Package 'hyper2'

November 12, 2025

Type Package

Title The Hyperdirichlet Distribution, Mark 2

Version 3.2

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Description A suite of routines for the hyperdirichlet distribution and reified Bradley-Terry; supersedes the 'hyperdirichlet' package; uses 'disordR' discipline <doi:10.48550/ARXIV.2210.03856>. To cite in publications please use Hankin 2017 <doi:10.32614/rj-2017-061>, and for Generalized Plackett-Luce likelihoods use Hankin 2024 <doi:10.18637/jss.v109.i08>.

License GPL (>= 2)

LazyData yes

Depends R (>= 4.1.0)

Suggests knitr, markdown, rmarkdown, testthat, bookdown, rticles, covr

VignetteBuilder knitr

Imports Rcpp (>= 1.0-7), partitions, disordR (>= 0.0-9), alabama, calibrator, Rdpack, magrittr, cubature, methods

LinkingTo Rcpp

URL https://github.com/RobinHankin/hyper2, https://robinhankin.github.io/hyper2/

BugReports https://github.com/RobinHankin/hyper2/issues

RoxygenNote 7.1.1

RdMacros Rdpack

NeedsCompilation yes

Author Robin K. S. Hankin [aut, cre] (ORCID: <https://orcid.org/0000-0001-5982-0415>)

Repository CRAN

Date/Publication 2025-11-12 15:40:02 UTC

Contents

`hyper2-package` *The Hyperdirichlet Distribution, Mark 2*

Description

A suite of routines for the hyperdirichlet distribution and reified Bradley-Terry; supersedes the 'hyperdirichlet' package; uses 'disordR' discipline <doi:10.48550/ARXIV.2210.03856>. To cite in publications please use Hankin 2017 <doi:10.32614/rj-2017-061>, and for Generalized Plackett-Luce likelihoods use Hankin 2024 <doi:10.18637/jss.v109.i08>.

Details

The DESCRIPTION file:

Package:	hyper2
Type:	Package
Title:	The Hyperdirichlet Distribution, Mark 2
Version:	3.2
Authors@R:	person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@gmail.co
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
Description:	A suite of routines for the hyperdirichlet distribution and reified Bradley-Terry; supersedes the 'hyperdirich
License:	GPL (>= 2)
LazyData:	yes
Depends:	R (>= 4.1.0)
Suggests:	knitr, markdown, rmarkdown, testthat, bookdown, rticles, covr
VignetteBuilder:	knitr
Imports:	Rcpp (>= 1.0-7), partitions, disordR (>= 0.0-9), alabama, calibrator, Rdpack, magrittr, cubature, methods
LinkingTo:	Rcpp
URL:	https://github.com/RobinHankin/hyper2, https://robinhankin.github.io/hyper2/
BugReports:	https://github.com/RobinHankin/hyper2/issues
RoxygenNote:	7.1.1
RdMacros:	Rdpack
Author:	Robin K. S. Hankin [aut, cre] (ORCID: <https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

cplusplus	Wrappers to c calls
curling	Curling at the Winter Olympics, 1998-2018
dirichlet	Dirichlet distribution and generalizations
equalp.test	Hypothesis testing
eurodance	Eurovision Dance contest dataset
eurovision	Eurovision Song contest dataset
fillup	Fillup function
formula1	Formula 1 dataset
ggol	Order statistics
gradient	Differential calculus
handover	Dataset on communication breakdown in handover between physicians
head.hyper2	First few terms of a distribution: DEPRECATED
hepatitis	Hepatitis dataset discussed by Agresti
hyper2	Basic functions in the hyper2 package
hyper2-package	The Hyperdirichlet Distribution, Mark 2
hyper3	Weighted probability vectors: 'hyper3' objects
icons	Dataset on climate change due to O'Neill
increment	Increment and decrement operators
interzonal	1963 World Chess Championships
javelin	Javelin dataset
jester	Jester dataset
karate	Karate dataset
karpov_kasparov_anand	Karpov, Kasparov, Anand
keep	Keep or discard players
length.hyper2	Length method for hyper2 objects
loglik	Log likelihood functions
masterchef	Masterchef series 6
matrix2supp	Convert a matrix to a likelihood function
maxp	Maximum likelihood estimation
moto	MotoGP dataset
mult_grid	Kronecker matrix product functionality
ordertable	Order tables
ordertable2points	Calculate points from an order table
ordertable2supp	Translate order tables to support functions
ordertrans	Order transformation
ordervec2supp3	Various functionality for races and hyper3 likelihood functions
pairwise	Pairwise comparisons
pentathlon	Pentathlon
powerboat	Powerboat dataset
profile	Profile likelihood and support
psubs	Substitute players of a hyper2 object
pwa	Player with advantage
ranktable	Convert rank tables to and from order tables
rhyper2	Random 'hyper2' objects
rhyper3	Random hyper3 objects
rowing	Rowing dataset, sculling

rp	Random samples from the prior of a 'hyper2' object
rrace	A random race with given BT strengths
rrank	Random ranks
skating	Figure skating at the 2002 Winter Olympics
soling	Sailing at the 2000 Summer Olympics - soling
summary.hyper2	Summary method for hyper2 objects
suplist	Methods for suplist objects
suppfun	Convert various datasets to support functions.
surfing	Surfing dataset
sushi	Sushi dataset
table_tennis	Match outcomes from repeated table tennis matches
tennis	Match outcomes from repeated doubles tennis matches
tidy	Tidy up a hyper2 object
universities	New Zealand University ranking data
volleyball	Results from the NOCS volleyball league
volvo	Race results from the 2014-2015 Volvo Ocean Race
zapweak	Zap weak competitors
zipf	Zipf's law

A generalization of the Dirichlet distribution, using a more computationally efficient method than the **hyperdirichlet** package. The software is designed for the analysis of order statistics and team games.

The hex sticker features the "draw monster", a concept introduced in Hankin (2020) as a reified entity, representing the winner of a drawn chess game. The character was visualized and sketched by my daughter Annabel. In the chess world, draws are often seen as dull and undesirable, and the draw monster embodies this sentiment: he is a dampening presence, a metaphorical wet blanket. The drawing conveys this lack of fighting spirit well, capturing the essence of an uninspired and anticlimactic outcome.

Author(s)

Robin K. S. Hankin [aut, cre] (ORCID: <https://orcid.org/0000-0001-5982-0415>)

Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>

References

- R. K. S. Hankin (2010). "A Generalization of the Dirichlet Distribution", *Journal of Statistical Software*, 33(11), 1-18, doi:10.18637/jss.v033.i11
- R. K. S. Hankin 2017. "Partial Rank Data with the hyper2 Package: Likelihood Functions for Generalized Bradley-Terry Models". *The* R *Journal* 9:2, pages 429-439, doi:10.32614/rj2017061
- R. K. S. Hankin 2024. "Generalized Plackett-Luce Likelihoods", *Journal of Statistical Software*, 109(8), 1-17, doi:10.18637/jss.v109.i08

Examples

```
icons
maxp(icons)
```

`as.ordertable` *Convert an order table with DNS entries to a nice order table*

Description

Given an ordertable such as `F1_table_2017` which is a "wikitable" object, function `as.ordertable()` returns a nicified version in which entries such as DNS are replaced with zeros. Finishing competitors are assigned numbers $1 - n$ with no gaps; the function can be used to extract a subset of competitors.

Function `ordertable2supp()` offers similar functionality but returns a `hyper2` object directly.

Usage

```
as.ordertable(w)
```

Arguments

`w`	A generalized ordertable, a wikitable

Details

Operates columnwise, and treats any entry not coercible to numeric as DNF.

Value

Returns an ordertable suitable for coercion to a `hyper2` object.

Author(s)

Robin K. S. Hankin

See Also

`ordertable`,`ordertable2supp`

Examples

```
as.ordertable(F1_table_2017)
ordertable2supp(as.ordertable(F1_table_2017[1:9,]))
```

`attemptstable2supp3` *Translate attempt tables to hyper3 support functions*

Description

description here

Usage

```
attemptstable2supp3(a, decreasing, give.supp=TRUE, dnf.last=TRUE)
```

Arguments

`a`	Data frame, see details
`decreasing`	Boolean, with `TRUE` meaning that the highest score wins [e.g. javelin distances] and `FALSE` meaning that the lowest score wins [e.g. times for a race]
`give.supp`	Boolean, return the support function or the order statistic
`dnf.last`	Boolean, should NA entries count as coming last (TRUE) or be ignored (FALSE)

Details

Function `attemptstable2supp3()` is intended for use on attempts tables like `javelin`.

These objects can be generated by running script `inst/javelin.Rmd`, which includes some further discussion and technical documentation, and creates file `javelin.rda` which resides in the `data/` directory.

Value

Returns a `hyper3` object

Author(s)

Robin K. S. Hankin

See Also

`ordertable2supp`,`javelin`

Examples

```
jj <- javelin_table[1:3,]
jj
attemptstable2supp3(jj)
```

B *Normalizing constant for the hyperdirichlet distribution*

Description

Numerical techniques for calculating the normalizing constant for the hyperdirichlet distribution

Usage

```
B(H, disallowed=NULL, give=FALSE, ...)
probability(H, disallowed=NULL, ...)
mgf(H, powers, ...)
dhyper2(ip,H,...)
dhyper2_e(e,H,include.Jacobian=TRUE)
mean_hyper2(H, normalize=TRUE, ...)
Jacobian(e)
e_to_p(e)
p_to_e(p)
```

Arguments

H	Object of class hyper2
powers	Vector of length dim(x) whose elements are the powers of the expectation; see details section
disallowed	Function specifying a subset of the simplex over which to integrate; default NULL means to integrate over the whole simplex. The integration proceeds over p with disallowed(p) evaluating to FALSE
e, p	A vector; see details
ip	A vector of probabilities corresponding to indep(p) where p is vector with unit sum
include.Jacobian	Boolean, with default TRUE meaning to include the Jacobian transformation in the evaluation, and FALSE meaning to ignore it; use FALSE for likelihood work and TRUE for probability densities
give	Boolean, with default FALSE meaning to return the value of the integral and TRUE meaning to return the full output of adaptIntegrate()
normalize	Boolean, indicates whether return value of mean_hyper2() is normalized to have unit sum
...	Further arguments passed to adaptIntegrate()

Details

- Function B() returns the normalizing constant of a hyperdirichlet likelihood function. Internally, p is converted to e (by e_to_p()) and the integral proceeds over a hypercube. This function can be very slow, especially if disallowed is used.

- Function `dhyper2(ip,H)` is a probability density function on the independent components of a unit-sum vector, that is, `ip=indep(p)`. This function calls `B()` each time so might be a performance bottleneck.
- Function `probability()` gives the probability of an observation from a hyperdirichlet distribution satisfying `!disallowed(p)`.
- Function `mgf()` is the moment generating function, taking an argument that specifies the powers of `p` needed: the expectation of $\prod_{i=1}^{n} p_i^{\text{powers}[i]}$ is returned.
- Function `mean_hyper2()` returns the mean value of the hyperdirichlet distribution. This is computationally slow (consider `maxp()` for a measure of central tendency). The function takes a `normalize` argument, not passed to `adaptIntegrate()`: this is Boolean with `FALSE` meaning to return the value found by integration directly, and default `TRUE` meaning to normalize so the sum is exactly 1

Value

- Function `B()` returns a scalar: the normalization constant
- Function `dhyper2()` is a probability density function over `indep(p)`
- Function `mean()` returns a k-tuple with unit sum
- Function `mgf()` returns a scalar equal to the expectation of `p^power`
- Functions `is.proper()` and `validated()` return a Boolean
- Function `probability()` returns a scalar, a (Bayesian) probability

Note

The adapt package is no longer available on CRAN; from 1.4-3, the package uses `adaptIntegrate` of the cubature package.

Author(s)

Robin K. S. Hankin

See Also

loglik

Examples

```
# Two different measures of central tendency:
# mean_hyper2(chess,tol=0.1)   # takes ~10s to run
maxp(chess)                    # faster

# Using the 'disallowed' argument typically results in slow run times;
# use high tol for speed:

# probability(chess,disallowed=function(p){p[1]>p[2]},tol=0.5)
# probability(chess,disallowed=function(p){p[1]<p[2]},tol=0.5)

# Above should sum to 1 [they are exclusive and exhaustive events]
```

balance *Enforce the zero power sum condition*

Description

Sometimes a hyper2 object is unbalanced in the sense that its powers do not sum to zero. This is rectified by balance(), which modifies the power of the bracket corresponding to the sum of all pnames accordingly.

Usage

```
balance(H)
```

Arguments

H	object of class hyper2 or hyper3

Details

This is just a convenience function, all it does is

```
H[pnames(H)] <- 0
H[pnames(H)] <- -sum(pnames(H))
H
```

(the first line ensures that H[pnames(H)] is over-written correctly by the second). Package vignette zeropower discusses the zero power sum condition.

Value

Returns a balanced hyper2 object

Author(s)

Robin K. S. Hankin

See Also

print.hyper2

Examples

```
H <- hyper2()
H["a"] <- 6
H["b"] <- 3
H[c("a","c")] <- 7
H <- balance(H)
maxp(H)
```

`baseball` *Baseball results, following Agresti*

Description

Results from repeated games among seven baseball teams, following Agresti

Usage

```
data(baseball)
```

Format

A `hyper2` object that gives a likelihood function

Details

Agresti discusses results from seven baseball teams in the 1987 season of the Eastern Division of the American League.

A results table and likelihood function is given in the package as `baseball_table` and `baseball` respectively. The maximum likelihood estimate is given as `baseball_maxp`, but can be reproduced by `maxp(baseball)`.

These objects can be generated by running script `inst/home_advantage.Rmd`, which includes some further discussion and technical documentation, and creates file `baseball.rda` which resides in the `data/` directory.

References

A. Agresti 2002. "Categorical data analysis". John Wiley and Sons; p437

See Also

`hyper3`

Examples

```
baseball_table
baseball_table[1:3,1:3]
home_away3(baseball_table[1:3,1:3],1.3)
```

carcinoma *Carcinoma dataset discussed by Agresti*

Description

A dataset considered by Agresti. Seven clinicians are asked whether they see evidence for carcinoma on different patients.

Usage

```
data(carcinoma)
```

Format

A hyper2 object that gives a likelihood function

Details

Object carcinoma_table is drawn from Agresti. The first seven columns correspond to the seven clinicians A-G, the next is the count of observations, and the remaining columns are fitted values according to different models discussed by Agresti.

Object carcinoma is a likelihood function (of class lsl) on the Bradley-Terry strengths of the seven clinicians. The clinicians diagnosed the presence or absence of carcinoma on a total of 118 patients in a blind rating scheme. The maximum likelihood estimator for the clinicians' Bradley-Terry strengths is given as carcinoma_maxp, which is computationally expensive to find. The package also includes carcinoma_count, which is a different estimator for the Clinicians' BT strengths.

These objects can be generated by running script inst/carcinoma.Rmd, which includes some further discussion and technical documentation, and creates file carcinoma.rda which resides in the data/ directory.

References

A. Agresti, 2002. "Categorical data analysis". John Wiley and Sons. Table 13.1, p542.

See Also

race3,hepatitis

Examples

```
pie(carcinoma_maxp)
```

`character_to_number` *Convert a character vector to a numeric vector*

Description

Convert string descriptions of competitors into their number

Usage

```
character_to_number(char, pnames)
char2num(char, pnames)
```

Arguments

`char`	Character vector to be converted
`pnames`	Names vector (usually `pnames(H)`)

Details

In earlier versions of this package, the internal mechanism of functions such as `ggrl()`, and all the `C++` code, operated with the competitors labelled with a non-negative integer; it is then natural to refer to the competitors as `p1`, `p2`, etc.

However, sometimes the competitors have names (as in, for example, the `rowing` dataset). If so, it is more natural to refer to the competitors using their names rather than an arbitrary integer.

Function `character_to_number()` converts the names to numbers. If an element of `char` is not present in `pnames`, an error is returned (function `char2num()` is an easy-to-type synonym). The function is here because it is used in `ggrl()`.

Author(s)

Robin K. S. Hankin

See Also

`suppfun`

Examples

```
x <- sample(9)
names(x) <- sample(letters[1:9])
H <- ordervec2supp(x)
character_to_number(letters[1:3],pnames(H))

char2num(c("PB","L"),pnames(icons))
```

chess *Chess playing dataset*

Description

A tally of wins and losses for games between three chess players: Topalov, Anand, Karpov.

Usage

```
data(chess)
```

Details

(there are three chess datasets in the package, documented at `interzonal.Rd` [the 1963 World championship], `kka.Rd` [Karpov-Kasparov-Anand dataset], and `chess.Rd` [rock-paper-scissors using Topalov-Anand-Karpov])

This is a very simple dataset that can be used for illustration of `hyper2` idiom.

The players are:

- Grandmaster Veselin Topalov. FIDE world champion 2005-2006; peak rating 2813
- Grandmaster Viswanathan Anand. FIDE world champion 2000-2002, 2008; peak rating 2799
- Grandmaster Anatoly Karpov. FIDE world champion 1993-1999; peak rating 2780

Observe that Topalov beats Anand, Anand beats Karpov, and Karpov beats Topalov (where "beats" means "wins more games than").

The games thus resemble a noisy version of "rock paper scissors".

The likelihood function does not record who played white; see `karpov_kasparov_anand` for such a dataset.

These objects can be generated by running script `inst/rock_paper_scissors.Rmd`, which includes some further discussion and technical documentation and creates file `chess.rda` which resides in the `data/` directory.

File `inst/ternaryplot_hyper2.Rmd` gives an example showing the `chess` likelihood function that uses `Ternary::ternaryPlot()`.

References

- https://en.chessbase.com/

See Also

`karpov_kasparov_anand`

Examples

```
data(chess)
maxp(chess)

 mgf(chess,c(Anand=2),tol = 0.1)  # tolerance for speed
```

---

consistency *Consistency check for hyper2 objects*

Description

Given a hyper2 object, calculate the maximum likelihood point in two ways and plot one against the other to check for consistency.

Usage

```
consistency(H, plot=TRUE, ...)
```

Arguments

H	A hyper2 object
plot	If TRUE (default), plot a comparison and return a matrix invisibly, and if FALSE return the matrix. Modelled on argument plot of hist
...	Further arguments, passed to points()

Details

Given a hyper2 object, calculate the maximum likelihood estimate of the players' strengths using maxp(); then reverse the pnames attribute and calculate the players' strengths again. These two estimates should be identical but small differences highlight numerical problems. Typically, the differences are small if there are fewer than about 25 players.

Reversing the pnames() is cosmetic in theory but is a non-trivial operation: for example, it changes the identity of the fillup from the last player to the first.

Value

Returns a named three-row matrix with first row being the direct evaluate, second row being the reverse of the reversed evaluate, and the third being the difference

Author(s)

Robin K. S. Hankin

See Also

ordertrans

Examples

```
# consistency(icons)

x <- icons
y <- icons
pnames(y) <- rev(pnames(y))
gradient(x,indep(equalp(x)))
gradient(y,indep(equalp(y)))
```

`constructor` *Formula 1 dataset: the constructors' championship*

Description

Race results from 2017 Formula One constructors' Championship

Usage

```
data(constructor)
```

Format

A `hyper3` object that gives a likelihood function

Details

The Constructors championship runs parallel to the Formula 1 drivers' championship. The package currently includes data from 2020 and 2021; the following text applies to both years. I will add more years eventually.

Object `constructor_2021_table` is a dataframe, taken from Wikipedia, with rows corresponding to performance of a constructor. Each constructor fields two cars in each race; the identity of the driver is not important in this context (and indeed may change as the season progresses). The first column is the name of the constructor, the next 22 columns show the ranks of the constructors' cars, and the final one is the points awarded. At each venue, the constructor's best performance is listed first. The constructors' names change quite frequently (e.g. "Red Bull Racing-TAG Heuer" raced 2016,2017, and 2018; "Red Bull Racing-Honda" raced 2019, 2020, and 2021. I am not sure whether to treat these as separate entities or not; file `inst/constructor_names.txt` gives a dataframe of team names and years they competed (not currently part of the package). The row names of the dataframe cannot be the constructors because these are not unique.

Object `constructor_2021_maxp` gives the maximum likelihood estimate for the constructors' strengths. The corresponding `hyper3` likelihood function `constructor_2021` is produced by `ordertable2supp3()`.

These objects can be generated by running script `inst/race3.Rmd`, which includes some further discussion and technical documentation, and creates file `constructor.rda` which resides in the `data/` directory.

References

Wikipedia contributors. (2022, April 14). 2021 Formula One World Championship. In _Wikipedia, The Free Encyclopedia_. Retrieved 05:16, April 17, 2022, from https://en.wikipedia.org/w/index.php?title=2021_Formula_One_World_Championship&oldid=1082745216

See Also

formula1

Examples

```
dotchart(constructor_2021_maxp)
```

---

counterstrike *Counterstrike*

Description

A kill-by-kill analysis of a counterstrike game.

Usage

```
data(counterstrike)
```

Details

E-sports are a form of competition using video games. E-sports are becoming increasingly popular, with high-profile tournaments attracting over 400 million viewers, and prize pools exceeding US$20m.

Counter Strike: Global Offensive (CS-GO) is a multiplayer first-person shooter game in which two teams of five compete in an immersive virtual reality combat environment. CS-GO is distinguished by the ability to download detailed gamefiles in which every aspect of an entire match is recorded, it being possible to replay the match at will.

Statistical analysis of such gamefiles is extremely difficult, primarily due to complex gameplay features such as cooperative teamwork, within-team communication, and real-time strategic fluidity.

It is the task of the statistician to make robust inferences from such complex datasets, and here I discuss data from an influential match between "FaZe Clan" and "Cloud9", two of the most successful E-sports syndicates of all time, when they competed at Boston 2018.

Dataset `counterstrike` is a loglikelihood function for the strengths of ten counterstrike players; `counterstrike_maxp` is a precomputed evaluate, and `zacslist` the observations used to calculate the loglikelihood function.

The probability model is similar to that of `NBA`: when a player kills (scores), this is taken to be a success of the whole team rather than the shooter.

File `inst/counterstrike.R` and `inst/counterstrike_random.R` include some further randomisation tests and discussion.

The objects documented here can be generated by running script `inst/counterstrike.Rmd`, which includes some further discussion and technical documentation and creates file `counterstrike.rda` which resides in the `data/` directory.

Counterstrike dataset kindly supplied by Zachary Hankin.

References

- https://www.youtube.com/watch?v=XKWzlG4jDnI
- https://en.wikipedia.org/wiki/FaZe_Clan
- https://en.wikipedia.org/wiki/Cloud9

Examples

```
dotchart(counterstrike_maxp)
```

---

cplusplus *Wrappers to c calls*

Description

Various low-level wrappers to C functions, courtesy of Rcpp

Usage

```
overwrite(L1, powers1, L2, powers2)
accessor(L,powers,Lwanted)
assigner(L,p,L2,value)
addL(L1,p1,L2,p2)
identityL(L,p)
evaluate(L, powers, probs, pnames)
differentiate(L, powers, probs, pnames, n)
differentiate_n(L, powers, probs, pnames, n)
```

Arguments

L, L1, L2, Lwanted	Lists with character vector elements, used to specify the brackets of the hyperdirichlet distribution
p, p1, p2, powers, powers1, powers2	A numeric vector specifying the powers to which the brackets are raised
value	RHS in assignment, a numeric vector
probs	Vector of probabilities for evaluation of log-likelihood
pnames	Character vector of names
n	Integer specifying component to differentiate with respect to

Details

These functions are not really intended for the end-user, as out-of-scope calls may cause crashes.

Value

These functions return a named List

Author(s)

Robin K. S. Hankin

---

curling *Curling at the Winter Olympics, 1998-2018*

Description

Data for women's Olympic Curling at the 2002 Winter Olympics.

Usage

```
data(curling)
```

Details

There are five datasets loaded by `data("curling")`:

- `curling_table`, an order table for Winter Olympics years 1998,2002,2006,2010,2014, and 2018 for 13 countries.
- `curling1`, a log likelihood function on the assumption that not attending (indicated by `NA`) is equivalent to a `DNS` in Formula 1
- `curling2`, a log likelihood function on the assumption that not attending is noninformative
- `curling1_maxp` and `curling2_maxp`, corresponding evaluates

These objects can be generated by running script `inst/curling.Rmd`, which includes some further discussion and technical documentation and creates file `curling.rda` which resides in the `data/` directory.

Author(s)

Robin K. S. Hankin

References

- Wikipedia contributors. Curling at the Winter Olympics [Internet]. Wikipedia, The Free Encyclopedia; 2021 Jan 7, 14:23 UTC [cited 2021 Jan 21]. Available from: https://en.wikipedia.org/w/index.php?title=Curling_at_the_Winter_Olympics&oldid=998891075

Examples

```
data(curling)
dotchart(curling1_maxp)
```

`dirichlet` *Dirichlet distribution and generalizations*

Description

The Dirichlet distribution in likelihood (for p) form, including the generalized Dirichlet distribution due to Connor and Mosimann

Usage

```
dirichlet(powers, alpha)
dirichlet3(powers, lambda=NULL)
GD(alpha, beta, beta0=0)
GD_wong(alpha, beta)
rdirichlet(n,H)
is.dirichlet(H)
rp_unif(n,H)
```

Arguments

`powers`	In function `dirichlet()` a (named) vector of powers
`alpha, beta`	A vector of parameters for the Dirichlet or generalized Dirichlet distribution
`beta0`	In function `GD()`, an arbitrary parameter
`H`	Object of class `hyper2`
`lambda`	Vector of weights in `dirichlet3()`
`n`	Number of observations

Details

These functions are really convenience functions.

Function `rdirichlet()` returns random samples drawn from a Dirichlet distribution using the gamma distribution. If second argument `H` is a `hyper2` object, it is tested [with `is.dirichlet()`] for being a Dirichlet distribution. If so, samples from it are returned. If not, (e.g. `icons`), an error is given. If `H` is not a `hyper2` object, it is interpreted as a (possibly named) vector of parameters α [**not** a vector of powers].

Function `rp_unif()` returns uniformly distributed vectors, effectively using `H*0`; but note that this uses Dirichlet sampling which is much faster and better than the Metropolis-Hastings functionality documented at `rp.Rd`.

Functions `GD()` and `GD_wong()` return a likelihood function corresponding to the Generalized Dirichlet distribution as presented by Connor and Mosimann, and Wong, respectively. In `GD_wong()`,

alpha and beta must be named vectors; the names of alpha give the names of $x_1, \ldots, x_k$ and the last element of beta gives the name of x_{k+1}.

Function dirichlet3() returns a hyper3 object with weights lambda. If lambda is length less than that of powers, it is padded with 1s [so default NULL corresponds to unit weights, that is, a hyper2 object]. A use-case is given in inst/rock_paper_scissors_monster.Rmd.

Note

A dirichlet distribution can have a term with zero power. But this poses problems for hyper2 objects as zero power brackets are dropped.

Function dirichlet3() is a replacement for now removed function pair3().

Function rdirichlet() commits a very mild (but necessary in the absence of a working dismat package) violation of disordR discipline, as the columns of the returned matrix have the same order as pnames(H)

Author(s)

Robin K. S. Hankin

References

- R. J. Connor and J. E. Mosimann 1969. "Concepts of independence for proportions with a generalization of the Dirichlet distribution". *Journal of the American Statistical Association*, 64:194–206
- T.-T. Wong 1998. "Generalized Dirichlet distribution in Bayesian Analysis". *Applied Mathematics and Computation*, 97:165–181

See Also

hyper2,rp

Examples

```
x1 <- dirichlet(c(a=1,b=2,c=3))
x2 <- dirichlet(c(c=3,d=4))

x1+x2

H <- dirichlet(c(a=1,b=2,c=3,d=4))
rdirichlet(10,H)
colMeans(rdirichlet(1e4,H))

dirichlet3(c(fish=3,chips=2),lambda=1.8)
dirichlet3(c(x=6,y=5,z=2),1:3)
```

eurodance *Eurovision Dance contest dataset*

Description

Voting patterns from Eurovision Dance Contest 2008

Usage

```
data(eurovision)
```

Format

A hyper2 object that gives a likelihood function.

Details

Object eurodance is a hyper2 object that gives a likelihood function for the skills of the 14 competitor countries in 2008 Eurovision Dance contest. Object eurodance_table gives the original dataset and eurodance_maxp the evaluate of the competitors' Plackett-Luce strengths.

The dataset is interesting because, in addition to the regular votes by each nation, there is an Expert jury vote as well. We may use Plackett-Luce likelihoods to compare the performance of the Expert jury with the national votes.

These objects can be generated by running script inst/eurodance.Rmd, which includes some further discussion and technical documentation and creates file eurodance.rda which resides in the data/ directory.

References

- Wikipedia contributors, "Eurovision Song Contest 2009—Wikipedia, The Free Encyclopedia", 2018, https://en.wikipedia.org/w/index.php?title=Eurovision_Song_Contest_2009&oldid=838723921 [Online; accessed 13-May-2018].
- P. M. E. Altham, personal communication

See Also

eurodance

Examples

```
data(eurodance)
dotchart(eurodance_maxp)
```

`eurovision` *Eurovision Song contest dataset*

Description

Voting patterns from Eurovision 2009

Usage

```
data(eurovision)
```

Format

A `hyper2` object that gives a likelihood function.

Details

Object `eurovision` is a `hyper2` object that gives a likelihood function for the skills of the 18 competitor countries in semi-final 1 of the 2009 Eurovision Song contest. Object `eurovision_table` gives the original dataset and `eurovision_maxp` the evaluate of the competitors' Plackett-Luce strengths.

The motivation for choosing this particular dataset is that Pat Altham (Statistical Laboratory, Cambridge) considered it with a view to discover similarities between voters. In the current analysis, the likelihood function `eurovision` assumes their independence.

These objects can be generated by running script `inst/eurovision.Rmd`, which includes some further discussion and technical documentation and creates file `eurovision.rda` which resides in the `data/` directory.

References

- Wikipedia contributors, "Eurovision Song Contest 2009—Wikipedia, The Free Encyclopedia", 2018, https://en.wikipedia.org/w/index.php?title=Eurovision_Song_Contest_2009&oldid=838723921 [Online; accessed 13-May-2018].
- P. M. E. Altham, personal communication

See Also

`eurodance`

Examples

```
data(eurovision)
dotchart(eurovision_maxp)
```

Extract *Extract or replace parts of a hyper2 object*

Description

Extract or replace parts of a hyper2 object

Usage

```
## S3 method for class 'hyper2'
x[...]
## S3 replacement method for class 'hyper2'
x[index, ...] <- value
assign_lowlevel(x,index,value)
overwrite_lowlevel(x,value)
```

Arguments

x	An object of class hyper2
...	Further arguments, currently ignored
index	A list with integer vector elements corresponding to the brackets whose power is to be replaced
value	Numeric vector of powers

Details

These methods should work as expected, although the off-by-one issue might be a gotcha.

For the extract method, H[L], a hyper2 object is returned. The replace method, H[L] <- value, the index specifies the brackets whose powers are to be overwritten; standard disordR protocol is used.

If the index argument is missing, viz H1[] <- H2, this is a special case. Argument H1 must be a hyper2 object, and the idiom effectively executes H1[brackets(H2)] <- powers(H2), but more efficiently (note that this operation is well-defined even though the order of the brackets is arbitrary). This special case is included in the package because it has a very natural C++ expression [function overwrite() in the src/ directory] that was too neat to omit.

Altering (incrementing or decrementing) the power of a single bracket is possible using idiom like H[x] <- H[x] + 1; this is documented at Ops.hyper2, specifically hyper2_sum_numeric() and a discussion is given at increment.Rd.

Functions assign_lowlevel() and overwrite_lowlevel() are low-level helper functions and not really intended for the end-user.

Value

The extractor method returns a hyper2 object, restricted to the elements specified

Note

Use `powers()` and `brackets()` to extract a numeric vector of powers or a list of integer vectors respectively.

Replacement idiom `H[x] <- val` cannot use non-trivial recycling. This is because the elements of `H` are stored in an arbitrary order, but the elements of `val` are stored in a particular order. Also see function `hyper2_sum_numeric()`.

Author(s)

Robin K. S. Hankin

See Also

`hyper2`,`Ops.hyper2`

Examples

```
data(chess)

chess["Topalov"]
chess[c("Topalov","Anand")]
chess[c("Anand","Topalov")]

# Topalov plays Anand and wins:

chess["Topalov"] <- chess["Topalov"]+1
chess[c("Topalov","Anand")] <- chess[c("Topalov","Anand")]-1


# Topalov plays *Kasparov* and wins:
chess["Topalov"] <- chess["Topalov"] + 1
chess[c("Topalov","Kasparov")] <- chess[c("Topalov","Kasparov")] -1

# magrittr idiom:
# chess["Topalov"] %<>% inc
# chess[c("Topalov","Kasparov")] %<>% dec

# overwriting idiom:
H <- hyper2(list("Topalov","X"),6)
chess[] <- H

H <- icons
```

`fillup` *Fillup function*

Description

Function `fillup()` concatenates a vector with a 'fillup' value to ensure a unit sum; if given a matrix, attaches a column so the rowsums are 1.

Function `indep()` is the inverse: it removes the final element of a vector, leaving only an independent set.

Usage

```
fillup(x,H=NULL,total=1)
indep(x)
```

Arguments

`x`	Numeric vector
`H`	Object with `pnames()` attribute, typically of class `hyper2` or `hyper3`, used for names if supplied
`total`	Total value for probability

Details

Usually you want the total to be one, to enforce the unit sum constraint. Passing `total=0` constrains the sum to be zero. This is useful when considering δp; see the example at `gradient.Rd`.

Author(s)

Robin K. S. Hankin

See Also

`equalp`,`gradient`

Examples

```
fillup(c(1/2,1/3))

indep(c(1/2,1/3,1/6))

fillup(indep(icons_maxp))
fillup(indep(icons_maxp),icons)
```

formula1 *Formula 1 dataset*

Description

Race results from 2017 Formula One World Championship

Usage

```
data(formula1)
formula1_points_systems(top=11)
```

Arguments

top	Number of drivers to retain in formula1_points_systems()

Format

A hyper2 object that gives a likelihood function

Details

Object formula1 is a hyper2 object that gives a likelihood function for the strengths of the competitors of the 2017 Formula One (Drivers') World Championship. Object F1_table_2017 is an order table: a data frame with rows being drivers, columns being venues, and entries being places. Thus looking at the first row, first column we see that Hamilton placed second in Austria.

The package uses files like inst/formula1_2017.txt as primary sources. These are generally copied from wikipedia, converted into tab-separated clean seven bit ascii, and tidied up a little. I have removed diacritics from names, so we see "Raikkonen", "Perez", etc. Also where distinct drivers with the same surname compete, I have indicated this, e.g. schumacher_R is Ralf Schumacher, schumacher_M is Michael, and schumacher_Mick is Mick; the underscore device means that quoting should not be needed in R idiom. I have not been entirely consistent here, with Bruno Senna appearing as "Senna_B" and Nelson Piquet Junior appearing as "PiquetJ" [on the grounds that in these cases the fathers, being more eminent, should be the primary eponym] although this might change in the future.

Object F1_table_2017 is simply the first 20 columns of read.table(inst/formula1_2017.txt) and object F1_points_2017 is column 21. The final column of all the text files is the points and it is easy to mistake this for a venue with results (doing so will give a "Error in ordervec2supp(o) : nonzero elements of d should be 1,2,3,4,...,n" error).

The likelihood function formula1 is ordertable2supp(F1_table_2017) [NB: suppfun(F1_table_2017) fails: suppfun() will not try to guess whether its argument is a ranktable or an ordertable]. The datasets in the package are derived from text files in the inst/ directory (e.g. formula1_2017.txt) by script file inst/f1points_Omaker.R. Executing this script creates files like formula1_results_2017.rda.

The text files can be converted directly into ranktable objects and support functions as follows:

```
a <- read.table("formula1_2022.txt",header=TRUE)
a <- a[,seq_len(ncol(a)-1)]  # strips out the points column
wikitable_to_ranktable(a)
ordertable2supp(a)      # works fine
suppfun(ordertable(a)) # Same as previous line, but suppfun() needs to know what its argument is
```

To convert to a numeric matrix with DNS etc converted to NA:

```
a <- read.table("formula1_2024.txt",header=TRUE)
a <- a[, -ncol(a)]
a <- as.matrix(a)
storage.mode(a) <- "numeric"
```

[this is used in file inst/test_formula1.R, which provides a consistency check for files inst/formula1_???.txt].

Function formula1_points_system() gives various possible points systems for the winner, second, third, etc, placing drivers.

The constructors' championship is discussed at constructor.Rd.

There is a large amount of documentation in the inst/ directory in the form of Rmd files.

References

"Wikipedia contributors", *2017 Formula One World Championship—Wikipedia, The Free Encyclopedia*, 2018. https://en.wikipedia.org/w/index.php?title=2017_Formula_One_World_Championship&oldid=839923210 [Online; accessed 14-May-2018]

See Also

ordertable2supp,constructor

Examples

```
summary(formula1)
## Not run: #Takes too long
dotchart(maxp(formula1))

## End(Not run)
```

---

ggol *Order statistics*

Description

Various functions for calculating the likelihood function for order statistics

Usage

```
ggrl(H, ...)
general_grouped_rank_likelihood(H, ...)
goodbad(winners,losers)
elimination(all_players)
rankvec_likelihood(v,nonfinishers)
race(v,nonfinishers)
```

Arguments

H	Object of class hyper2
...	Numeric or character vectors specifying groups of players with equal rank, with higher-ranking groups coming earlier in the argument list
all_players, winners, losers	Numeric or character vectors specifying competitors. See details
v	A character vector specifying ranks. Thus c("b","c","a") means that b came first, c second, and a third
nonfinishers	A character vector with entries corresponding to competitors who did not finish. Thus race(c("a","b"),c("p","q")) means that the field is a,b,c,d; a came first, b came second and c and d did not finish

Details

These functions are designed to return likelihood functions, in the form of lists of hyper2() objects, for typical order statistics such as the results of rowing heats or MasterChef ments.

Direct use of rankvec_likelihood() is discouraged: use suppfun() instead, for example suppfun(letters).

Function ggrl() is an easily-typed alias for general_grouped_rank_likelihood().

Function goodbad() is a convenience function for ggrl() in which a bunch of contestants is judged. It returns a likelihood function for the observation that the members of one subset were better than those of another. Thus goodbad(letters[1:3],letters[4:5]) corresponds to the observation that d and e were put into an elimination trial (and abc were not).

Function elimination() gives a likelihood function for situations where the *weakest* player is identified at each stage and subsequently eliminated from the competition. It is intended for situations like the Great British Bake-off and Masterchef in which the observation is which player was chosen to leave the show. In this function, argument all_players is sensitive to order, unlike choose_winners() and choose_losers() (an integer n is interpreted as letters[seq_len(n)]). Element i of all_players is the i^{th} player to be eliminated. Thus the first element of all_players is the first player to be eliminated (and would be expected to have the lowest strength). The final element of all_players is the last player to be eliminated (or alternatively the only player not to be eliminated).

Function rank_likelihood() is deprecated: use [S3 generic] supp.ranktable() instead. This takes a character vector of competitors with the order of elements corresponding to the finishing order; a Plackett-Luce likelihood function is returned. Thus v=c("d","b","c","a") corresponds to d coming first, b second, c third, and a fourth. Function race() is an arguably more memorable synonym.

An example of `race()` is given in `inst/rowing.Rmd`, and examples of `ggrl()` are given in `inst/loser.Rmd` and `inst/masterchef.Rmd`.

Author(s)

Robin K. S. Hankin

See Also

`rrank`,`ordertable2supp`,`race3`

Examples

```
W <- hyper2(pnames=letters[1:5])
W1 <- ggrl(W, 'a', letters[2:4],'e')  # 6-element list
W2 <- ggrl(W, 'b', letters[3:5],'a')  # 6-element list

like_single_list(equalp(W1),W1)
like_series(equalp(W1),list(W1,W2))

if(FALSE){  # takes too long
# run 10 races:
r1 <- rrank(10,p=(7:1)/28)
colnames(r1) <- letters[1:7]

# Likelihood function for r1:
suppfun(r1)

# convert a rank table to a support function:
suppfun(wikitable_to_ranktable(volvo_table))

H <- hyper2()
for(i in 1:20){
  H <- H + race(sample(letters[1:5],sample(3,1),replace=FALSE))
}
equalp.test(H) # should not be significant (null is true)

H1 <- hyper2(pnames=letters[1:5])
H2 <- choose_losers(H1,letters[1:4],letters[1:2])   # {a,b} vs {c,d}; {a,b} lost
maxplist(H2,control=list(maxit=1))  # control set to save time
}
```

`gradient` *Differential calculus*

Description

Given a `hyper2` object and a point in probability space, function `gradient()` returns the gradient of the log-likelihood; function `hessian()` returns the bordered Hessian matrix. By default, both functions are evaluated at the maximum likelihood estimate for p, as given by `maxp()`.

Usage

```
gradient(H, probs=indep(maxp(H)))
hessian(H,probs=indep(maxp(H)),border=TRUE)
hessian_lowlevel(L, powers, probs, pnames,n)
is_ok_hessian(M, give=TRUE)
```

Arguments

H	A hyper2 object
L, powers, n	Components of a hyper2 object
probs	A vector of probabilities
pnames	Character vector of names
border	Boolean, with default TRUE meaning to return the bordered Hessian and FALSE meaning to return the Hessian (warning: this option does not respect the unit sum constraint)
M	A bordered Hessian matrix, understood to have a single constraint (the unit sum) at the last row and column; the output of hessian(border=TRUE)
give	Boolean with default FALSE meaning for function is_ok_hessian() to return whether or not M corresponds to a negative-definite matrix, and TRUE meaning to return more details

Details

Function gradient() returns the gradient of the log-likelihood function. If the hyper2 object is of size n, then argument probs may be a vector of length $n-1$ or n; in the former case it is interpreted as indep(p). In both cases, the returned gradient is a vector of length $n-1$. The function returns the derivative of the loglikelihood with respect to the $n-1$ independent components of $(p_1, \ldots, p_n)$, namely $(p_1, \ldots, p_{n-1})$. The fillup value p_n is calculated as $1-(p_1+\cdots+p_{n-1})$.

Function gradientn() returns the gradient of the loglikelihood function but ignores the unit sum constraint. If the hyper2 object is of size n, then argument probs must be a vector of length n, and the function returns a named vector of length n. The last element of the vector is not treated differently from the others; all n elements are treated as independent. The sum need not equal one.

Function hessian() returns the *bordered Hessian*, a matrix of size $n+1 \times n+1$, which is useful when using Lagrange's method of undetermined multipliers. The first row and column correspond to the unit sum constraint, $\sum p_1 = 1$. Row and column names of the matrix are the pnames() of the hyper2 object, plus "usc" for "Unit Sum Constraint".

The unit sum constraint borders could have been added with idiom magic::adiag(0,pad=1,hess), which might be preferable.

Function is_ok_hessian() returns the result of the second derivative test for the maximum likelihood estimate being a local maximum on the constraint hypersurface. This is a generalization of the usual unconstrained problem, for which the test is the Hessian's being negative-definite.

Function hessian_lowlevel() is a low-level helper function that calls the C++ routine.

Further examples and discussion is given in file inst/gradient.Rmd. See also the discussion at maxp on the different optimization routines available.

Value

Function `gradient()` returns a vector of length $n-1$ with entries being the gradient of the log-likelihood with respect to the $n-1$ independent components of $(p_1,\ldots,p_n)$, namely $(p_1,\ldots,p_{n-1})$. The fillup value p_n is calculated as $1-(p_1,\ldots,p_{n-1})$.

If argument `border` is `TRUE`, function `hessian()` returns an n-by-n matrix of second derivatives; the borders are as returned by `gradient()`. If `border` is `FALSE`, ignore the fillup value and return an $n-1$-by-$n-1$ matrix.

Calling `hessian()` at the evaluate will not return exact zeros for the constraint on the fillup value; `gradient()` is used and this does not return exact zeros at the evaluate.

Author(s)

Robin K. S. Hankin

Examples

```
data(chess)
p <- c(1/2,1/3)
delta <- rnorm(2)/1e5  # delta needs to be quite small

deltaL  <- loglik(p+delta,chess) - loglik(p,chess)
deltaLn <- sum(delta*gradient(chess,p + delta/2))   # numeric

deltaL - deltaLn  # should be small [zero to first order]

H <- hessian(icons)
is_ok_hessian(H)
```

---

handover — *Dataset on communication breakdown in handover between physicians*

Description

Object `handover` is a likelihood function corresponding to a dataset arising from 69 medical malpractice claims and concerns handover (or hand-off) between physicians. This dataset was analysed by Lin et al. (2009), and further analysed by Altham and Hankin (2010). The computational methods are presented in the (unmaintained) **hyperdirichlet** and **aylmer** packages and a further discussion is given in the "integration" vignette of the **hyper2** package. The original dataset is `handover_table`, a three-by-three matrix of counts.

Usage

```
data(handover)
```

Details

These objects can be generated by running script `inst/handover.Rmd`, which includes some further discussion and technical documentation, and creates file `handover.rda` which resides in the `data/` directory.

References

- Y. Lin and S. Lipsitz and D. Sinha and A. A. Gawande and S. E. Regenbogen and C. C. Greenberg, 2009. "Using Bayesian p-values in a 2×2 table of matched pairs with incompletely classified data". *Journal of the Royal Statistical Society, Series C*, 58:2
- P. M. E. Altham and R. K. S. Hankin, 2010. "Using recently developed software on a 2×2 table of matched pairs with incompletely classified data". *Journal of the Royal Statistical Society, series C*, 59(2): 377-379
- R. K. S. Hankin 2010. "A generalization of the Dirichlet distribution". *Journal of Statistical software*, 33:11
- L. J. West and R. K. S. Hankin 2008. "Exact tests for two-way contingency tables with structural zeros". *Journal of Statistical software*, 28:11

Examples

```
data(handover)
maxp(handover)
```

`head.hyper2` *First few terms of a distribution: DEPRECATED*

Description

First few terms in a hyperdirichlet distribution

Usage

```
## S3 method for class 'hyper2'
head(x, ...)
```

Arguments

`x`	Object of class `hyper2`
`...`	Further arguments, passed to `head()`

Details

Function is `x[head(brackets(x), ...)]`

Value

Returns a `hyper2` object

Author(s)

Robin K. S. Hankin

Examples

```
p <- zipf(5)
names(p) <- letters[1:5]
H <- suppfun(rrank(20,p))
head(H)
```

---

hepatitis *Hepatitis dataset discussed by Agresti*

Description

A dataset considered by Agresti

Usage

```
data(hepatitis)
```

Format

A `hyper2` object that gives a likelihood function

Details

Object `hepatitis_table` is drawn from Agresti, table 12.16, page 533. Object `hepatitis` is a likelihood function of class `lsl` and `hepatitis_maxp` a pre-calculated evaluate.

These objects can be generated by running script `inst/hepatitis.Rmd`, which includes some further discussion and technical documentation, and creates file `hepatitis.rda` which resides in the `data/` directory.

References

A. Agresti, 2002. "Categorical data analysis". John Wiley and Sons. Table 13.1, p542.

See Also

`race3`,`hepatitis`

Examples

```
pie(hepatitis_maxp)
```

hyper2 *Basic functions in the hyper2 package*

Description

Basic functions in the hyper2 package

Usage

```
hyper2(L=list(), d=0, pnames)
## S3 method for class 'hyper2'
brackets(H)
## S3 method for class 'hyper2'
powers(H)
## S3 method for class 'hyper2'
pnames(H)
## S3 method for class 'suplist'
pnames(H)
size(H)
as.hyper2(L,d,pnames)
is.hyper2(H)
is_valid_hyper2(L,d,pnames)
is_constant(H)
```

Arguments

H	A hyper2 object
L	A list of character vectors whose elements specify the brackets of a hyper2 object
d	A vector of powers; `hyper2()` recycles *only if* `d` is of length 1
pnames	A character vector specifying the names of p_1 through p_n.

Details

These are the basic functions of the hyper2 package. Function `hyper()` is the low-level creator function; `as.hyper2()` is a bit more user-friendly and attempts to coerce its arguments into a suitable form; for example, a matrix is interpreted as rows of brackets.

Functions `pnames()` and `pnames<-()` are the accessor and setter methods for the player names. Length-zero character strings are acceptable player names. The setter method `pnames<-()` can be confusing. Idiom such as `pnames(H) <- value` does not change the likelihood function of `H` (except possibly its domain). When called, it changes the `pnames` internal vector, and will throw an error if any element of `c(brackets(H))` is not present in `value`. It has two uses: firstly, to add players who do not appear in the brackets; and secondly to rearrange the `pnames` vector (the canonical use-case is `pnames(H) <- rev(pnames(H))`). If you want to change the player names, use `psubs()` to substitute players for other players.

Function `is_valid_hyper2()` tests for valid input, returning a Boolean. This function returns an error if a bracket contains a repeated element, as in `hyper2(list(c("a","a")),1)`.

Note that it is perfectly acceptable to have an element of `pnames` that is not present in the likelihood function (this would correspond to having no information about that particular player).

Function `size()` returns the (nominal) length n of nonnegative vector $p = (p_1, \ldots, p_n)$ where $p_1 + \cdots + p_n = 1$.

Author(s)

Robin K. S. Hankin

See Also

`Ops.hyper2`, `Extract.hyper2`, `loglik`, `hyper2-package` `psubs`

Examples

```
(o <- hyper2(list("a","b","c",c("a","b"),letters[1:3]),c(1:3,-1,-5)))
(p <- hyper2(list("a",c("a","d")),c(1,-1)))
o+p


# Verify that the MLE is invariant under reordering
pnames(icons) <- rev(pnames(icons))
maxp(icons) - icons_maxp # should be small
```

`hyper3` *Weighted probability vectors:* `hyper3` *objects*

Description

Objects of class `hyper3` are a generalization of `hyper2` objects that allow the brackets to contain weighted probabilities.

As a motivating example, suppose two players with Bradley-Terry strengths p_1, p_2 play chess where we quantify the first-mover advantage with a term λ. Suppose p_1 plays white $a + b$ times with a wins and b losses, and plays black $c + d$ times with c wins and d losses. Then a sensible likelihood function might be

$$\left(\frac{\lambda p_1}{\lambda p_1 + p_2}\right)^a \left(\frac{p_2}{\lambda p_1 + p_2}\right)^b \left(\frac{p_1}{p_1 + \lambda p_2}\right)^c \left(\frac{\lambda p_2}{p_1 + \lambda p_2}\right)^d$$

If $a = 1, b = 2, c = 3, d = 4$ and $\lambda = 1.3$, appropriate package idiom might be:

```
H <- hyper3()
H[c(p1=1.3)]      %<>% inc(1) # a=1
H[c(p2=1)]        %<>% inc(2) # b=2
H[c(p1=1.3,p2=1)] %<>% dec(3) # a+b=1+2=3
H[c(p1=1)]        %<>% inc(3) # c=3
H[c(p2=1.3)]      %<>% inc(4) # d=4
H[c(p1=1,p2=1.3)] %<>% dec(7) # c+d=3+4=7
H
> log( (p1=1)^3 * (p1=1, p2=1.3)^-7 * (p1=1.3)^1 * (p1=1.3, p2=1)^-3 *
(p2=1)^2 * (p2=1.3)^4)
```

The general form of terms of a `hyper3` object would be $(w_1 p_1 + \cdots + w_r p_r)^{\alpha}$; the complete object would be

$$\mathcal{L}(p_1, \ldots, p_n) = \prod_{j=1}^{N} \left(\sum_{i=1}^{n} w_{ij} p_i \right)^{\alpha_i}$$

where we understand that $p_n = 1 - \sum_{i=1}^{n-1} p_i$; many of the weights might be zero. We see that the weights w_{ij} may be arranged as a matrix and this form is taken by function `hyper3_m()`.

Usage

```
hyper3(B = list(), W = list(), powers = 0, pnames)
hyper3_bw(B = list(), W = list(), powers = 0, pnames)
hyper3_nv(L=list(),powers=0,pnames)
hyper3_m(M,p,stripzeros=TRUE)
```

Arguments

`B`	A list of brackets
`W`	A list of weights
`L`	A list of named vectors
`powers`	Numeric vector of powers
`pnames`	Character vector of player names
`M`	Matrix of weights, column names being player names
`p`	Vector of powers, length equal to `ncol(M)`
`stripzeros`	Boolean with default `TRUE` meaning to silently remove all-zero rows of `M`

Details

- Function `hyper3()` is the user-friendly creation method, which dispatches to a helper function depending on its arguments.
- Function `hyper3_bw()` takes a list of brackets (character vectors) and a list of weights (numeric vectors) and returns a `hyper3` object.
- Function `hyper3_nv()` takes a list of named vectors and returns a `hyper3` object.

- Function `hyper3_m()` takes a matrix with rows being the brackets (entries are weights) and a numeric vector of powers.
- Function `evaluate3()` is a low-level helper function that evaluates a log-likelihood at a point in probability space. Don't use this: use the user-friendly `loglik()` instead, which dispatches to `evaluate3()`.
- Function `maxp3()` is a placeholder (it is not yet written). But the intention is that it will maximize the log-likelihood of a `hyper3` object over the Bradley Terry strengths *and* any weights given. This might not be possible as currently envisaged; I present some thoughts in `inst/kka.Rmd`.
- Function `list2nv()` converts a list of character vectors into a named vector suitable for use as argument `e` of function `cheering3()`. It is used in `inst/global_liveability_ranking.Rmd`.
- Function `as.namedvectorlist()` takes a `hyper3` object and returns a disoRdered list of named vectors corresponding to the brackets and their weights.
- Function `setweight()` alters the weight of every occurrence of a set of players. It is vectorised, so `setweight(H,c("a","b"),88:89)` sets the weight of `a` to 88 and `b` to 89. Replacement methods are defined, so "`H["a"] <- as.weight(3)`" will set the weight of every occurrence of player `a` to 3. If `H` is a `hyper2` object, it will be coerced to `hyper3`.

Value

Generally return or deal with `hyper3` objects

Note

Functionality for `hyper3` objects is generally indicated by adding a "`3`" to function names, e.g. `gradient()` goes to `gradient3()`.

Vignette `hyper3\_creation` discusses the different creation methods for `hyper3` objects.

Author(s)

Robin K. S. Hankin

See Also

`hyper2`

Examples

```
hyper3(B=list("a",c("a","b"),"b"),W=list(1.2,c(1.2,1),1),powers=c(3,4,-7))
hyper3(list(c(a=1.2),c(b=1),c(a=1.2,b=1)),powers=c(3,4,-7))
## Above two objects should be identical.

## Third method, send a matrix:
M <- matrix(rpois(15,3),5,3)
colnames(M) <- letters[1:3]
hyper3(M,c(2,3,-1,-5,1))   # second argument interpreted as powers
```

```
## Standard way to generate a hyper3 object is to create an empty object
## and populate it using the replacement methods:

a <- hyper3()  # default creation method [empty object]

a[c(p1=1.3)] <- 5
a[c(p2=1  )] <- 2
a[c(p1=1.3,p2=1)] <- -7
a

chess3  # representative simple hyper3 object

H1 <- rankvec_likelihood(letters[sample(6)])
H2 <- rankvec_likelihood(letters[sample(6)])
H1["a"] <- as.weight(1.2)         # "a" has some disadvantage in H1
H1[c("b","c")] <- as.weight(2:3)  # "b" and "c" have some advantage in H1
H2[c("c","d")] <- as.weight(1.5)  # "c" and "d" have some advantage in H2
H1+H2
```

`icons` *Dataset on climate change due to O'Neill*

Description

Object `icons_matrix` is a matrix of nine rows and six columns, one column for each of six icons relevant to climate change. The matrix entries show the number of respondents who indicated which icon they found most concerning. The nine rows show different classes of respondents who were exposed to different subsets (of size four) of the six icons.

The columns correspond to the different stimulus icons used, detailed below. An extensive discussion is given in West and Hankin 2008, and Hankin 2010; an updated analysis is given in the `icons` vignette.

Object `icons` is the corresponding likelihood function, which can be created with `saffy(icons_matrix)`.

The object is used in `inst/ternaryplot_hyper2.Rmd` which shows a ternary plot of random samples.

Usage

```
data(icons)
```

Details

The six icons were used in this study were:

PB polar bears, which face extinction through loss of ice floe hunting grounds

NB The Norfolk Broads, which flood due to intense rainfall events

L London flooding, as a result of sea level rise

THC The Thermo-haline circulation, which may slow or stop as a result of anthropogenic modification of the hydrological cycle

OA Oceanic acidification as a result of anthropogenic emissions of carbon dioxide

WAIS The West Antarctic Ice Sheet, which is calving into the sea as a result of climate change

Author(s)

Robin K. S. Hankin

Source

Data kindly supplied by Saffron O'Neill of the University of East Anglia

References

- S. J. O'Neill and M. Hulme 2009. *An iconic approach for representing climate change*. Global Environmental Change, 19:402-410
- I. Lorenzoni and N. Pidgeon 2005. *Defining Dangers of Climate Change and Individual Behaviour: Closing the Gap*. In *Avoiding Dangerous Climate Change* (conference proceedings), UK Met Office, Exeter, 1-3 February
- R. K. S. Hankin 2010. "A generalization of the Dirichlet distribution". *Journal of Statistical software*, 33:11

See Also

`matrix2supp`

Examples

```
data(icons)
pie(icons_maxp)
equalp.test(icons)
```

`increment` *Increment and decrement operators*

Description

Syntactic sugar for incrementing and decrementing likelihood functions.

Frankly they don't do anything that `magrittr:add()` and `magrittr:subtract()` don't (except have a default value of 1 (which is surprisingly useful).

Usage

```
inc(H, val = 1)
dec(H, val = 1)
trial(winners,players,val=1)
```

Arguments

H	A hyper2 object
winners, players	Numeric or character vectors specifying the winning team and the losing team
val	Numeric

Details

A very frequent operation is to increment a single term in a hyper2 object. If

```
> H <- hyper2(list("a",c("a","b"),"c",c("a","b","c")),c(1:3,-6))
> H
log( a * (a + b + c)^-6 * b^2 * c^3)
```

Suppose we wish to increment the power of a+b. We could do:

```
H[c("a","b")] <- H[c("a","b")] + 1
```

(see the discussion of hyper2_sum_numeric at Ops.hyper2.Rd; also vignette zeropower). Alternatively we could use magrittr pipes:

```
H[c("a","b")] %<>% add(1)
```

But inc and dec furnish convenient idiom to accomplish the same thing:

```
H[c("a","b")] %<>% inc
```

Functions inc and dec default to adding or subtracting 1, but other values can be supplied:

```
H[c("a","b")] %<>% inc(3)
```

Or even

```
H[c("a","b")] %<>% inc(H["a"])
```

The convenience function trial() takes this one step further and increments the 'winning team' and decrements the bracket containing all players. The winners are expected to be players.

```
> trial(c("a","b"),c("a","b","c"))
> (a + b) * (a + b + c)^-1
```

Using trial() in this way ensures that the powers sum to zero.

```
  H <- trial(c("a","b"),c("a","b","c"))
  H %<>% inc(trial("a",c("a","b")))
  H
```

The inc and dec operators and the trial() function are used in inst/kka.Rmd.

Author(s)

Robin K. S. Hankin

Examples

```
data(chess)

## Now suppose we observe an additional match, in which Topalov beats
## Anand.  To incorporate this observation into the LF:



trial("a",c("a","b"))

chess <- chess + trial("Topalov",c("Topalov","Anand"))
```

`interzonal` *1963 World Chess Championships*

Description

Likelihood functions for players' strengths in the fifth Interzonal tournament which occurred as part of the 1963 Chess world Championships in Stockholm, 1962.

Details

(there are three chess datasets in the package, documented at `interzonal.Rd` [the 1963 World championship], `kka.Rd` [Karpov-Kasparov-Anand dataset], and `chess.Rd` [rock-paper-scissors using Topalov-Anand-Karpov])

The 1963 World Chess Championship was notable for allegations of Soviet collusion. Specifically, Fischer publicly alleged that certain Soviet players had agreed in advance to draw all their games. The championship included an "interzonal" tournament in which 23 players competed in Stockholm; and a "Candidates" tournament in which 8 players competed in Curacao.

Likelihood functions `interzonal` and `interzonal_collusion` are created by files '`inst/interzonal.Rmd`', which is heavily documented and include some analysis. Object `interzonal` includes a term for drawing, ("`draw`"), assumed to be the same for all players; object `interzonal_collusion` includes in addition to `draw`, a term for the drawing in Soviet-Soviet matches, "`coll`".

Some other analysis is given in files `inst/curacao11962_threeplayers.R` and `inst/curacao1962_threeplayers_rest_`

See Also

chess,karpov_kasparov_anand

Examples

```
pie(interzonal_maxp)

# samep.test(interzonal,c("Fischer","Geller")) # takes too long
```

javelin *Javelin dataset*

Description

Results from the men's javelin, 2020 Summer Olympics.

- `javelin_table`, a dataframe in the form of an "attempts table", detailing the throw distances of eight competitors (diacritics have been removed) for each of six throws
- `javelin1` and `javelin2` Support functions corresponding to the weighted Plackett-Luce likelihood. The suffix "1" means that no-throws are counted as losing attempts; suffix "2" means that no-throws are ignored.
- `javelin1_maxp` and `javelin2_maxp` are the corresponding maximum likelihood estimates for the players' strengths
- javelin_vector is a named vector with elements being the throw distances and names being the thrower

Usage

```
data(javelin)
```

Format

As detailed above

Details

These objects can be generated by running script `inst/javelin.Rmd`, which includes some further discussion and technical documentation, and creates file `javelin.rda` which resides in the `data/` directory.

See Also

`attemptstable2supp3`

Examples

```
pie(javelin1_maxp)
```

`jester` *Jester dataset*

Description

A likelihood function for the Jester datasets

Usage

```
data(jester)
```

Details

Object `jester` is a likelihood function for the 91 jokes rated by the first 150 respondents in file '`jester_dataset_1_3.zip`', taken from Goldberg et al. Object `jester_maxp` is the result of running `maxp(jester)`. The results table of (nearly) all jokes and respondents is given as `jester_table` in which each row is a joke and each column a respondent.

The dataset is interesting because it has been analysed by many workers, including Goldberg, for patterns; here I assume that all the respondents behave identically (but randomly). It is included here because it is a very severe numerical challenge in the context of the `hyper2` package. I am not convinced that `maxjest` is even close to the true evaluate.

Objects `jester`, `jester_table`, and `jester_maxp` can be generated by running script '`inst/jester.Rmd`', which includes some further technical documentation. This file takes about 10 minutes to run.

References

Eigentaste: A Constant Time Collaborative Filtering Algorithm. Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. *Information Retrieval*, 4(2), 133-151. July 2001.

Examples

```
# maxp(jester)  # takes too long

# Note that the possibly poor identification of the evaluate
#  nevertheless allows us to reject the null of equality:

(LAM <- -2*(loglik(equalp(jester),jester)-loglik(jester_maxp,jester)))
pval <- pchisq(LAM,df=size(jester),lower.tail=FALSE)
```

karate *Karate dataset*

Description

Dataset from the 2018 World Karate Championships, men's 67kg. It is an example of a dataset with too many degrees of freedom to be analysed easily by the package.

Usage

```
data(karate)
```

Details

Object `karate_table` is a dataframe of results showing results from the 2018 World Karate Championships, men's 67kg; `karate` is the associated likelihood function. There are two maximum likelihood estimates given; `karate_maxp`, the evaluate as returned by `maxp()`, and `karate_maxp`, returned by `zermelo()` [the value given by `maxp()` itself is less likely].

These objects can be generated by running script `inst/karate.Rmd`, which includes some further discussion and technical documentation and creates file `karate.rda` which resides in the `data/` directory.

Note

Table `karate_table` misses uninformative matches, that is, competitions with 0-0 results.

References

https://en.wikipedia.org/wiki/2018_World_Karate_Championships

See Also

zapweak

Examples

```
summary(karate)
```

karpov_kasparov_anand *Karpov, Kasparov, Anand*

Description

Data of three chess players: Karpov, Kasparov, and Anand. Includes two likelihood functions for the strengths of the players, and an array of game results

Details

(there are three chess datasets in the package, documented at `interzonal.Rd` [the 1963 World championship], `kka.Rd` [Karpov-Kasparov-Anand dataset], and `chess.Rd` [rock-paper-scissors using Topalov-Anand-Karpov])

The strengths of chess players may be assessed using the generalized Bradley-Terry model. The `karpov_kasparov_anand` `hyper2` likelihood function allows one to estimate the players' strengths, propensity to draw, and also the additional strength conferred by playing white as personified by a draw monster and a white monster `draw` and `white` respectively.

Object `karpov_kasparov_anand` assumes that the draw potential is the same for all three players; likelihood function `kka_3draws` allows the propensity to draw to differ between the three players.

The reason that the players are different from those in the `chess` dataset is that the original data does not seem to be available any more.

Dataset `kka` refers to scorelines of matches between three chess players (Kasparov, Karpov, Anand). It is a named numeric vector with names such as '`karpov_plays_white_beats_kasparov`' which has value 18: we have a total of 18 games between Karpov and Kasparov in which Karpov played white and beat Kasparov.

Object `chess3` is a simple `hyper3` object corresponding to pairwise comparison with draws; `chess3_maxp` is the evaluate, conditional on the estimated white-player advantage and draw proclivity. This object is created and discussed in `inst/kka.Rmd`. Array `kka_array` presents the same information in a 3D array.

All data drawn from `https://www.chessgames.com` (search for "Kasparov vs Karpov", etc). Note that the database allows one to sort by white wins or black wins (there is a 'refine search' tab at the bottom). Some searches have more than one page of results. Numbers here downloaded 17 February 2019. Note that only 'classical games' are considered here (rapid and exhibition games being ignored).

These objects can be generated by running script `inst/kka.Rmd`, which includes some further discussion and technical documentation and creates file `kka.rda` which resides in the `data/` directory.

See Also

`chess`

Examples

```
karpov_kasparov_anand
# pie(maxp(karpov_kasparov_anand))  # takes ~10s

M <- kka_array[,,1] + 1i*kka_array[,,3]
home_away(M)
home_away3(M,lambda=1.2)
```

---

keep *Keep or discard players*

Description

Flawed functionality to keep or discard subsets of the players in a hyper2 object or order table.

Usage

```
discard_flawed2(x, unwanted,...)
keep_flawed(H, wanted)
discard_flawed(H, unwanted)
```

Arguments

H	A hyper2 object
x	An order table
wanted, unwanted	Players to keep or discard. May be character or integer or logical
...	Further arguments passed to wikitable_to_ranktable(), notably points

Details

Do not use these functions. They are here as object lessons in poor thinking. To work with a subset of competitors, see the example at as.ordertable.

Functions keep_flawed2() and discard_flawed2() take an order table and keep or discard specified rows, returning a reduced order table. This is not a trivial operation.

Functions keep_flawed() and discard_flawed() will either keep or discard players specified in the second argument. It is not clear to me that these functions have any reasonable probabilistic interpretation and file inst/retain.Rmd gives a discussion.

Given a wikitable or ordertable, it is possible to create a likelihood function based on a subset of rows using the incomplete=TRUE argument; see the example at ?ordertable2supp. But this method is flawed too because it treats non-finishers as if they finished in the order of their rows.

Function as.ordertable() is the correct way to consider a subset of players in a wikitable.

Author(s)

Robin K. S. Hankin

See Also

ordertable2supp,tidy

Examples

```
maxp(icons)
discard_flawed(icons,c("OA","WAIS"))

## Not run: # (takes too long)
data("skating")
maxp(skating)[1:4]      # numbers work, keep the first four skaters
maxp(keep_flawed(skating,pnames(skating)[1:4])) # differs!

## End(Not run)
```

---

length.hyper2 *Length method for hyper2 objects*

Description

Length method for hyper2 objects, being the number of different brackets in the expression

Usage

```
## S3 method for class 'hyper2'
length(x)
```

Arguments

x	hyper2 object

Author(s)

Robin K. S. Hankin

Examples

```
data("oneill")
length(icons)
seq_along(icons)
```

loglik *Log likelihood functions*

Description

Returns a log-likelihood for a given hyper2 or hyper3 object at a specific point in probability space

Usage

```
loglik(p, H, log = TRUE)
loglik_single(p,H,log=TRUE)
like_single_list(p,Lsub)
like_series(p,L,log=TRUE)
```

Arguments

H	An object of class hyper2 or hyper3
p	A probability point. See details
log	Boolean with default TRUE meaning to return the log-likelihood and FALSE meaning to return the likelihood
L, Lsub	A list of hyper2 objects, or a list of list of loglik objects

Details

Function loglik() is a straightforward likelihood function. It can take a vector of length n=size(H) or size(H)-1. If given the vector $p = (p_1, \ldots, p_{n-1})$ it appends the fillup value, and then returns the (log) likelihood (names are discarded in this case). If given a vector $p = (p_1, \ldots, p_n)$ [notionally summing to 1] it requires a named vector, and names must match those of H. The vector is reordered if necessary.

If p is a matrix, the rows are interpreted as probability points.

Function loglik_single() is a helper function that evaluates a likelihood function single point in probability space. Functions like_single_list() and like_series() are intended for use with ggrl().

Note

Likelihood is defined up to an arbitrary multiplicative constant. Log-likelihood (also known as *support*) is defined up to an arbitrary additive constant.

If function loglik() is given a probability vector of length n, the vector must satisfy the unit sum constraint (up to a small tolerance). Also, it must be a named vector with names matching the pnames of argument H.

```
> pnames(chess)
[1] "Topalov" "Anand"   "Karpov"
> loglik(c(Topalov=0.7,Anand=0.2,Karpov=0.1),chess)
[1] -69.45364
> loglik(c(Karpov=0.1,Topalov=0.7,Anand=0.2),chess)  # identical, just a different order
[1] -69.45364
```

But if given a vector of length `n-1` [e.g. the value of `indep()`], then the names are ignored and the entries are interpreted as the BT strengths of `pnames(H)[seq_len(n-1)]`:

```
> loglik(c(0.7,0.2),chess)
[1] -69.45364
> loglik(c(foo=0.7,bar=0.2),chess)  # names are ignored
[1] -69.45364
```

(the above applies for `H` a `hyper2` or `hyper3` object).

Empty brackets are interpreted consistently: that is, zero whatever the probability vector (although the print method is not perfect).

Author(s)

Robin K. S. Hankin

See Also

`maxp`

Examples

```
data(chess)
loglik(c(1/3,1/3),chess)

loglik(rp(14,icons),icons)

## Not run:  # takes too long
like_series(masterchef_maxp,masterchef)
like_series(indep(equalp(masterchef)),masterchef)

## End(Not run)

W <- hyper2(pnames=letters[1:6])
W1 <- ggrl(W, 'a', letters[2:5],'f')              # 24-element list
W2 <- ggrl(W, c('a','b'), c('c','d'),c('e','f'))  # 2^3=8 element list

like_single_list(rep(1/6,5),W1)      # information from first observation
like_series(rep(1/6,5),list(W1,W2))  # information from both observations

# hyper3 objects:
```

```
H3 <- ordervec2supp3(letters[c(1,2,3,3,2,1,2)])
loglik(c(a=1,b=2,c=3)/6,H3)
loglik(c(a=1,c=3,b=2)/6,H3) # identical
```

`masterchef` *Masterchef series 6*

Description

Data from Australian Masterchef Series 6

Usage

```
data(masterchef)
```

Format

Object `masterchef` is a list of `hyper2` objects; `masterchef_pmax` and `masterchef_constrained_pmax` are named vectors with unit sum.

Details

The object is created using the code in `inst/masterchef.Rmd`, which is heavily documented. Not all the information available is included in the likelihood function as some of the early rounds result in an unmanageably large list. Inclusion is controlled by Boolean vector `doo`.

The definitive source is the coloured table on the wiki page.

References

Wikipedia contributors, "MasterChef Australia (series 6)," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=MasterChef_Australia_(series_6)&oldid=758432561 (accessed January 5, 2017).

See Also

`ggrl`

Examples

```
a1 <- indep(equalp(masterchef[[1]]))           # equal strengths
a2 <- indep(masterchef_maxp)               # MLE
a3 <- indep(masterchef_constrained_maxp)   # constrained MLE

## Not run:  # takes too long
like_series(a1, masterchef)
like_series(a2, masterchef)
like_series(a3, masterchef)

## End(Not run)
```

`matrix2supp` *Convert a matrix to a likelihood function*

Description

Functions to convert matrix observations to likelihood functions. Each row is an observation of some kind, and each column a player.

Function `ordertable2supp()` is documented separately at `ordertable2supp`.

Usage

```
saffy(M)
volley(M)
```

Arguments

`M`	A matrix of observations

Details

Two functions are documented here:

- `saffy()`, which converts a matrix of restricted choices into a likelihood function; it is named for Saffron O'Neill. The canonical example would be Saffron's climate change dataset, documented at `icons`. Function `saffy()` returns the appropriate likelihood function for the dataset.
- `volley()`, which converts a matrix of winning and losing team members to a likelihood function. The canonical example is the volleyball dataset. Each row is a volleyball game; each column is a player. An entry of 0 means "on the losing side", an entry of 1 means "on the winning side", and an entry of `NA` means "did not play".

Author(s)

Robin K. S. Hankin

See Also

`icons`,`volleyball`

Examples

```
icons == saffy(icons_table)  # should be TRUE

volley(volleyball_table) == volleyball # also should be TRUE
```

maxp *Maximum likelihood estimation*

Description

Find the maximum likelihood estimate for p, also equal probabilities

Usage

```
maxp(H, startp=NULL, give=FALSE, fcm=NULL, fcv=NULL, SMALL=1e-6, n=1,
   show=FALSE, justlikes=FALSE, ...)
maxplist(Hlist, startp=NULL, give=FALSE, fcm=NULL, fcv=NULL, SMALL=1e-6, ...)
maxp_single(H, startp=NULL, give=FALSE, fcm=NULL, fcv=NULL, SMALL=1e-6,
   maxtry=100, ...)
maxp_single2(H, startp=NULL, give=FALSE, fcm=NULL, fcv=NULL, SMALL=1e-6,
   maxtry=100, ...)
maxp_simplex(H, n=100, show=FALSE, give=FALSE, ...)
maxp_lsl(HLSL, startp = NULL, give = FALSE, fcm = NULL, fcv = NULL, SMALL=1e-6, ...)
equalp(H)
```

Arguments

H	A hyper2 or hyper3 object
Hlist	A list with elements all hyper2 objects
HLSL	An lsl object
startp	A vector of probabilities specifying the start-point for optimization; if a full unit-sum vector, then the fill-up value will be removed by indep() (except for maxp_lsl())
give	Boolean, with default FALSE meaning to return just the evaluate (including fillup), and TRUE meaning to return the entire formal output of the optimization routine. In function maxp(), a non-logical value [such as 0] gives only the two most important pieces of information, viz the evaluate and a log-likelihood
fcm, fcv	Further problem-specific constraints
n	Number of start points to use
show	Boolean, with TRUE meaning to show successive estimates
justlikes	Boolean, with TRUE meaning to return just a vector of estimated likelihoods
SMALL	Numerical minimum for probabilities
maxtry	Integer specifying maximum number of times to try constrOptim() with slightly differing start points, to avoid a known R bug which reports wmmin is not finite, bugzilla id 17703
...	Further arguments which maxp() passes to constrOptim()

Details

Function maxp() returns the maximum likelihood estimate for p, which has the unit sum constraint implemented.

Function maxplist() does the same but takes a list of hyper2 objects (for example, the output of ggrl()). Note that maxplist() does not have access to the gradient of the objective function, which makes it slow.

If function maxp() is given a suplist object it dispatches to maxplist().

Functions maxp_single() and maxp_single2() are helper functions which perform a single constrained optimization using base::constrOptim() or alabama::constrOptim.nl() respectively. The functions should produce identical (or at least very similar) results. They are used by maxp() and maxp_simplex() which dispatch to either maxp_single() or maxp_single2() depending on the value of option use_alabama. If TRUE, they will use (experimental) maxp_single2(), otherwise (default) maxp_single(). Function maxp_single() is prone to the "wmmin not finite" bug [bugzilla id 17703] but on the other hand is a bit slower. I am not sure which one is better at this time.

Function maxp_simplex() is intended for complicated or flat likelihood functions where finding local maxima might be a problem. It repeatedly calls maxp_single(), starting from a different randomly chosen point in the simplex each time. This function does not take fcm or fcv arguments, it operates over the whole simplex (hence the name). Further arguments, . . . , are passed to maxp_single().

The functions do not work for the masterchef_series6 likelihood function. These require a bespoke optimization as shown in the vignette.

Function equalp() returns the value of p for which all elements are the same.

In functions maxp() etc, arguments fcm and fcv implement linear constraints to be passed to constrOptim(). These constraints are in addition to the usual nonnegativity constraints and unitsum constraint, and are added to the ui and ci arguments of constrOptim() with rbind() and c() respectively. The operative lines are in maxp_single():

```
UI <- rbind(diag(nrow = n - 1), -1, fcm)
CI <- c(rep(SMALL, n - 1), -1 + SMALL, fcv)
```

where in UI, the first $n-1$ rows enforce nonnegativity of p_i, $1 \leq p < n$; row n enforces nonnegativity of the fillup value p_n; and the remaining (optional) rows enforce additional linear constraints. Argument CI is a vector with corresponding elements.

Examples of their use are given in the "icons" vignette.

Note

In manpages elsewhere, n=2 is sometimes used. Previous advice was to use n=10 or greater in production work, but I now think this is overly cautious and n=1 is perfectly adequate unless the dimension of the problem is large.

The (bordered) Hessian is given by function hessian(), documented at gradient.Rd; use this to assess the "sharpness" of the maximum.

Function maxp() takes hyper2 or hyper3 objects but it does not currently work with lsl objects; use maxp_lsl().

The built-in datasets generally include a pre-calculated result of running maxp(); thus hyper2 object icons and icons_maxp are included in the same .rda file.

Function maxp() can trigger a known R bug (bugzilla id 17703) which reports "wmmin is not finite". Setting option use_alabama to TRUE makes the package use a different optimization routine.

Author(s)

Robin K. S. Hankin

See Also

gradient,fillup

Examples

```
maxp(icons)

W <- hyper2(pnames=letters[1:5])
W1 <- ggrl(W, 'a', letters[2:3],'d')  # W1 is a suplist object
## Not run: maxp(W1)  # takes a long time to maximize a suplist
```

---

moto *MotoGP dataset*

Description

Race results from the 2019 Grand Prix motorcycling season

Usage

```
data(moto)
```

Details

Object moto_table is a dataframe of results showing ranks of 28 drivers (riders?) in the 2019 FIM MotoGP World Championship. The format is standard, that is, can be interpreted by function ordertable2supp() if the final points column is removed. The corresponding support function is motoGP_2019.

These objects can be generated by running script inst/moto.Rmd, which includes some further discussion and technical documentation and creates file moto.rda which resides in the data/ directory.

Note

Many drivers have names with diacritics, which have been removed from the dataframe.

References

Wikipedia contributors. (2020, February 8). 2019 MotoGP season. In *Wikipedia, The Free Encyclopedia.* Retrieved 08:16, February 20, 2020, from https://en.wikipedia.org/w/index.php?title=2019_MotoGP_season&oldid=939711064

See Also

ordertable2supp

Examples

```
pie(moto_maxp)
```

`mult_grid` *Kronecker matrix product functionality*

Description

Peculiar version of `expand.grid()` for matrices

Usage

```
mult_grid(L)
pair_grid(a,b)
```

Arguments

`L`	List of matrices
`a, b`	Matrices

Details

Function `pair_grid(a,b)` returns a matrix with each column of `a` `cbind()`-ed to each column of `b`.

Function `mult_grid()` takes a list of matrices; it is designed for use by `ggrl()`.

Author(s)

Robin K. S. Hankin

See Also

ggrl

Examples

```
pair_grid(diag(2),diag(3))
mult_grid(lapply(1:4,diag))
```

NBA *Basketball dataset*

Description

A point-by-point analysis of a basketball game

Usage

```
data(NBA)
```

Details

Dataset `NBA_table` is a dataframe contains a point-by-point analysis of a basketball match. Each row corresponds to a point scored. The first column is the time of the score, the second is the number of points scored, the third shows which team had possession at the start of play, and the fourth shows which team scored. The other columns show the players. Table entries show whether or not that particular player was on the pitch when the point was scored.

Likelihood function `NBA` is a `hyper2` object that gives the log-likelihood function for this dataset. There is a player named "`possession`" that is a reified entity representing the effect of possession.

Object `NBA_maxp` is not the result of running `maxp(NBA)`; it was obtained by repeatedly running `maxp_simplex()` on a fault-tolerant system [it triggers a known R bug, bugzilla id 17703, giving a "`wmmin not finite`" error]. It is not clear to me that likelihood function `NBA` has a well-defined global maximum.

Object `NBA` poses difficulty for the numerical optimization routines for some reason.

Note that function `volley()` is not applicable because we need to include possession.

These objects can be generated by running script `inst/NBA.Rmd`, which includes some further discussion and technical documentation and creates file `NBA.rda` which resides in the `data/` directory.

References

https://www.espn.com/nba/playbyplay?gameId=400954514

See Also

volleyball

Examples

```
data(NBA)
dotchart(NBA_maxp)
```

Ops.hyper2 *Arithmetic Ops Group Methods for hyper2 objects*

Description

Allows arithmetic operators "+", "*" and comparison operators "==" and "!=", to be used for hyper2 objects.

Specifically, H1 + H2 implements addition of two log-likelihood functions, corresponding to incorporation of additional independent observational data; and n*H1 implements H1+H1+...+H1, corresponding to repeated independent observations of the same data.

There are no unary operations for this class.

Usage

```
## S3 method for class 'hyper2'
Ops(e1, e2 = NULL)
## S3 method for class 'hyper2'
sum(x,...,na.rm=FALSE)
hyper2_add(e1,e2)
hyper2_sum_numeric(H,r)
```

Arguments

e1, e2	Objects of class hyper2, here interpreted as hyperdirichlet distributions
x, ..., na.rm	In the sum() method, objects to be summed; na.rm is currently ignored
H, r	In function hyper2_sum_numeric(), object H is a hyper2 object and r is a length-one real vector (a number)

Details

If two independent datasets have hyper2 objects H1 and H2, then package idiom for combining these would be H1+H2; the additive notation "+" corresponds to addition of the support (or multiplication of the likelihood). So hyper2 objects are better thought of as support functions than likelihood functions; this is reflected in the print method which explicitly wraps the likelihood function in a "log()".

Idiom H1-H1 returns H1 + (-1)*H2, useful for investigating the difference between likelihood functions arising from two different observations, or different probability models. An example is given in inst/soling.Rmd.

Testing for equality is not straightforward for two implementation reasons. Firstly, the object itself is stored internally as a stl map, which does not store keys in any particular order; and secondly, the stl set class is used for the brackets. A set does not include information about the order of its elements; neither does it admit repeated elements. See examples.

Function hyper2_sum_numeric() is defined so that idiom like icons["L"] + 5 works as expected. This means that icons["L"] <- icons["L"] + 3 and icons["L"] %<>%inc(3) work (without this, one has to type icons["L"] <- powers(icons["L"]) + 3, which sucks).

Raising a hyper2 object to a power returns an error.

Value

Returns a hyper2 object or a Boolean.

Author(s)

Robin K. S. Hankin

Examples

```
chess2 <- hyper2(list("Kasparov","Karpov",c("Kasparov","Karpov")),c(2,3,-5))

chess + chess2

maxp(chess+chess2)
```

Ops.hyper3 *Arithmetic Ops Group Methods for hyper3 objects*

Description

Allows arithmetic operators "+", "*" and comparison operators "==" and "!=", to be used for hyper3 objects.

Specifically, H1 + H2 implements addition of two log-likelihood functions, corresponding to incorporation of additional independent observational data; and n*H1 implements H1+H1+...+H1, corresponding to repeated independent observations of the same data.

Usage

```
## S3 method for class 'hyper3'
Ops(e1, e2 = NULL)
hyper3_add(e1,e2)
hyper3_prod(e1,n)
```

Arguments

e1, e2	Objects of class hyper3
n	Numeric vector of length 1

Details

Pretty much everything documented here is a straightforward translation of the corresponding hyper2 functionality.

Value

Returns a hyper3 object or a Boolean.

Author(s)

Robin K. S. Hankin

Examples

```
H1 <- hyper3(list(c(a=1.2),c(b=1),c(a=1.2,b=1)),powers=c(3,4,-7))
H2 <- hyper3(list(c(a=1.2),c(b=1.2),c(a=2.2,b=1.2)),powers=c(2,3,-5))

H1
H2
```

`ordertable` *Order tables*

Description

Order tables

Details

The package makes extensive use of order tables and these are discussed here together with a list of order tables available in the package as data. See also `ranktable.Rd`.

The prototypical ordertable would be `pentathlon_table`:

```
> pentathlon_table
An ordertable:
             shooting fencing swimming riding running
Moiseev             5       1        1      6       5
Zadneprovskis       6       2        5      5       1
Capalini            4       6        2      3       4
Cerkovskis          3       3        7      7       2
Meliakh             1       7        4      1       6
Michalik            2       4        6      2       7
Walther             7       5        3      4       3
```

Although `pentathlon_table` is a dataset in the package, the source dataset is also included in the `inst/` directory as file `pentathlon.txt`; use idiom like `read.table("inst/pentathlon.txt")` to load the order table.

Object `pentathlon_table` is a representative example of an ordertable. Each row is a competitor, each column an event (venue, judge, ...). The first row shows Moiseev's ranking in shooting (5th), fencing (1st), and so on. The first column shows the ranks of the competitors in shooting. Thus Moiseev came fifth, Zadneprovskis came 6th, and so on.

However, to create a likelihood function we need ranks, not orders. We need to know, for a given event, who came first, who came second, and so on (an extended discussion on the difference

between rank and order is given at rrank). We can convert from an order table to a rank table using `ordertable_to_ranktable()` (see also `ranktable.Rd`):

```
> ordertable_to_ranktable(pentathlon_table)
         c1            c2            c3         c4       c5
shooting Meliakh       Michalik      Cerkovskis Capalini Moiseev
fencing  Moiseev       Zadneprovskis Cerkovskis Michalik Walther
swimming Moiseev       Capalini      Walther    Meliakh  Zadneprovskis
riding   Meliakh       Michalik      Capalini   Walther  Zadneprovskis
running  Zadneprovskis Cerkovskis    Walther    Capalini Moiseev
         c6            c7
shooting Zadneprovskis Walther
fencing  Capalini      Meliakh
swimming Michalik      Cerkovskis
riding   Moiseev       Cerkovskis
running  Meliakh       Michalik
```

Above, we see the same data in a different format (an extended discussion on the difference between rank and order is given in rrank).

Many of the order tables in the package include entries that correspond to some variation on "did not finish". Consider the volvo dataset:

```
> volvo_table
           leg1 leg2 leg3 leg4 leg5 leg6 leg7 leg8 leg9
AbuDhabi      1    3    2    2    1    2    5    3    5
Brunel        3    1    5    5    4    3    1    5    2
Dongfeng      2    2    1    3  DNF    1    4    7    4
MAPFRE        7    4    4    1    2    4    2    4    3
Alvimedica    5    5    3    4    3    5    3    6    1
SCA           6    6    6    6    5    6    6    1    7
Vestas        4  DNF  DNS  DNS  DNS  DNS  DNS    2    6
```

In the above order table, we have `DNF` for "did not finish" and `DNS` for "did not start". The `formula1` order table has other similar entries such as `DSQ` for "disqualified" and a discussion is given at `ordertable2supp.Rd`.

Links are given below to all the order tables in the package. Note that the table in `inst/eurovision.Rmd` (`wiki_matrix`) is not an order table because no country is allowed to vote for itself.

To coerce a table like the Volvo dataset shown above into an order table [that is, replace `DNS` with zeros, and also force nonzero entries to be contiguous], use `as.ordertable()`.

There is an experimental extraction method which extracts certain rows of an ordertable; this is used in `inst/skating.Rmd`.

Author(s)

Robin K. S Hankin

See Also

ordertable2supp,rrank, ranktable,as.ordertable

Examples

```
ordertable_to_ranktable(soling_table)
suppfun(soling_table) == soling  # should be TRUE
```

`ordertable2points` *Calculate points from an order table*

Description

Given an order table and a schedule of points, calculate the points awarded to each competitor.

Usage

```
ordertable2points(o, points,totals=TRUE)
```

Arguments

`o`	Order table
`points`	A numeric vector indicating number of points awarded for first, second, third, etc placing
`totals`	Boolean, with default `TRUE` meaning to return the points for each player (row) and `FALSE` meaning to return the entire table but with orders replaced with points scored

Value

Returns either an order table or a named numeric vector

Author(s)

Robin K. S. Hankin

See Also

`ordertable`

Examples

```
points <- c(25, 18, 15, 12, 10, 8, 6, 4, 2, 1, 0, 0)
o <- as.ordertable(F1_table_2017)
ordertable2points(o,points)
```

`ordertable2supp` *Translate order tables to support functions*

Description

Function `ordertable2supp()` is discouraged: use [S3 generic] `suppfun()` instead.

Usage

```
ordertable2supp(x, noscore, incomplete=TRUE)
ordervec2supp(d)
```

Arguments

`x`	Data frame, see details
`d`	A named numeric vector giving order; zero entries are interpreted as that competitor coming last (due to, e.g., not finishing)
`incomplete`	Boolean, with `FALSE` meaning to insist that each rank $1, 2, ..., n$ is present [zero entries mean "did not place" irregardless]. The default `TRUE` allows for gaps. This is useful if we are considering the first few lines of an ordertable because there might be missing ranks.
`noscore`	Character vector giving the abbreviations for a non-finishing status such as "did not finish" or "disqualified". A missing argument is interpreted as `c("Ret", "WD", "DNS", "DSQ", "DNP", "NC")`

Details

Function `ordertable2supp()` is intended for use on order tables such as found at https://en.wikipedia.org/wiki/2019_Moto3_season. This is a common format, used for Formula 1, motoGP, and other racing sports. Prepared text versions are available in the package in the `inst/` directory, for example `inst/motoGP_2019.txt`. Use `read.table()` to create a data frame which can be interpreted by `ordertable2supp()`.

Function `ordervec2supp()` takes an order vector `d` and returns the corresponding Plackett-Luce loglikelihood function as a `hyper2` object. It requires a named vector; names of the elements are interpreted as names of the players. Use argument `pnames` to supply the players' names (see the examples).

```
> x <- c(b=2,c=3,a=1,d=4,e=5) # a: 1st, b: 2nd, c: 3rd etc
> ordervec2supp(x)
log( a * (a + b + c + d + e)^-1 * (a + b + d + e)^-1 * b * (b + d +
e)^-1 * c * (d + e)^-1 * e)
```

$$\frac{a}{a+b+c+d+e}\cdot\frac{b}{b+c+d+e}\cdot\frac{c}{c+d+e}\cdot\frac{d}{d+e}\cdot\frac{e}{e}$$

Zero entries mean "did not finish":

```
> ordervec2supp(c(b=1,a=0,c=2))  # b: 1st, a: DNF, c: second
log((a + b + c)^-1 * (a + c)^-1 * b * c)
```

$$\frac{b}{a+b+c}\cdot\frac{c}{a+c}$$

Note carefully the difference between `ordervec2supp()` and `rankvec_likelihood()`, which takes a character vector:

```
>  names(sort(x))
[1] "a" "b" "c" "d" "e"
> rankvec_likelihood(names(sort(x)))
log( a * (a + b + c + d + e)^-1 * b * (b + c + d + e)^-1 * c * (c + d +
e)^-1 * d * (d + e)^-1)
> rankvec_likelihood(names(sort(x))) == ordervec2supp(x)
[1] TRUE
>
```

Function `order_obs()` was used in the integer-indexed paradigm but is obsolete in the name paradigm. A short vignette applying `ordervec2supp()` and `ordertable2supp()` to the salad dataset of the **prefmod** package [and further analysed in the **PlackettLuce** package] is presented at `inst/salad.Rmd`.

Value

Returns a `hyper2` object

Author(s)

Robin K. S. Hankin

See Also

`ordertable`,`ordertable2supp3`

Examples

```
ordertable2supp(soling_table)    # discouraged
suppfun(soling_table)            # use this instead

# competitors a-f, racing at two venues:
x <- data.frame(
    venue1=c(1:5,"Ret"),venue2=c("Ret",4,"Ret",1,3,2),
    row.names=letters[1:6])

## First consider all competitors; incomplete=FALSE checks that all
## finishing competitors have ranks 1-n in some order for some n:

ordertable2supp(x, incomplete=FALSE)   # discouraged
```

```
suppfun(ordertable(x), incomplete=FALSE)           # use this instead


## Now consider just a-d; must use default incomplete=TRUE as at venue2
## the second and third ranked competitors are not present in x[1:4,]:

ordertable2supp(x[1:4,])




## Function ordervec2supp() is lower-level, used for order vectors:

a1 <- c(a=2,b=3,c=1,d=5,e=4) # a: 2nd, b: 3rd, c: 1st, d: 5th, e: 4th
a2 <- c(a=1,b=0,c=0,d=2,e=3) # a: 2nd, b: DNF, c: DNF, d: 2nd, e: 3rd
a3 <- c(a=1,b=3,c=2)         # a: 1st, b: 3rd, c: 2nd. NB only a,b,c competed
a4 <- c(a=1,b=3,c=2,d=0,e=0) # a: 1st, b: 3rd, c: 2nd, d,e: DNF


## results of ordervec2supp() may be added with "+" [if the observations
## are independent]:

H1 <- ordervec2supp(a1) + ordervec2supp(a2) + ordervec2supp(a3)
H2 <- ordervec2supp(a1) + ordervec2supp(a2) + ordervec2supp(a4)

## Thus H1 and H2 are identical except for the third race.  In H1, 'd'
## and 'e' did not compete, but in H2, 'd' and 'e' did not finish (and
## notionally came last):

pmax(H1)
pmax(H2)   # d,e not finishing affects their estimated strength
```

`ordertrans` *Order transformation*

Description

Given an order vector, shuffle so that the players appear in a specified order.

Usage

```
ordertrans(x,players)
ordertransplot(ox,oy,plotlims, ...)
```

Arguments

`x`	A (generalized) order vector

players	A character vector specifying the order in which the players will be listed; if missing, use sort(names(x))
ox, oy	Rank vectors
plotlims	Length two numeric vector giving x and y plot limits. If missing, use sensible default
...	Further arguments, passed to plot()

Details

The best way to describe this function is with an example:

```
> x <- c(d=2,a=3,b=1,c=4)
> x
d a b c
2 3 1 4
```

In the above, we see x is an order vector showing that d came second, a came third, b came first, and c came fourth. This is difficult to deal with because one has to search through the vector to find a particular competitor, or a particular rank. This would be harder if the vector was longer. If we wish to answer the question "where did competitor a come? where did b come?" we would want an *order* vector in which the competitors are in alphabetical order. This is accomplished by ordertrans():

```
> o <- ordertrans(x)
> o
a b c d
3 1 4 2
```

(this is equivalent to o <- x[order(names(x))]). Object o contains the same information as x, but presented differently. This says that a came third, b came first, c came fourth, and d came second. In particular, the Plackett-Luce order statistic is identical:

```
> ordervec2supp(x) == ordervec2supp(o)
> [1] TRUE
```

There is a nice example of ordertrans() in inst/eurovision.Rmd, and package vignette ordertrans provides further discussion and examples.

Function ordertrans() takes a second argument which allows the user to arrange an order vector into the order specified.

Function ordertrans() also works in the context of hyper3 objects:

```
x <- c(d=2,a=3,b=1,a=4)
x
d a b a
2 3 1 4
ordertrans(x)
a a b d
3 4 1 2
```

Object x shows that d came second, a came third and fourth, and b came first. We can see that ordertrans() gives the same information in a more intelligible format. This functionality is useful in the context of hyper3 likelihood functions.

Value

Returns a named vector

Note

The argument to ordertrans() is technically an order vector because it answers the question "where did the first-named competitor come?" (see the discussion at rrank). But it is not a helpful order vector because you have to go searching through the names—which can appear in any order—for the competitor you are interested in. I guess "generalised order vector" might be a better description of the argument.

Author(s)

Robin K. S. Hankin

See Also

rrank

Examples

```
x <- c(e=4L,a=7L,c=6L,b=1L,f=2L,g=3L,h=5L,i=8L,d=9L)
x
ordertrans(x,letters[1:9])

o <- unclass(skating_table)[,1]
names(o) <- rownames(skating_table)
o
ordertrans(o)

ordertrans(sample(icons_maxp),icons)


rL <- volvo_maxp   # rL is "ranks Likelihood"
rL[] <- rank(-volvo_maxp)

r1 <- as.numeric(unclass(volvo_table)[,1])  # ranks race 1
names(r1) <- rownames(volvo_table)
ordertransplot(rL,r1,xlab="likelihood rank, all races",ylab="rank, race 1")
```

ordervec2supp3 *Various functionality for races and hyper3 likelihood functions*

Description

Various functions for calculating the likelihood function for order statistics in the context of hyper3 likelihood functions. Compare `ggol()` for `hyper2` objects. Used in the `constructor()` suite of analysis.

Usage

```
num3(v,helped=NULL,lambda=1)
den3(v,helped=NULL,lambda=1)
char2nv(x)
ordervec2supp3(v,nonfinishers=NULL)
ordervec2supp3a(v,nonfinishers=NULL,helped=NULL,lambda=1)
rankvec_likelihood3(v,nonfinishers=NULL)
ordertable2supp3(a)
cheering3(v,e,help,nonfinishers=NULL)
args2ordervec(...)
```

Arguments

`v`	Ranks in the form of a character vector. Element `v[1]` is the first-placed competitor, element `v[2]` the second, and so on. For example, `ordervec2supp3(c('b','b','a','c','a'))`
`nonfinishers`	Character vector (a set) showing players that did not finish. See details section and examples
`a`	An ordertable
`helped`	vector of entities being helped
`e`, `help`, `lambda`	Parameters controlling non-independence with `e` a named integer vector specifying equivalence classes of the competitors: names correspond to the competitors, values to their equivalence class, and `help` a numeric vector with entries corresponding to the equivalence classes of `e` and values the strength of the support
`x`	A character vector of competitors
`...`	Arguments passed to `args2ordervec()`

Details

Function `args2ordervec()` takes arguments with names corresponding to players, and entries corresponding to performances (e.g. distances thrown by a javelin, or times for completing a race). It returns a character vector indicating the rank statistic. See examples, and also the javelin vignette.

Function `ordervec2supp3()` takes character vector showing the order of finishing [i.e. a rank statistic], and returns a generalized Plackett-Luce support function in the form of a `hyper3` object. It can take the output of `args2ordervec()` or `rrace3()`. For example:

`ordervec2supp3(c("a","b"),nonfinishers=c("a","b"))`

corresponds to a race between two twins of strength a and two twins of strength b, with only one of each pair finishing; a comes first and b comes second; symbolically

$$a \succ b \succ \{a,b\} \longrightarrow \mathcal{L}(a,b\,|a+b=1\,) = \frac{a}{2a+2b}\cdot\frac{b}{a+2b}$$

Further,

`ordervec2supp3(c("a","b"),c("a","b","c"))`

corresponds to adding a singleton competitor of strength c who did not finish:

$$a \succ b \succ \{a,b,c\} \longrightarrow \mathcal{L}(a,b,c\,|a+b+c=1\,) = \frac{a}{2a+2b+c}\cdot\frac{b}{a+2b+c}$$

(observe that this likelihood function is informative about c). See the examples section below. Experimental function `ordervec2supp3a()` is a generalized version of `ordervec2supp3()` that allows for cheering effects.

Functions `num3()` and `den3()` are low-level helper functions that calculate the numerator and denominator for Plackett-Luce likelihood functions with clones; used in `ordervec2supp3()` and `ordervec2supp3a()`.

Function `ordertable2supp3()` takes an order table (the canonical example is the constructors' formula 1 grand prix results, see `constructor.Rd` and returns a generalized Plackett-Luce support function in the form of a `hyper3` object.

Function `char2nv()` takes a character vector and returns a named vector with entries corresponding to their names' counts. It is used in the extraction and replacement methods for `hyper3` objects.

Function `cheering3()` is a generalization of `ordervec2supp3()`. Competitors who are not mentioned in argument e are assumed to be in an equivalence class of size 1, that is, they are not supported (or indeed suppressed) by anyone else: they are singletons in the terminology of Hankin (2006). Extensive discussions are presented at `inst/plackett_luce_monster.Rmd` and `inst/eurovision.Rmd`.

File `inst/javelin.Rmd` and `inst/race3.Rmd` show some use-cases for these functions.

Note

Function `ordervec2supp3()` is mis-named [it takes a *rank* vector, not an *order* vector]; it will be renamed `rankvec_likelihood3()`, eventually.

Author(s)

Robin K. S. Hankin

See Also

`ordertable2supp`,`ordertrans`

Examples

```
ordervec2supp3(c("a","a","b","c","a","b","c"))
ordervec2supp3(rrace3())
ordervec2supp3(c("a","b"),nonfinishers=c("a","b"))  # a > b >> {a,b}



(o <- args2ordervec(a=c(1,6,9), b=c(2,3,4), c=c(1.1,11.1)))
H <- ordervec2supp3(o)
H
# equalp.test(H)   # takes too long for here


## Race: six competitors a-f finishing in alphabetical order.  Mutually
## supporting groups: (acd), (bf), (e).  Competitor "e" is not
## suppported by anyone else (he is a singleton) so does not need to be
## mentioned in argument 'e' and there are only two helpfulnesses to be
## considered: that of (acd) and that of (bf), which we will take to be
## 1.88 and 1.1111 respectively:

cheering3(v=letters[1:6],e=c(a=1,c=1,b=2,d=1,e=2),help=c(1.88,1.1111))




## Another race: four competitors, including two clones of "a", and two
## singletons "b" and "c".  Here "a" helps his clone at 1.88; and "b"
## and "c" help one another at 1.111:

cheering3(v=c("a","b","a","c"),e=c(a=1,b=2,c=2),help=c(1.8,1.111))


## Same race as above but this time there are two clones of "b", one of
## whom did not finish:

cheering3(v=c("a","b","a","c"),e=c(a=1,b=2,c=2),help=c(1.8,1.111),"b")


## Most common case would be that the clones help each other but noone
## else:

cheering3(v=c("a","b","a","c"),e=c(a=1,b=2,c=3),help=c(1.8,1.111,1),"b")
```

---

pairwise *Pairwise comparisons*

Description

Function pairwise() takes a matrix of pairwise comparisons and returns a hyper2 likelihood function. Function zermelo() implements a standard iterative procedure for maximization of pairwise Bradley-Terry likelihoods (such as those produced by function pairwise()).

Function home_away() takes two matrices, one for home wins and one for away wins. It returns a hyper2 support function that includes a home advantage ghost. Function home_away3() is the same, but returns a hyper3 object. A complex matrix is interpreted as real parts being the home wins and imaginary parts away wins.

Function home_away_table() takes a dataframe of results (each row being a single match) and returns a table amenable to analysis by home_away() or home_away3(). If give takes its default value of FALSE, draws are discarded and a complex matrix of wins and losses is returned. Files inst/monster_vs_lambda.Rmd and inst/home_advantage.Rmd show some use-cases. Argument teams is a character vector that specifies the teams to be tabulated (useful if one wishes to change the default ordering of the teams).

Function white_draw3() returns a hyper3 likelihood function for pairwise comparisons, one of whom has a home team-type advantage (white player in the case of chess). It is designed to work with an array of dimensions $n \times n \times 3$, where n is the number of players. It is used in inst/kka.Rmd to create chess3 likelihood function.

Usage

```
pairwise(M)
zermelo(M, maxit = 100, start, tol = 1e-10, give = FALSE)
home_away(home_games_won, away_games_won)
home_away3(home_games_won, away_games_won,lambda)
home_away_table(a, give=FALSE, teams)
white_draw3(A,lambda,D)
```

Arguments

M	Matrix of pairwise comparison results
maxit	Maximum number of iterations
start	Starting value for iteration; if missing, use equalp()
tol	Numerical tolerance for stopping criterion
give	In zermelo(), Boolean with default FALSE meaning to return the evaluate and TRUE meaning to return all iterations; in home_away_table() governs output form
home_games_won, away_games_won	Matrices showing home games won and away games won
lambda	The home ground advantage (or white advantage in chess)
D	Weight of draw
A	Array of dimension n*n*3, with A[,,i] corresponding to white wins, white draws, and white losses for i=1,2,3. The canonical example would be kka_array, see inst/kka.Rmd for details

a, teams — In function home_away_table(), argument a is a data frame (typically of football results), give a boolean governing output form, and teams a list of football teams. See details

Details

In function zermelo(), the diagonal is disregarded.

If home_games_won is complex, then the real parts of the entries are interpreted as home games won, and the imaginary parts as away games won.

Note

An extended discussion of pairwise() is given in inst/zermelo.Rmd and also inst/karate.Rmd. Functions home_away() and home_away3() are described and used in inst/home_advantage.Rmd; see Davidson and Beaver 1977.

Experimental function pair3() is now removed as dirichlet3() is more general and has nicer idiom; pair3(a=4, b=3, lambda=1.88) and dirichlet3(c(a=4, b=3), 1.88) give identical output.

Author(s)

Robin K. S. Hankin

References

- D. R. Hunter 2004. "MM algorithms for generalized Bradley-Terry models". *The Annals of Statistics*, volume 32, number 1, pages 384–406
- S. Borozki and others 2016. "An application of incomplete pairwise comparison matrices for ranking top tennis players". arXiv:1611.00538v1 10.1016/j.ejor.2015.06.069
- R. R. Davidson and R. J. Beaver 1977. "On extending the Bradley-Terry model to incorporate within-pair order effects". *Biometrics*, 33:693–702

See Also

maxp

Examples

```
 #Data is the top 5 players from Borozki's table 1

M <- matrix(c(
0,10,0, 2,5,
4, 0,0, 6,6,
0, 0,0,15,0,
0, 8,0, 0,7,
1 ,0,3, 0,0
),5,5,byrow=TRUE)
players <-  c("Agassi","Becker","Borg","Connors","Courier")
dimnames(M) <- list(winner=players,loser=players)
```

```
M
# e.g. Agassi beats Becker 10 times and loses 4 times
pairwise(M)
zermelo(M)
# maxp(pairwise(M))  # should be identical (takes ~10s to run)

M2 <- matrix(c(NA,19+2i,17,11+2i,16+5i,NA,12+4i,12+6i,12+2i,19+10i,
NA,12+4i,11+2i,16+2i,11+7i,NA),4,4)
teams <- LETTERS[1:4]
dimnames(M2) <- list("@home" = teams,"@away"=teams)
home_away(M2)
# home_away3(M2,lambda=1.2)  # works but takes too long (~3s)
home_away3(M2[1:3,1:3],lambda=1.2)

M <- kka_array[,,1] + 1i*kka_array[,,3] # ignore draws
home_away(M)
# home_away3(M,lambda=1.3)  # works but takes too long (~3s)

white_draw3(kka_array,1.88,1.11)
```

`pentathlon` *Pentathlon*

Description

Results from the Men's pentathlon at the 2004 Summer Olympics

Usage

```
data(pentathlon)
```

Format

A `hyper2` object that gives a likelihood function

Details

Object `pentathlon` is a `hyper2` object that gives a likelihood function for the strengths of the top seven competitors at the Modern Men's Pentathlon, 2004 Summer Olympics.

Object `pentathlon_table` is an order table: a data frame with rows being competitors, columns being disciplines, and entries being places. Thus looking at the first row, first column we see that Moiseev placed fifth at shooting.

These objects can be generated by running script `inst/pentathlon.Rmd`, which includes some further discussion and technical documentation and creates file `pentathlon.rda` which resides in the `data/` directory.

Note

Many of the competitors' names have diacritics, which I have removed.

References

"Wikipedia contributors", *Modern pentathlon at the 2004 Summer Olympics - Men's*. Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Modern_pentathlon_at_the_2004_Summer_Olympics_%E2%80%93_Men%27s&oldid=833081611, [Online; accessed 5-March-2020]

See Also

ordertable

Examples

```
data(pentathlon)
pie(pentathlon_maxp)
```

---

powerboat *Powerboat dataset*

Description

Race results from the 2018 F1 Powerboat World Championship

Usage

```
data(powerboat)
```

Details

Object `powerboat_table` is a dataframe of results showing ranks of 21 drivers in the 2018 F1 Powerboat World Championship. The format is standard, that is, can be interpreted by function `ordertable2supp()` and indeed `ordertable2supp(powerboat_table[,1:7])` gives the corresponding support function, `powerboat`.

File `inst/powerboat.txt` is the source text file; to create `powerboat_table` use

`read.table(system.file("powerboat.txt",package="hyper2"))`

The dataset used here corrects an apparent typo in the wikipedia table (see github issue 37).

These objects can be generated by running script `inst/powerboat.Rmd`, which includes some further discussion and technical documentation and creates file `powerboat.rda` which resides in the `data/` directory.

Note

Many drivers have names with diacritics, which have been removed from the dataframe.

References

Wikipedia contributors. (2019, October 9). 2018 F1 Powerboat World Championship. In *Wikipedia, The Free Encyclopedia*. Retrieved 00:45, February 21, 2020, from https://en.wikipedia.org/w/index.php?title=2018_F1_Powerboat_World_Championship&oldid=920386507

See Also

ordertable2supp

Examples

```
pie(powerboat_maxp)
```

Print *Print methods*

Description

Print methods for hyper2 and hyper3 objects

Usage

```
## S3 method for class 'hyper2'
print(x, ...)
## S3 method for class 'hyper3'
print(x, ...)
```

Arguments

x	An object of class hyper2 or hyper3
...	Further arguments, currently ignored

Details

Used mainly for their side-effect of printing the log-likelihood function. In the print method, a natural logarithm is indicated with "log()"—not "ln()"—consistent with R builtin terminology base::log().

The hyper2 print method is sensitive to option give_warning_on_nonzero_power_sum. If TRUE, a warning is issued if the powers have nonzero sum. This is usually what you want because observations are typically multinomial; a warning indicates nonzero sum of powers, which should prompt us to check the coding. Vignette zeropower gives a discussion of this issue.

Value

Returns the hyper2 or hyper3 object it was sent, invisibly.

Function pnv() ("**p**rint **n**amed **v**ector") takes a named vector and returns a character string that is used in the hyper3 print method. It is sensitive to base R print options such as digits and scipen. Currently there is no space around the "=" symbol but this is easy to change.

Note

Sometimes the use of `pnv()` can be confusing, as distinct brackets can appear to be identical, as per the example. See how the two terms with power -1 appear to be identical but actually differ by `1e-12`, invisible to the print method which only shows seven significant figures.

Author(s)

Robin K. S. Hankin

Examples

```
data(chess)
chess

getOption("digits")
hyper3(list(c(a=1), c(a=1, b=pi),c(a=1, b=pi+1e-12)), powers = c(2, -1, -1))
```

---

profile *Profile likelihood and support*

Description

Given a support function, return a profile likelihood curve

Usage

```
profsupp(H, i, p, relative=TRUE, ...)
profile_support_single(H, i, p, evaluate=FALSE, ...)
```

Arguments

`H`	hyper2 object
`i`	Name of player for which profile support is to be calculated
`p`	Strength of element `i`
`evaluate`	Boolean, with default `FALSE` meaning to return the maximal support for `p_i=p` and `TRUE` meaning to return the evaluate
`relative`	Boolean; if `TRUE` (default), return the support relative to the maximum support attained; if false, return the support as returned by `profile_support_single()`.
`...`	Arguments passed to `maxp()`

Value

Returns the support at a particular value of p_i, or the evaluate conditional on p_i.

Author(s)

Robin K. S. Hankin

See Also

loglik

Examples

```
## Not run:   # takes too long
p <- seq(from=0.5,to=0.4,len=10)
u <- profsupp(icons,"NB",p)
plot(p,u-max(u))
abline(h=c(0,-2))

## End(Not run)
```

---

psubs *Substitute players of a hyper2 object*

Description

Given a hyper2 object, substitute some players

Usage

```
psubs(H, from, to)
psubs_single(H, from, to)
```

Arguments

H	hyper2 object
from, to	Character vector of players to substitute and their substitutes

Details

Function psubs() substitutes one or more player names, replacing player from[i] with to[i]. If argument to is missing, all players are substituted, the second argument taken to be the replacement: interpret psubs(H,vec) as psubs(H,from=pnames(H),to=vec).

Compare pnames<-(), which can only add players, or reorder existing players.

Function psubs_single() is a low-level helper function that takes a single player and its substitute; it is not intended for direct use.

Value

Returns a hyper2 object

Author(s)

Robin K. S. Hankin

Examples

```
psubs(icons,c("L","NB"),c("London","Norfolk Broads"))

rhyper2() |> psubs(letters,LETTERS)   # ignore i,j,k,...,z

psubs(icons,tolower(pnames(icons)))
```

---

pwa *Player with advantage*

Description

Commonly, when considering competitive situations we suspect that one player has an advantage of some type which we would like to quantify in terms of an additional strength. Examples might include racing at pole position, playing white in chess, or playing soccer at one's home ground. Function `pwa()` ("player with advantage") returns a modified `hyper2` object with the additional strength represented as a reified entity.

Usage

```
pwa(H, pwa, chameleon = "S")
```

Arguments

`H`	A hyper2 object
`pwa`	A list of the players with the supposed advantage; may be character in the case of a named hyper2 object, or an integer vector
`chameleon`	String representing the advantage

Details

Given an object of class `hyper2` and a competitor `a`, we replace every occurrence of `a` with `a+S`, with `S` representing the extra strength conferred.

However, the function also takes a vector of competitors. If there is more than one competitor, the resulting likelihood function does not seem to instantiate any simple situation.

Nice examples of `pwa()` are given in '`inst/cook.Rmd`' and '`inst/universities.Rmd`'.

Value

Returns an object of class `hyper2`.

Note

Earlier versions of this package gave a contrived sequence of observations, presented as an example of `pwa()` with multiple advantaged competitors. I removed it because the logic was flawed, but it featured a chameleon who could impersonate (and indeed eat) certain competitors, which is why the third argument is so named.

The aliases commemorate some uses of the function in the vignettes and markdown files in the '`inst/`' directory.

Author(s)

Robin K. S. Hankin

See Also

`ordervec2supp`

Examples

```
summary(formula1 |> pwa("Hamilton","pole"))

H <- ordervec2supp(c(a = 2, b = 3, c = 1, d = 5, e = 4))
pwa(H,'a')

## Four races between a,b,c,d:
H1 <- ordervec2supp(c(a = 1, b = 3, c = 4, d = 2))
H2 <- ordervec2supp(c(a = 0, b = 1, c = 3, d = 2))
H3 <- ordervec2supp(c(a = 4, b = 2, c = 1, d = 3))
H4 <- ordervec2supp(c(a = 3, b = 4, c = 1, d = 2))

## Now it is revealed that a,b,c had some advantage in races 1,2,3
## respectively.  Is there evidence that this advantage exists?

## Not run:   # takes ~10 seconds, too long for here
specificp.test(pwa(H1,'a') + pwa(H2,'b') + pwa(H3,'c') + H4,"S")

## End(Not run)
```

`ranktable` *Convert rank tables to and from order tables*

Description

Convert rank tables (as generated by `rrank()`, for example) to order tables like the formula 1 tables; and convert back. Print and summary methods for rank tables are documented here. See also `ordertable.Rd`.

Usage

```
ranktable_to_ordertable(xrank)
ordertable_to_ranktable(xorder)
wikitable_to_ranktable(wikitable, strict=FALSE)
## S3 method for class 'ranktable'
summary(object, ...)
## S3 method for class 'ranktablesummary'
print(x,...)
```

Arguments

x, xrank, object	A rank table, an object with class ranktable, for example the value of rrank()
xorder, wikitable	Order tables. Argument wikitable refers to a generalized order table which can include entries such as DNF signifying did not finish.
strict	Controls for wikitable_to_ranktable()
...	Further arguments (currently ignored)

Details

Function ranktable_to_ordertable() is trivial; ordertable_to_ranktable() less so. The prototype for order tables would be skating_table.

Function ordertable_to_ranktable(x) checks for each column being a permutation of seq_len(nrow(x)) and, if not, it stops. In particular, DNF entries are out of scope. To convert order tables such as F1_table_2017, which include DNF entries, use wikitable_to_ranktable() or ordertable2supp() to produce a likelihood function.

Function ranktable_to_printable_object() is a helper function that coerces a ranktable object to a matrix that prints nicely.

File inst/ordertable_to_ranktable.Rmd discusses the ranktable print method and also sets out a common gotcha for interpretation of the internal structure of ranktable objects.

Value

An order table or rank table

Author(s)

Robin K. S. Hankin

See Also

rrank, ordertable2supp

Examples

```
p <- (5:1)/15
names(p) <- letters[1:5]
xrank <- rrank(12,p,rnames=month.abb)
```

RCLF *Dataset from four Scottish football clubs*

Description

These objects refer to results from football matches among four Scottish clubs: Rangers, Celtic, Livinston, Falkirk. A detailed analysis is presented in `inst/home_advantage.Rmd`, which creates the objects documented here from scratch.

1. Object `RCLF3_table` is a table of results for the four clubs: home wins, draws, and away wins. `RCLF3_lambda_max` for the maximum likelihood estimate for λ.
2. Object `RCLF3` is a `hyper3` likelihood function for the table of won games only [that is, excluding draws], using `RCLF3_lambda_max` for the value of λ.
3. Object `RCLF3_maxp` is its evaluate at the MLE for lambda

Usage

```
data(RCLF)
```

Details

The objects are created by `inst/home_advantage.Rmd`.

Author(s)

Robin K. S. Hankin

Source

Data obtained from `www.worldfootball.net`

Examples

```
data(RCLF)
RCLF3_table
RCLF3_maxp
RCLF3_lambda_max

loglik(RCLF3_maxp,RCLF3)
equalp.test(RCLF3)
```

rhyper2 *Random* hyper2 *objects*

Description

Random hyper2 loglikelihood functions, intended as quick "get you going" examples

Usage

```
rhyper2(n = 8, s = 5, pairs = TRUE, teams = TRUE, race = TRUE, pnames)
```

Arguments

n	Number of competitors, treated as even
s	Integer, Measure of the complexity of the log likelihood function
pairs, teams, race	Boolean, indicating whether or not to include different observations
pnames	Character vector of names, if missing interpret as letters; set to NA meaning no names

Note

Function rhyper2() returns a likelihood function based on random observations. To return a random probability vector drawn from a from a given (normalized) likelihood function, use rp().

Author(s)

Robin K. S. Hankin

See Also

rp,rrace

Examples

```
rhyper2()
rp(2,icons)
```

rhyper3 *Random hyper3 objects*

Description

Various random hyper3 objects, in the context of the race metaphor. They return "get you going" examples of hyper3 objects. The defaults are simple but non-trivial and have straightforward interpretations.

The defaults are

```
pn: c(a=2,   b=4,   c=2,   d=1  )  # numbers (two "a"s, four "b"s etc)
ps: c(a=0.3, b=0.1, c=0.2, d=0.4)  # strengths
```

Usage

```
rwinner3(pn = c(a=2, b=4, c=2, d=1), ps = c(a=0.3, b=0.1, c=0.2, d=0.4))
rpair3(n=5, s=3, lambda=1.3)
rrace3(pn = c(a=2, b=4, c=2, d=1), ps=c(a=0.3, b=0.1, c=0.2, d=0.4))
rracehyper3(n=4, size=9, ps=NULL, races=3)
rhyper3(n=5, s=4, type='race', ...)
```

Arguments

pn	A named integer vector showing numbers of each type of player
ps	A named vector showing strengths of each type of player
n, size, races, s, type	Arguments specifying the complexity of the random hyper3 object returned. See details
lambda	Parameter
...	Further arguments passed to rracehyper3() or rpair3()

Details

These functions return hyper3 objects, as indicated by the 3 in their names.

- Function rwinner3() is a low-level helper function that takes a player number argument pn, and a player strength argument ps. It performs an *in silico* race, and returns the (name of) the winner, chosen randomly from a field of runners with appropriate strengths. It is used repeatedly by rrace3() to select a winner from the diminishing pool of still-running players.
- Function rpair3() returns a hyper3 object corresponding to repeated pairwise comparisons including a white-player advantage represented by lambda.
- Function rrace3() returns a rank statistic corresponding to finishing order for a Plackett-Luce race. The output can be passed to ordervec2supp3().

- Function rracehyper3() returns a more complicated hyper3 object corresponding to repeated races.
- Function rhyper3() returns an even more complicated hyper3 object corresponding to repeated races and pairwise comparisons.

Argument n generally specifies the number of distinct types of players. Files inst/mann_whitney_wilcoxon.Rmd and inst/javelin.Rmd show some use-cases for these functions.

Note

In function rracehyper3() [and by extension rhyper3()], if argument n exceeds 26 and argument pn takes its default value of NULL, then an error will be returned because there are only 26 players, one for each letter a-z.

Author(s)

Robin K. S. Hankin

See Also

rrank,ordertable2supp,ordertrans

Examples

```
rracehyper3()
rrace3()
rwinner3()
rhyper3()
rpair3()
ordervec2supp3(rrace3())

table(replicate(100,which(rrace3(pn=c(a=1,b=10),ps=c(a=0.9,b=0.1))=='a')))
```

---

rowing *Rowing dataset, sculling*

Description

Data from Men's single sculls, 2016 Summer Olympics

Usage

```
data(rowing)
```

Format

Object rowing is a hyper2 object that gives a likelihood function for the 2016 men's sculls.

Details

Object rowing is created by the code in inst/rowing.Rmd. This reads file inst/rowing.txt, each line of which is a heat showing the finishing order. Object rowing_table is the corresponding R list.

File inst/rowing_minimal.txt has the same data but with dominated players (that is, any group of players none of whom have beaten any player not in the group) have been removed. This is because dominated players have a ML strength of zero.

References

Wikipedia contributors, "Rowing at the 2016 Summer Olympics—Men's single sculls", *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Rowing_at_the_2016_Summer_Olympics_%E2%80%93_Men%27s_single_sculls&oldid=753517240 (accessed December 7, 2016).

See Also

ggrl

Examples

```
dotchart(rowing_maxp)
```

---

rp *Random samples from the prior of a* hyper2 *object*

Description

Uses Metropolis-Hastings to return random samples from the prior of a hyper2 object

Usage

```
rp(n, H, startp = NULL, fcm = NULL, fcv = NULL, SMALL = 1e-06, l=loglik, fillup=TRUE, ...)
```

Arguments

H	Object of class hyper2
n	Number of samples
startp	Starting value for the Markov chain, with default NULL being interpreted as starting from the evaluate
fcm, fcv	Constraints as for maxp()
SMALL	Notional small value for numerical stability
l	Log-likelihood function with default loglik()
fillup	Boolean, with default TRUE meaning to return a matrix with the fillup value added, and column names matching the pnames() of argument H
...	Further arguments, currently ignored

Details

Uses the implementation of Metropolis-Hastings from the MCE package to sample from the posterior PDF of a hyper2 object.

If the distribution is Dirichlet, use rdirichlet() to generate random observations: it is much faster, and produces serially independent samples. To return *uniform* samples, use rp_unif() (documented at dirichlet.Rd).

Value

Returns a matrix, each row being a unit-sum observation.

Note

Function rp() a random sample from a given normalized likelihood function. To return a random likelihood function, use rhyper2().

File inst/ternaryplot_hyper2.Rmd shows how to use Ternary::ternaryPlot() with rp().

Author(s)

Robin K. S. Hankin

See Also

maxp,loglik,dirichlet,rhyper2

Examples

```
rp(10,icons)

plot(loglik(rp(30,icons),icons),type='b')
```

---

rrace *A random race with given BT strengths*

Description

Returns a rank vector suitable for interpretation with race().

Usage

```
rrace(strengths)
```

Arguments

strengths	Named vector with names being players and values being their Bradley-Terry strengths

Details

Uses a simple recursive system to generate the ranks.

Value

Returns a character vector with entries corresponding to the competitor. The first element is the winner, the second the runner-up, and so on, until the final element is the last to cross the finishing line.

Author(s)

Robin K. S. Hankin

See Also

rrace3,hyper2

Examples

```
o <- c(a=0.4, b=0.3, c=0.2, d=0.1)
rrace(o)

suppfun(rrace(o))

as.ranktable(t(replicate(10, rrace(o)))) # same as rrank(10,o)
```

---

rrank *Random ranks*

Description

A function for producing ranks randomly, consistent with a specified strength vector

Usage

```
rrank(n = 7, p = (4:1)/10, pnames=NULL, fill = FALSE, rnames=NULL)
rrankk(n = 37, p = (20:1)/210, pnames=NULL, fill=FALSE, rnames=NULL)
## S3 method for class 'ranktable'
print(x, ...)
rrank_single(p)
rorder_single(p)
```

Arguments

n	Number of observations
p	Strength vector
pnames	Character vector ("player names") specifying names of the columns
rnames	Character vector ("row names" or "race names") specifying names of the rows
fill	Boolean, with default FALSE meaning to interpret the elements of p as strengths, notionally summing to one; and TRUE meaning to augment p with a fillup value
x, ...	Arguments passed to the print method

Value

Random rank observations. Function rrank(), with no arguments, returns a small get-you-going example of a ranktable object. Function rrankk() returns a bigger and more complicated object.

If n=1, rrank() returns a vector; if n>1 it returns a matrix with n rows, each corresponding to a ranking. The canonical example is a race in which the probability of competitor i coming first is $p_i / \sum p_j$, where the summation is over the competitors who have not already finished.

If, say, the first row of rrank() is c(e, d, b, a, c), then competitor e came first, competitor d came second, competitor b came third, and so on.

Note that function rrank() returns an object of class ranktable. The column names appear as "c1, c2,..." which is intended to be read "came first", "came second", and so on. The difference between *rank* and *order* can be confusing.

```
> x <- c(a=3.01, b=1.04, c=1.99, d=4.1)
> x
   a    b    c    d
3.01 1.04 1.99 4.10
> rank(x)
a b c d
3 1 2 4
> order(x)
[1] 2 3 1 4
```

In the above, rank() shows us that element a of x (viz 3.01) is the third largest, element b (viz 1.04) is the smallest, and so on; order(x) shows us that the smallest element x is x[2], the next smallest is x[3], and so on. Thus x[order(x)] == sort(x), and rank(x)[order(x)] == seq_along(x). In the current context we want ranks not orders; we want to know who came first, who came second, and so on:

```
R> rrank(2,(4:1)/10)
     c1 c2 c3 c4
[1,]  2  3  1  4
[2,]  1  3  2  4
R>
```

In the above, each row is a race; we have four runners and two races. In the first race (the top row), runner number 2 came first, runner 3 came second, runner 1 came third, and so on. In the second race (bottom row), runner 1 came first, etc. Taking the first race as an example:

Rank: who came first? runner 2. Who came second? runner 3. Who came third? runner 1. Who came fourth? runner 4. Recall that the Placket-Luce likelihood for a race in which the rank statistic was 2314 (the first race) would be $\frac{p_2}{p_2+p_3+p_1+p_4} \cdot \frac{p_3}{p_3+p_1+p_4} \cdot \frac{p_1}{p_1+p_4} \cdot \frac{p_4}{p_4}$.

Order: where did runner 1 come? third. Where did runner 2 come? first. Where did runner 3 come? second. Where did runner 4 come? fourth. Thus the order statistic would be 3124.

Vignette "skating_analysis" gives another discussion.

Note that function rrank() returns an object of class "rrank", which has its own print method. This can be confusing. Further details are given at ranktable.Rd.

Function rrank_single() is a low-level helper function:

```
> p <- c(0.02,0.02,0.9,0.02,0.02,0.02)  # competitor 3 the strongest
> rrank_single(p)
[1] 3 2 4 6 4 1
```

Above, we see from p that competitor 3 is the strongest, coming first with 92% probability. And indeed the resulting rank statistic given by rorder_single() shows competitor 3 coming first, 2 coming second, and so on. Compare rorder_single():

```
> rorder_single(p)
[1] 6 3 1 4 5 2
>
```

Above we see see from rrank_single(p) that competitor 1 came sixth, competitor 2 came third, and competitor 3 came first (as you might expect, as competitor 3 is the strongest). Note that the R idiom for rorder_single() is the same as that used in the **permutations** package for inverting a permutation: o[o] <-seq_along(o).

Note

Similar functionality is given by rrace(), documented at rhyper3.

Author(s)

Robin K. S. Hankin

See Also

ordertrans, suppfun, skating, rhyper3

Examples

```
rrank_single(zipf(9))

ptrue <- (4:1)/10
names(ptrue) <- letters[1:4]
rrank(10,p=ptrue)

H <- suppfun(rrank(10,p=ptrue))

## Following code commented out because they take too long:

# mH <- maxp(H)   # should be close to ptrue
# H <- H + rank_likelihood(rrank(30,mH)) # run some more races
# maxp(H)  # revised estimate with additional data
```

---

skating *Figure skating at the 2002 Winter Olympics*

Description

A likelihood function for the competitors at the Ladies' Free Skate at the 2002 Winter Olympics

Usage

```
skating
```

Details

Three objects `skating`, a log-likelihood function for the competitors' strengths, `skating_table`, an order table for each of the 9 judges, and `skating_maxp`, the result of `maxp(skating)`, which is included to save time in the examples.

These objects can be generated by running script `inst/skating.Rmd`, which includes some further discussion and technical documentation. The dataset is interesting because it has been analysed by many workers, including Lock and Lock, for consistency between the judges.

Note that file is structured so that each competitor is a row, and each judge is a column. Function `suppfun()` requires a transpose of this to operate.

Object `skating_table` is an order table, taken from Lock and Lock. It corrects what appears to be an error in which judge 5 ranked both Butyrskaya and Kettunen 12; there is no 13. Using EM, I reckon that Butyrskaya should be ranked twelfth and Kettunen thirteenth.

Note

There is an (`Rbuildignore`-d) discussion of a `skeleton` dataset in the `inst/` directory of the repo, it's easy to confuse this with `skating`.

Author(s)

Robin K. S. Hankin

References

- https://en.wikipedia.org/wiki/Figure_skating_at_the_2002_Winter_Olympics#Full_results_2
- Robin Lock and Kari Frazer Lock, Winter 2003. "Judging Figure Skating Judges". *STATS 36*, ASA

Examples

```
data(skating)
dotchart(skating_maxp)

ordertable_to_ranktable(skating_table)

rL <- sort(skating_maxp,decreasing=TRUE)
rL[] <- seq_along(rL)
rO <- seq_len(nrow(skating_table))
names(rO) <- rownames(skating_table)
ordertransplot(rO,rL,
   xlab="official rank",ylab="likelihood rank",
   main="Ladies free skating, 2002 Winter Olympics")
```

---

soling *Sailing at the 2000 Summer Olympics - soling*

Description

Race results from the 2000 Summer Olympics: soling

Usage

```
data(soling)
```

Format

A hyper2 object that gives a likelihood function

Details

The Soling three person keelboat event at the 2000 Summer Olympic games furnishes a rich dataset. An order table and likelihood function is given in the package as `soling_table` and `soling` respectively. Data from the round robins and the quarter final is given in matrices `soling_rr1`, `soling_rr2`, `soling_qf` respectively.

These objects can be generated by running script `inst/soling.Rmd`, which includes some further discussion and technical documentation, and creates file `soling.rda` which resides in the `data/` directory.

References

Wikipedia contributors, "Sailing at the 2000 Summer Olympics - Soling," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Sailing_at_the_2000_Summer_Olympics_%E2%80%93_Soling&oldid=945362535 (accessed March 23, 2020).

See Also

`ordertable2supp`

Examples

```
data(soling)
ordertable_to_ranktable(soling_table)
pie(soling_maxp)
```

`summary.hyper2` *Summary method for hyper2 objects*

Description

Give a summary of a hyper2 object, and a print method

Usage

```
## S3 method for class 'hyper2'
summary(object, ...)
## S3 method for class 'summary.hyper2'
print(x, ...)
```

Arguments

`object`, `x`	Object of class hyper2
`...`	Further arguments, currently ignored

Details

Mostly self-explanatory, based on the equivalent in the `untb` package.

Author(s)

Robin K. S. Hankin

See Also

hyper2

Examples

```
summary(icons)
```

---

suplist *Methods for suplist objects*

Description

Basic functionality for lists of hyper2 objects, allowing the user to concatenate independent observations which are themselves composite objects such as returned by ggrl().

Usage

```
## S3 method for class 'suplist'
Ops(e1, e2)
## S3 method for class 'suplist'
sum(x,...,na.rm=FALSE)
suplist_add(e1,e2)
suplist_times_scalar(e1,e2)
as.suplist(L)
```

Arguments

e1, e2	Objects of class suplist, here interpreted as a list of possible likelihood functions (who should be added)
x, ..., na.rm	In the sum() method, objects to be summed; na.rm is currently ignored
L	A list of hyper2 objects

Details

A suplist object is a list of hyper2 objects. Each element is a hyper2 object that is consistent with an incomplete rank observation R; the list elements are exclusive and exhaustive for R. If S is a suplist object, and S=list(H1,H2,...,Hn) where the Hi are hyper2 objects, then $\text{Prob}(p|H_1) + \cdots + \text{Prob}(p|H_n)$. This is because the elements of a suplist object are disjoint alternatives.

It is **incorrect** to say that a likelihood function $\mathcal{L}_S(p)$ for p is the sum of separate likelihood functions. This is incorrect because the arbitrary multiplicative constant messes up the math, for example we might have $\mathcal{L}_{H_1}(p) = C_1\text{Prob}(p|H_1)$ and $\mathcal{L}_{H_2}(p) = C_2\text{Prob}(p|H_2)$ and indeed $\mathcal{L}_{H_1\cup H_2}(p) = C_{12}\left(\text{Prob}(p|H_1) + \text{Prob}(p|H_2)\right)$ but

$$\mathcal{L}_{H_1}(p) + \mathcal{L}_{H_2}(p) \neq C_1\text{Prob}(p|H_1) + C_2\text{Prob}(p|H_2)$$

(the right hand side is meaningless).

Functions `suplist_add()` and `sum.suplist()` implement "`S1+S2`" as the support function for independent observations `S1` and `S2`. The idea is that the support functions "add" in the following sense. If `S1=list(H1,...,Hr)` and `S2=list(I1,...,Is)` where `Hx,Ix` are `hyper2` objects, then the likelihood function for "`S1+S2`" is the likelihood function for `S1` followed by (independent) `S2`. Formally

$$\text{Prob}(p|S_1+S_2) = (\text{Prob}(p|H_1)+\cdots+\text{Prob}(p|H_r))\cdot(\text{Prob}(p|I_1)+\cdots+\text{Prob}(p|I_s))$$

$$\log\text{Prob}(p|S_1+S_2) = \log(\text{Prob}(p|H_1)+\cdots+\text{Prob}(p|H_r))+\log(\text{Prob}(p|I_1)+\cdots+\text{Prob}(p|I_s))$$

However, `S1+S2` is typically a large and unwieldy object, and can be very slow to evaluate. These functions are here because they provide slick package idiom.

The experimental `lsl` mechanism furnishes an alternative methodology which is more computationally efficient at the expense of a non-explicit likelihood function. It is not clear at present (2022) which of the two systems is better.

Value

Returns a `suplist` object.

Author(s)

Robin K. S. Hankin

See Also

`Ops.hyper2`,`Extract`,`loglik`

Examples

```
W <- hyper2(pnames=letters[1:5])
W1 <- ggrl(W, 'a', letters[2:3],'d')  # 2-element list
W2 <- ggrl(W, 'e', letters[1:3],'d')  # 6-element list
W3 <- ggrl(W, 'c', letters[4:5],'a')  # 2-element list

# likelihood function for independent observations  W1,W2,W3:

W1+W2+W3 # A 2*6*2=24-element list

like_single_list(equalp(W),W1+W2+W3)
## Not run: dotchart(maxplist(W1+W1+W3),pch=16) # takes a long time

a <- lsl(list(W1,W2,W3),4:6)  # observe W1 four times, W2 five times and W3 six times
loglik_lsl(equalp(W),a,log=TRUE)
```

suppfun *Convert various datasets to support functions.*

Description

Function suppfun() is an S3 generic that returns a support function, dispatching on the class of its primary argument.

Usage

```
## S3 method for class 'ordertable'
suppfun(x, ...)
## S3 method for class 'ranktable'
suppfun(x, times, ...)
## S3 method for class 'character'
suppfun(x, nonfinishers, ...)
suppfun(x, ...)
```

Arguments

x	Dataset to be converted to a support function
times	Vector corresponding to rows of x giving how many times that observation was obtained (cf preflib datasets); recycled if necessary
nonfinishers	Character vector of competitors who did not finish, passed to rankvec_likelihood()
...	Further arguments

Details

Function suppfun() is intended as a consistent S3 generic approach to converting datasets to support functions. I am gradually going to switch out use of ordertable2supp() in favour of suppfun() in the docs.

If given a data frame, suppfun() will not guess whether it is to be interpreted as an ordertable or a ranktable, and return an error.

Value

Generally, return a support function

Author(s)

Robin K. S. Hankin

Examples

```
suppfun(pentathlon_table)
suppfun(rrank(10, p = (7:1)/28))
suppfun(letters)
suppfun(c(c = 2, b = 1, a = 3))
```

---

surfing *Surfing dataset*

Description

Data from the 2019 World Surf League (WSL) tour

Usage

```
data(surfing)
```

Details

The package contains four datasets from WSL 2019:

- surfing, a log likelihood function for the strengths of the competitors
- surfing_maxp, corresponding precalculated evaluate
- surfing_venuetypes, a dataframe showing the beach types at the different venues of the tour

These objects can be generated by running script inst/surfing.Rmd, which includes some further discussion and technical documentation and creates file surfing.rda which resides in the data/ directory.

Author(s)

Robin K. S. Hankin

Examples

```
dotchart(surfing_maxp)
```

sushi *Sushi dataset*

Description

Dataset from the *preflib* website, discussing sushi preferences.

Usage

```
data(sushi)
```

Details

Sushi preference dataset.

- Object `sushi_table` is a database of sushi preferences; each row is a judge and the rows are rank vectors of the different sushi types.
- Object `sushi` is a likelihood function at the estimated value of lambda
- Object `sushi_maxp` is the maximum likelihood estimate for the Bradley-Terry strengths of the sushis, at the estimate for lambda.
- `sushi_eq_classes` is a named vector giving the equivalences classes of sushi. Class 1 is tuna, class 2 is everything else.

These objects can be generated by running script `inst/sushi.Rmd`, which includes some further discussion and technical documentation and creates file `sushi.rda` which resides in the `data/` directory.

Examples

```
sushi
sushi_maxp
```

T20 *Indian Premier League T20 cricket*

Description

Cricket dataset, T20 Indian Premier League 2008-2017

Usage

```
data(T20)
```

Details

Dataframe `T20_table` has one row for each T20 IPL match in the period 2008-2017 with the exception of seven drawn matches and three no-result matches which were removed. Object `T20` is a likelihood function for the strengths of the 13 teams, and `T20_toss` is a likelihood function that also includes a toss strength term.

These objects can be generated by running script `inst/T20.Rmd`, which is based on Chandel and Hankin 2019. This includes some further discussion and technical documentation and creates file `T20.rda` which resides in the `data/` directory.

References

- T. Chandel and R. K. S. Hankin 2019. "Analysing the impact of winning a coin toss in the Indian Premier League". Auckland University of Technology.

Examples

```
summary(T20)
dotchart(T20_maxp)
```

---

table_tennis *Match outcomes from repeated table tennis matches*

Description

Match outcomes from repeated singles table tennis matches

Usage

```
data(table_tennis)
```

Format

A likelihood function corresponding to the match outcomes listed below.

Details

There are four players, A, B, and C, who play singles table tennis matches with the following results:

- A vs B, A serves, 5-1
- A vs B, B serves, 1-3
- A vs C, A serves, 4-1
- A vs C, C serves, 1-2

As discussed in vignette `table_tennis_serve`, we wish to assess the importance of the serve. The vignette presents a number of analyses including a profile likelihood plot.

See vignette `table_tennis_serve` for an account of how to create `table_tennis`.

Examples

```
data(table_tennis)
dotchart(maxp(table_tennis))
```

`tennis` *Match outcomes from repeated doubles tennis matches*

Description

Match outcomes from repeated doubles tennis matches

Usage

```
data(tennis)
```

Format

A hyper2 object corresponding to the match outcomes listed below.

Details

There are four players, p_1 to p_4. These players play doubles tennis matches with the following results:

match	score
$\{p_1, p_2\}$ vs $\{p_3, p_4\}$	9-2
$\{p_1, p_3\}$ vs $\{p_2, p_4\}$	4-4
$\{p_1, p_4\}$ vs $\{p_2, p_3\}$	6-7
$\{p_1\}$ vs $\{p_3\}$	10-14
$\{p_2\}$ vs $\{p_3\}$	12-14
$\{p_1\}$ vs $\{p_4\}$	10-14
$\{p_2\}$ vs $\{p_4\}$	11-10
$\{p_3\}$ vs $\{p_4\}$	13-13

It is suspected that p_1 and p_2 have some form of team cohesion and play better when paired than when either solo or with other players. As the scores show, each player and, apart from p1-p2, each doubles partnership, is of approximately the same strength.

Dataset `tennis` gives the appropriate likelihood function for the players' strengths; and dataset `tennis_ghost` gives the appropriate likelihood function if the extra strength due to team cohesion of $\{p_1, p_2\}$ is represented by a ghost player.

These objects can be generated by running script `inst/tennis.Rmd`, which includes some further discussion and technical documentation and creates file `tennis.rda` which resides in the `data/` directory.

Source

Doubles tennis matches at NOCS, Jan-May 2008

References

Robin K. S. Hankin (2010). "A Generalization of the Dirichlet Distribution", *Journal of Statistical Software*, 33(11), 1-18

Examples

```
summary(tennis)

tennis |> psubs(c("Federer","Laver","Graf","Navratilova"))

## Following line commented out because it takes too long:
# specificp.gt.test(tennis_ghost,"G",0)
```

---

tests *Hypothesis testing*

Description

Tests different nulls against a free alternative

Usage

```
equalp.test(H, startp=NULL, ...)
knownp.test(H, p, ...)
samep.test(H, i, give=FALSE, startp=NULL, ...)
specificp.test(H, i, specificp=1/size(H),
         alternative = c("two.sided","less","greater"),  ...)
specificp.ne.test(H, i, specificp=1/size(H), ...)
specificp.gt.test(H, i, specificp=1/size(H), delta=1e-5, ...)
specificp.lt.test(H, i, specificp=1/size(H), ...)
## S3 method for class 'hyper2test'
print(x, ...)
```

Arguments

H	A likelihood function, an object of class hyper2
p	In knownp.test(), putative strength vector to be tested
...	Further arguments passed by equalp.test() to maxp() and ignored by print.hyper2test()
startp	Starting value for optimization
i	A character vector of names
specificp	Strength, real number between 0 and 1

alternative	a character string specifying the alternative hypothesis, must be one of two.sided (default), greater or less. You can specify just the initial letter (taken from t.test.Rd)
give	Boolean, with TRUE meaning to return more detailed debugging information, and default FALSE meaning to return a more user-friendly object of class equalp.test, which has its own print method
x	Object of class equalp.test, the result of equalp.test()
delta	Small value for numerical stability

Details

Given a hyper2 likelihood function, there are a number of natural questions to ask about the strengths of the players; see Hankin 2010 (JSS) for examples. An extended discussion is presented in vignette "hyper2" and the functions documented here cover most of the tests used in the vignette.

The tests return an object with class hyper2test, which has its own print method.

- Function equalp.test(H,p) tests the null that all strengths are equal to vector p. If p is missing, it tests $H_0: p_1 = p_2 = \cdots = p_n = \frac{1}{n}$, for example equalp.test(icons)
- Function knownp.test() tests the null that the strengths are equal to the elements of named vector p; it is a generalization of equalp.test(). Example: knownp.test(icons,zipf(6)).
- Function specificp.test(H,i,p) tests $H_0: p_i = p$, for example specificp.test(icons,"NB",0.1)
- Function samep.test() tests $H_0: p_{i_1} = p_{i_2} = \cdots = p_{i_k}$, for example samep.test(icons,c("NB","L")) tests that NB has the same strength as L.
- Functions specificp.ne.test(H,i,p), specificp.gt.test(H,i,p), and specificp.lt.test(H,i,p) are low-level helper functions that implement one- or two-sided versions of specificp.test() via the alternative argument, following t.test()

Value

The test functions return a list with class "hyper2test" containing the following components:

statistic	the difference in support between the null and alternative
p.value	the (asymptotic) p-value for the test, based on Wilks's theorem
estimate	the maximum likelihood estimate for p
method	a character string indicating what type of test was performed
data.name	a character string giving the name(s) of the data.

Note

Function specificp.gt.test() includes quite a bit of messing about to ensure that frequently-used idiom like specificp.gt.test(icons,"NB",0) works as expected, testing a null of p_NB=0 (observe that specificp.ne.test(icons,"NB",0) and specificp.gt.test(icons,"NB",0) will (correctly) throw an error). In the case of testing a strength's being zero, the support function is often quite badly-behaved near the constraint [think tossing a coin with probability p twice, observing

one head and one tail, and testing $p = 0$; at the constraint, the likelihood is zero, the support negative infinity, and the gradient of the support is infinite]. Numerically, the code tests `p_NB=delta`. Note that similar machinations are not required in `specificp.lt.test()` because a null of `p_NB=1` is unrealistic.

Function `samep.test()` does not have access to gradient information so it is slow, inaccurate, and may fail completely for high-dimensional datasets. If `any(i==n)`, this constrains the fillup value; this makes no difference mathematically but the function idiom is involved.

In functions `specificp.??.test(H,i,...)`, if `i` is not present in `H`, an error is returned although technically the result should be "not enough evidence to reject", as `H` is uninformative about `i`.

See Also

`maxp`

Examples

```
equalp.test(chess)

# samep.test(icons,c("NB","L"))
# knownp.test(icons,zipf(icons))
```

`tidy` *Tidy up a hyper2 object*

Description

Tidy up a hyper2 object by removing players about which we have no information

Usage

```
tidy(H)
```

Arguments

`H`	A `hyper2` object

Details

Function `tidy(H)` returns a hyper2 object mathematically identical to `H` but with unused players (that is, players that do not appear in any bracket) removed. Players about which `H` is uninformative are removed from the `pnames` attribute.

Note that idiom `pnames(H) <- foo` can also be used to manipulate the `pnames` attribute.

Author(s)

Robin K. S. Hankin

Examples

```
H <- hyper2(pnames=letters)
H["a"] <- 1
H["b"] <- 2
H[c("a","b")] <- -3

pnames(H)
pnames(tidy(H))

H == tidy(H)  # should be TRUE
```

`universities` *New Zealand University ranking data*

Description

Times Higher Education World University Rankings

Usage

```
data(universities)
```

Format

A `hyper2` object that gives a likelihood function for ranking of NZ universities

Details

The data is taken directly from the THE website, specifying "New Zealand":

https://www.timeshighereducation.com/world-university-rankings/2020/world-ranking#!/page/0/length/25/locations/NZ/sort_by/rank/sort_order/asc/cols/stats

Object `universities` is a `hyper2` support function and `universities_table` a data frame.

These objects can be generated by running script `inst/universities.Rmd`, which includes some further discussion and technical documentation, and creates file `universities.rda` which resides in the `data/` directory.

See Also

`ordertable`

Examples

```
summary(universities)

psubs(universities,c("AUT","UoA"),c("University of Auckland","Auckland University of Technology"))

pie(universities_maxp)
```

`volleyball` *Results from the NOCS volleyball league*

Description

Results from the NOCS volleyball league. Object `volleyball_table` is a matrix in which each column corresponds to a player and each row corresponds to a volleyball set; `volleyball` is the corresponding likelihood function in the form of a `hyper2` distribution.

Usage

```
data(volleyball)
```

Details

A volleyball *set* is a Bernoulli trial between two disjoint subsets of the players. The two subsets are denoted (after the game) as the "winners" and the "losers": these are denoted by `1` and `0` respectively.

Thus the first line reads of `volleyball_results` reads:

```
 p1  p2  p3  p4  p5  p6  p7  p8  p9
 1    0  NA   1   0   0  NA   1  NA
```

showing that the teams were `p1`, `p4` and `p8` against `p2`, `p5` and `p6`; players `p3`, `p7` and `p9` did not play.

These datasets illustrate the fact that such Bernoulli trials are only weakly informative.

These objects can be generated by running script `inst/volleyball.Rmd`, which includes some further discussion and technical documentation and creates file `volleyball.rda` which resides in the `data/` directory.

The dataset is used in an example at `zipf.Rd`: the players' strengths are not Zipf.

Author(s)

Robin K. S. Hankin

Source

Volleyball games at NOCS, 2006-2008

References

Robin K. S. Hankin (2010). "A Generalization of the Dirichlet Distribution", *Journal of Statistical Software*, 33(11), 1-18

See Also

zipf

Examples

```
volleyball == volley(volleyball_table)  # should be TRUE
```

---

volvo *Race results from the 2014-2015 Volvo Ocean Race*

Description

Race results from the twelfth edition of the round-the-world Volvo Ocean Race.

Usage

```
data(volvo)
```

Format

A hyper2 object that gives a likelihood function

Details

Object volvo is a hyper2 object that gives a likelihood function for the strengths of the competitors of the 2014-2015 Volvo Ocean Race; volvo_maxp is a precomputed maximum likelihood estimate of the competitors' strengths. Object volvo_table is a data frame with rows being teams and columns being legs.

These objects can be generated by running script inst/volvo.Rmd, which includes some further discussion and technical documentation and creates file volvo.rda which resides in the data/ directory.

References

Wikipedia contributors, 2019. "2014-2015 Volvo Ocean Race". In *Wikipedia, the free encyclopedia*. Retrieved 22:21, February 28, 2020. https://en.wikipedia.org/w/index.php?title=2014%E2%80%932015_Volvo_Ocean_Race&oldid=914916131,

See Also

ordertable2supp

Examples

```
pie(volvo_maxp)
# equalp.test(volvo)   # takes ~10 seconds to run

# convert table to a support function:
suppfun(volvo_table, noscore = c("DNF", "DNS"))
```

---

zapweak *Zap weak competitors*

Description

Given a hyper2 object, discard competitors with a small estimated strength.

Usage

```
zapweak(H, minstrength = 1e-05, maxit, ...)
```

Arguments

H	Object of class hyper2
minstrength	Strength below which to discard competitors
maxit	Maximum number of iterations; if missing, use size(H)-1
...	Further arguments, passed to maxp()

Details

Iteratively discards the weakest player (if the estimated strength is less than minstrength) using discard_flawed(). maxp(..,n=1) for efficiency.

Value

Returns a slimmed-down hyper2 object with weak players removed.

Note

This function is experimental and appears to be overly aggressive. For some likelihood functions zapweak() removes *all* the players.

I now think that there is no consistent way to remove weaker players from a likelihood function. I think the only way to do it is to look at the dataset that generates the likelihood function, somehow weed out the players with the poorest performance, and generate a new likelihood function without them.

Author(s)

Robin K. S. Hankin

See Also

discard_flawed,maxp

Examples

```
 zapweak(icons)        # removes noone
# zapweak(rowing)      # removes everyone...
```

---

zipf *Zipf's law*

Description

A very short function that reproduces Zipf's law: a harmonic rank-probability distribution, formally

$$p(i) = \frac{i^{-1}}{\sum_{i=1}^{N} i^{-1}}, \qquad i = 1, \ldots, N$$

The volleyball dataset might reasonably be assumed to be zipf, but one can reject this hypothesis at 5%, see the examples.

Usage

```
zipf(n)
```

Arguments

n	Integer; if a hyper2 object is supplied this is interpreted as `size(n)`

Value

Returns a numeric vector summing to one

Author(s)

Robin K. S. Hankin

See Also

knownp.test

Examples

```
zipf(icons)
knownp.test(volleyball,zipf(volleyball))
```

Index