
```

---

# elrepos *Move the position of an element in a vector.*

---

## Description

Move the position of an element in a vector.

## Usage

```
elrepos(x,i1,i2)
```

## Arguments

| | |
|---|---|
| x | A vector of unique values. |
| i1,i2 | The position (i1) in the vector of an element that should be ahead of the element in position i2. |

## Details

'elrepos' saves the element in position i1 of the vector x and removes that element from x. It then inserts the element that was in position i1 just before the element in position i2.

## Value

The vector 'x' with the position of one element changed as above.

## Note

Currently this function is only useful to perform the position changing for the function 'spsort'.

## Author(s)

Jim Lemon

## See Also

spsort

## Examples

```
x<-unlist(strsplit("lemon",""))
y<-elrepos(x,3,1)
z<-elrepos(y,2,1)
paste0(z,collapse="")
```

---

# fillArow *Impute a row of ranks using the existing values of rankings*

---

## Description

Imputes a row of missing ranks using the Lim-Wolfe procedure

## Usage

```
fillArow(x,ranksums=NA,Arow,maxcon=TRUE)
```

## Arguments

| | |
|---|---|
| x | A matrix of ranks that may contain ties and NAs. Columns represent objects ranked and rows represent ranking methods. |
| ranksums | The sums of ranks of all complete rows in 'x'. |
| Arow | The row of 'x' that is to be completed. |
| maxcon | Whether to impute rankings maximally consistent with the existing ones (TRUE) or minimally consistent (FALSE). |

## Details

'fillArow' imputes missing ranks in the row designated by 'Arow' using the information in 'ranksums'. If the ranks already completed provide information on the order of imputation, that is used directly for imputed ranks of maximal consistency or inversely for imputed ranks of minimal consistency. If the existing ranks do not provide such information, the missing ranks are permuted, and a list of matrices with all the permutations is substituted. This may involve a recursive call to 'fillArow' and produce a nested list of matrices. See Lim and Wolfe (2002) for details of this process.

## Value

The matrix 'x' with row 'Arow' completed or a list of such matrices, possibly nested.

## Author(s)

Jim Lemon

## References

Lim, D.H. & Wolfe, D.A. (2002) An efficient alternative to average ranks for testing with incomplete ranking data. Biometrical Journal, 43(2): 187-206.

## See Also

lwscreen, listBuilder, fillArows

## Examples

```
# The first example matrix from Lim and Wolfe (2002)
lwmat<-matrix(c(3,1,2,4,NA,2,1,NA,2,NA,1,NA),nrow=3,byrow=TRUE)
# complete the second row with maximal consistency
fillArow(lwmat,lwmat[1,],2)
# now with minimal consistency
fillArow(lwmat,lwmat[1,],2,maxcon=FALSE)
```

---

# fillArows *Impute ranks using the existing values of rankings*

---

## Description

Imputes missing ranks using the Lim-Wolfe procedure

## Usage

```
fillArows(x,maxcon=TRUE)
```

## Arguments

| | |
|---|---|
| x | A matrix of ranks that may contain ties and NAs. Columns represent objects ranked and rows represent ranking methods. |
| maxcon | Whether to impute rankings maximally consistent with the existing ones (TRUE) or minimally consistent (FALSE). |

## Details

'fillArows' imputes missing ranks by examining the completed ranks for each set of rows that have the same number of missing ranks. If more than one row has the minimum number of missing values, the order of these rows is permuted and the matrix 'x' becomes a list of matrices in which the values in the rows will be imputed in different orders. Another level of permutation and multiplication of matrices may occur in 'fillArow' to which the matrices are passed for the actual imputation. The function 'getLWargs' is called to get the arguments for 'fillArow'. See Lim and Wolfe (2002) for details of this process.

## Value

A list of one or more completed matrices of ranks, possibly nested.

## Author(s)

Jim Lemon

## References

Lim, D.H. & Wolfe, D.A. (2002) An efficient alternative to average ranks for testing with incomplete ranking data. Biometrical Journal, 43(2): 187-206.

## See Also

lwscreen, getLWargs, fillArow

## Examples

```
# The first example matrix from Lim and Wolfe (2002)
lwmat<-matrix(c(3,1,2,4,NA,2,1,NA,2,NA,1,NA),nrow=3,byrow=TRUE)
# complete with maximal consistency, permuting row order
fillArows(lwmat)
# now with minimal consistency as above
fillArows(lwmat,maxcon=FALSE)
```

getLWargs *Get the information about a matrix of ranks.*

## Description

Get the information required for imputing missing ranks.

## Usage

```
getLWargs(x)
```

## Arguments

| | |
|---|---|
| x | A matrix of ranks, usually containing missing values. |

## Details

'getLWargs' calculates the information required for 'fillArows' and 'fillArow' to impute the missing ranks in a matrix.

## Value

A list containing the following:

| | |
|---|---|
| ranksums | The column sums of the complete rows of the matrix. |
| Arows | The indices of the row(s) with the minimal number of missing values. |
| nArows | The number of Arows. |
| Brows | The indices of the complete rows. |

## Author(s)

Jim Lemon

## See Also

listBuilder

## Examples

```
# The first example matrix from Lim and Wolfe (2002)
lwmat<-matrix(c(3,1,2,4,NA,2,1,NA,2,NA,1,NA),nrow=3,byrow=TRUE)
getLWargs(lwmat)
```

`listBuilder` *Build a possibly nested list.*

## Description

Build a possibly nested list using the result of a function.

## Usage

```
listBuilder(x,FUN=NULL,fargs=NULL)
```

## Arguments

| | |
|---|---|
| `x` | The object that will be the first argument of '`FUN`', or a possibly nested list of such objects. |
| `FUN` | A function that can accept '`x`' as its first argument. |
| `fargs` | A list of the remaining arguments to '`FUN`'. |

## Details

'`listBuilder`' descends the list structure of '`x`' if it is a list until it encounters a non-list element. It then passes that element as the first argument to '`FUN`' and returns the value of '`FUN`'. This may be a list of elements, replacing the original element, hence the name.

## Value

If '`x`' is not a list and '`FUN`' is NULL, '`x`' is returned. If '`FUN`' creates a list from one or more elements of '`x`', a list or nested list will be returned. Successive calls to '`listBuilder`' can rapidly create very large, deeply nested list structures.

## Author(s)

Jim Lemon

## See Also

list

## Examples

```
# define a function that splits a vector into a list
splitvec<-function(x) {
 xlen<-length(x)
 if(xlen > 1) {
  newx<-vector("list",xlen)
  for(newlist in 1:xlen) newx[[newlist]]<-x[newlist]
  return(newx)
 }
```

```
 return(x)
}
testlist<-list(c(9,16),list(25,c(36,49)))
listBuilder(testlist,splitvec)
```

---

`listCrawler` *Descend a list, applying a function to each element.*

---

## Description

Descend a possibly nested list, seeking the element that has the extreme value of a function.

## Usage

```
listCrawler(x,FUN=NULL,maxval=TRUE,
 retval=list(indx=vector("numeric",0),element=NULL,value=NA))
```

## Arguments

| | |
|---|---|
| `x` | The object that will be the first argument of '`FUN`', or a possibly nested list of such objects. |
| `FUN` | A function that can accept '`x`' as its first argument. |
| `maxval` | Whether to look for maximal (TRUE) or minimal (FALSE) values of the function '`FUN`'. |
| `retval` | The list that is eventually returned. |

## Details

'`listCrawler`' descends the list structure of '`x`' applying '`FUN`' to any non-list elements it encounters. If the value of '`FUN`' is larger or smaller than the current extremum (depending upon the value of '`maxval`'), the new value becomes the current extremum. The default value of '`FUN`' just takes the value of the elements.

## Value

A list containing:

| | |
|---|---|
| `indx` | the indices of the element producing the extreme value of '`FUN`'. |
| `element` | The element that produced the extremum. |
| `value` | The extreme value of '`FUN`'. |

## Author(s)

Jim Lemon

## See Also

list, listBuilder

## Examples

```
# a simple example using the square root function
testlist<-list(list(9,16),list(25,list(36,49)))
# first get the default maximum
listCrawler(testlist,sqrt)
# then the minimum
listCrawler(testlist,sqrt,maxval=FALSE)
```

---

`lw.FriedmanTest` *Wrapper for the Friedman test function.*

---

## Description

Wrapper for the Friedman test function.

## Usage

```
lw.FriedmanTest(x)
```

## Arguments

| | |
|---|---|
| x | A matrix of ranks. |

## Details

Calls '`friedman.test`' and returns a vector containing the statistic and p value.

## Value

The statistic and p value returned by '`friedman.test`'.

## Author(s)

Jim Lemon

## See Also

friedman.test

`lwscreen` *Impute ranks using the existing values of rankings*

## Description

Completes a matrix with missing ranks for the values maximally and minimally consistent with existing values using the Lim-Wolfe procedure

## Usage

```
lwscreen(x,scrtest="lw.FriedmanTest")
```

## Arguments

| | |
|---|---|
| `x` | A matrix of ranks that may contain ties and NAs. Columns represent objects ranked and rows represent ranking methods. |
| `scrtest` | What test to use to determine the maximally and minimally consistent imputed values. |

## Details

'`lwscreen`' calls '`fillArows`' to impute the missing ranks in the matrix '`x`'. It then applies '`scrtest`' to all the matrices returned and finds the minimum and maximum values. See Lim and Wolfe (2002) for details of the algorithm.

The algorithm for finding the maximally consistent and inconsistent rank imputations is extremely computer intensive, creating large numbers of permuted matrices when tied ranksums or multiple rows with the same number of missing values are encountered. The APA election example in Lim and Wolfe (2002) is beyond the capability of the average PC in the present implementation.

## Value

The maximal and minimal statistics and p values for the list of completed rank matrices obtained.

## Author(s)

Jim Lemon

## References

Lim, D.H. & Wolfe, D.A. (2002) An efficient alternative to average ranks for testing with incomplete ranking data. Biometrical Journal, 43(2): 187-206.

## See Also

lw.FriedmanTest, listBuilder, fillArows

## Examples

```
# The first example matrix from Lim and Wolfe (2002)
lwmat<-matrix(c(3,1,2,4,NA,2,1,NA,2,NA,1,NA),nrow=3,byrow=TRUE)
lwscreen(lwmat)
```

---

`meanranks` *Calculate mean ranks with possible missing values*

---

## Description

Calculates mean ranks where some ranks may be missing

## Usage

```
meanranks(x,allranks=NULL,labels=NULL,rankx=FALSE)
```

## Arguments

| | |
|---|---|
| `x` | A matrix of ranks that may contain ties and NAs. Objects ranked are assumed to be columns and ranking methods rows. |
| `allranks` | An optional list of all ranks that might have been made. |
| `labels` | Optional labels for the ranks. |
| `rankx` | Whether to convert competition ranks, or any other set of numeric values, into the usual mean rankings for ties. |

## Details

'`meanranks`' calls '`muranks`' to complete the rank matrix before calculating the mean ranks for each column if there are any NAs in '`x`'.

Note that '`cats2ranks`' interprets each value in '`x`' as a nominal level variable and its index as the rank, while '`meanranks`' interprets values as ordinal level (ranks). Thus if a matrix or data frame of category labels is passed to '`meanranks`', it will not give the correct mean ranks.

## Value

A list with the following components:

| | |
|---|---|
| `ranks` | '`x`' with any NAs replaced by the mean of unallocated ranks for each row. |
| `labels` | The vector of labels, defaulting to the integers 1:allranks. |
| `mean.ranks` | A vector of mean ranks for each value of allranks. |

## Author(s)

Jim Lemon

## See Also

muranks, rank, cats2ranks

## Examples

```
# simulate "best/worst" ranking
x<-matrix(NA,nrow=10,ncol=10)
for(i in 1:10) {
 nbest<-sample(2:5,1)
 best<-1:nbest
 nworst<-sample(1:5,1)
 worst<-(11-nworst):10
 rankpos<-sample(1:10,nbest+nworst)
 x[i,rankpos]<-c(best,worst)
}
x
meanranks(x)
```

---

`muranks` *Complete a matrix of rankings*

---

## Description

Fills an incomplete matrix of rankings with means of unallocated ranks

## Usage

```
muranks(x,allranks=NULL,rankx=FALSE)
```

## Arguments

| | |
|---|---|
| `x` | A vector or matrix of rankings that may contain ties and NAs. Objects ranked are assumed to be columns and ranking methods rows. |
| `allranks` | An optional list of all ranks that might have been allocated. Defaults to the unique values in 'x'. |
| `rankx` | Whether to apply the 'rank' function (see Details). |

## Details

'muranks' assumes that the values in 'x' are rankings with values in the set 'allranks' or if that is NULL, between 1 and the number of columns or values in 'x'. If any values in 'x' are outside this range, or if the missing ranks are not sequential, the function will drop that row with a warning.

For each row, the function finds the mean of those ranks in 'allranks' that were not allocated and substitutes that value for any missing values in the row.

If 'rankx' is TRUE, each row is passed to 'rank'. This will convert competition ranks or any set of numbers to the usual mean rankings. This will also override the rejection of rows in which the missing ranks are not sequential, and may produce counterintuitive imputed ranks.

## Value

A matrix similar to 'x' in which any NAs are replaced by the mean of unallocated ranks for each row.

## Note

'muranks' will impute ranks for "best/worst" ranking, where the method (rater) allocates the highest ranks to the most preferred data objects and the lowest ranks to the least preferred. The mean of all unallocated ranks is imputed for unranked data objects. It is assumed that unranked data objects are considered less preferred than those allocated high ranks, more preferred than those allocated low ranks, and not differentiated from each other. If this assumption is not satisfied, 'muranks' will warn the operator that one or more rows have been dropped. To explain this behavior, consider the case in which a method allocates the ranks 1,2,3,5,7,8 to eight data objects. Two ranks have not been allocated, 4 and 6. It would be possible to impute the mean, 5, to both, but this ignores the implicit information that the two data objects were differentiated by the rank 5, which is "between" them. Only in the unlikely case that both were considered equivalent to the object ranked 5 would this be correct, as there is no way to establish which was more or less preferred. The operator should be aware that if 'rankx' is TRUE, the unranked objects will be allocated the lowest ranks, which is unlikely to be correct.

## Author(s)

Jim Lemon

## See Also

meanranks,rank

## Examples

```
# simulate ranking from the top with variable completion
x<-matrix(NA,nrow=10,ncol=10)
for(i in 1:10) {
 nx<-sample(2:10,1)
 xx<-sample(1:10,nx)
 x[i,xx]<-1:nx
}
x
muranks(x)
```

---

# page.trend.test *Page test for ordered alternatives*

---

## Description

calculates the Page test for ordered alternatives.

## Usage

```
page.trend.test(x,ranks=TRUE)
```

## Arguments

| | |
|---|---|
| x | a 2D matrix of ranks or observations. |
| ranks | Whether the values in x are ranks or observations. |

## Details

'page.trend.test' will accept a matrix of ranks where the rows represent methods (usually raters) and the columns represent related data objects. It apparently handles ties, but not missing values. For small values of k (methods) or N (data objects), 'page.trend.test' will try to look up the tabled values (as in Siegel & Castellan (1988) for significance. For 'k,N > 3,20' or 'k,N > 4-10,12', a normal approximation is returned. Only one of these values will be returned.

If 'ranks' is FALSE, the function ranks the values in 'x' and then calculates the test. If the values are already ranks, it usually makes no difference.

## Value

| | |
|---|---|
| ranks | matrix of ranks |
| mean.ranks | mean ranks of data objects |
| L | value of the L statistic |
| p.table | whether the obtained L exceeded the table value for small k,N |
| Z | The normal approximation for larger k,N |
| pZ | the probability of the obtained normal value for larger k,N |

## Note

The Page test for ordered alternatives is slightly more powerful than the Friedman analysis of variance by ranks.

## Author(s)

Jim Lemon - thanks to Mikhail Trofimov and Michael Kirchhof for discovering major errors in the function and supplying the corrections

## References

Siegel, S. & Castellan, N.J.Jr. (1988) Nonparametric statistics for the behavioral sciences. Boston, MA: McGraw-Hill.

## Examples

```
# Craig's data from Siegel & Castellan, p 186
soa.mat<-matrix(c(.797,.873,.888,.923,.942,.956,
 .794,.772,.908,.982,.946,.913,
 .838,.801,.853,.951,.883,.837,
 .815,.801,.747,.859,.887,.902),nrow=4,byrow=TRUE)
page.trend.test(soa.mat)
```

---

`permute` *Permute a vector.*

---

## Description

Permute the values contained in a vector.

## Usage

```
permute(x)
```

## Arguments

| | |
|---|---|
| `x` | The vector of values that are to be permuted. |

## Details

'`permute`' calculates the number of permutations and creates a matrix with that number of rows. It fills the first column with the elements of '`x`' in groups large enough to cover the permutations of a vector with one less value. It then fills the remaining columns by calling itself with all values except the one in the first row of the current block. If '`x`' has only two values, it returns the trivial permutation of '`x`' and its reverse.

## Value

A matrix in which each row is a permutation of the values in '`x`'.

## Author(s)

Jim Lemon

## See Also

fillArows, fillArow

## Examples

```
permute(c(5,8,3,9))
```

## print.cats2ranks *Print the result of cats2ranks*

### Description

Print the result of cats2ranks.

### Usage

```
 ## S3 method for class 'cats2ranks'
print(x,...)
```

### Arguments

| | |
|---|---|
| x | The result of cats2ranks. |
| ... | a dummy argument to keep S3 methods happy |

### Details

Displays the names and mean ranks of the output of 'cats2ranks' in order of numerically ascending ranks.

### Value

nil

### Author(s)

Jim Lemon

### See Also

cats2ranks

## print.lwstat *Print the result of lwscreen*

### Description

Print the result of lwscreen.

### Usage

```
 ## S3 method for class 'lwstat'
print(x,...)
```

### Arguments

| | |
|---|---|
| x | The result of lwscreen. |
| ... | a dummy argument to keep S3 methods happy |

### Details

Displays the output of 'lwscreen'.

### Value

nil

### Author(s)

Jim Lemon

### See Also

lwscreen

---

## print.meanranks *Print the result of meanranks*

---

### Description

Print the result of meanranks.

### Usage

```
 ## S3 method for class 'meanranks'
print(x,...)
```

### Arguments

| | |
|---|---|
| x | The result of meanranks. |
| ... | a dummy argument to keep S3 methods happy |

### Details

Displays the names and mean ranks of the output of 'meanranks' in order of numerically ascending ranks.

### Value

nil

### Author(s)

Jim Lemon

### See Also

meanranks

---

`print.page.trend.test` *prints the L statistic for Page's trend test*

---

### Description

prints the obtained L statistic and the associated probability for the normal approximation if the sample size is sufficiently large

### Usage

```
 ## S3 method for class 'page.trend.test'
print(x,...)
```

### Arguments

| | |
|---|---|
| `x` | an object returned from '`page.trend.test`' |
| `...` | arguments to be passed to '`print`' |

### Value

nil

### Author(s)

Jim Lemon

---

`spsort` *Simple partial sorting of a vector of elements.*

---

### Description

Sort the elements in a vector according to a set of precedence rules.

### Usage

```
spsort(x,L=NULL)
```

## Arguments

| | |
|---|---|
| x | A matrix or data frame with at least two columns. The first two columns are interpreted as precedence pairs, meaning that the element in column 1 should appear before the one in column 2. |
| L | The vector of elements to be sorted. If NULL, it becomes all of the unique elements in 'x[1:2,]'. |

## Details

'spsort' steps through rows of 'x' identifying the positions of the leading and trailing elements in each rule. If the leading element in the rule does not precede the trailing element in L, its position in L is moved to just ahead of the trailing element. If all of the possible precedence rules for the vector L are specified, the sorting will be unique. In most cases, the order of the result will depend upon the initial order of L and the order of the precendence rules.

## Value

The vector 'L' sorted by the rules in 'x'.

## Author(s)

Jim Lemon

## Examples

```
# Pedro's example
Smaller<-c("ASD", "DFE", "ASD", "SDR", "EDF", "ASD")
Larger<-c("SDR", "EDF", "KLM", "KLM", "SDR", "EDF")
matComp<-cbind(Smaller,Larger)
spsort(matComp)
# scramble the order of rules
nmatrows<-nrow(matComp)
spsort(matComp[sample(1:nmatrows,nmatrows),])
# David Urbina's example
priors<-c("A","B","C","C","D","E","E","F","G")
posts<-c("E","H","A","D","E","B","F","G","H")
dinnerMat<-cbind(priors,posts)
spsort(dinnerMat)
# add the condition that the taquitos must precede the guacamole
dinnerMat<-rbind(dinnerMat,c("G","B"))
spsort(dinnerMat)
# scramble the rows
nmatrows<-nrow(dinnerMat)
spsort(dinnerMat[sample(1:nmatrows,nmatrows),])
```

# Index