

# Package ‘hyper2’

March 4, 2021

**Type** Package

**Title** The Hyperdirichlet Distribution, Mark 2

**Version** 2.0-0

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

**Description**

A suite of routines for the hyperdirichlet distribution; supersedes the 'hyperdirichlet' package.

**License** GPL (>= 2)

**LazyData** yes

**Depends** methods, magrittr, cubature, R (>= 3.5.0)

**Suggests** knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**Imports** Rcpp (>= 0.12.5), partitions, EMC, mathjaxr

**LinkingTo** Rcpp

**URL** <https://github.com/RobinHankin/hyper2>

**BugReports** <https://github.com/RobinHankin/hyper2/issues>

**RoxygenNote** 6.1.1

**RdMacros** mathjaxr

## R topics documented:

hyper2-package . . . . .	3
as.ordertable . . . . .	5
B . . . . .	6
character_to_number . . . . .	8
chess . . . . .	9
counterstrike . . . . .	10
cplusplus . . . . .	11
curling . . . . .	12
dirichlet . . . . .	13
eurovision . . . . .	14
Extract . . . . .	15
fillup . . . . .	16
formula1 . . . . .	17
ggol . . . . .	18

gradient . . . . .	20
handover . . . . .	22
head.hyper2 . . . . .	23
hyper2 . . . . .	23
icons . . . . .	25
increment . . . . .	26
interzonal . . . . .	28
jester . . . . .	28
karate . . . . .	29
karpov_kasparov_anand . . . . .	30
keep . . . . .	31
length.hyper2 . . . . .	32
loglik . . . . .	32
masterchef . . . . .	34
matrix2supp . . . . .	35
maxp . . . . .	36
moto . . . . .	38
mult_grid . . . . .	38
NBA . . . . .	39
Ops.hyper2 . . . . .	40
ordertable . . . . .	41
ordertable2points . . . . .	43
ordertable2supp . . . . .	44
ordertrans . . . . .	46
pentathlon . . . . .	47
powerboat . . . . .	48
Print . . . . .	49
profile . . . . .	50
pwa . . . . .	51
ranktable . . . . .	52
rhyper2 . . . . .	53
rowing . . . . .	54
rp . . . . .	55
rrank . . . . .	56
skating . . . . .	58
soling . . . . .	59
summary.hyper2 . . . . .	60
suplist . . . . .	60
T20 . . . . .	62
table_tennis . . . . .	63
tennis . . . . .	63
tests . . . . .	64
tidy . . . . .	66
universities . . . . .	67
volleyball . . . . .	68
volvo . . . . .	69
zapweak . . . . .	70

hyper2-package

*The Hyperdirichlet Distribution, Mark 2***Description**

A suite of routines for the hyperdirichlet distribution; supersedes the 'hyperdirichlet' package.

**Details**

The DESCRIPTION file:

```

Package:      hyper2
Type:         Package
Title:        The Hyperdirichlet Distribution, Mark 2
Version:      2.0-0
Authors@R:    person(given=c("Robin", "K. S."), family="Hankin", role = c("aut", "cre"), email="hankin.rob@ma
Maintainer:   Robin K. S. Hankin <hankin.rob@ma
Description:  A suite of routines for the hyperdirichlet distribution; supersedes the 'hyperdirichlet' package.
License:      GPL (>= 2)
LazyData:    yes
Depends:      methods, magrittr, cubature, R (>= 3.5.0)
Suggests:    knitr, rmarkdown, testthat
VignetteBuilder: knitr
Imports:      Rcpp (>= 0.12.5), partitions, EMC, mathjaxr
LinkingTo:    Rcpp
URL:          https://github.com/RobinHankin/hyper2
BugReports:   https://github.com/RobinHankin/hyper2/issues
RoxygenNote: 6.1.1
RdMacros:    mathjaxr
Author:       Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

```

Index of help topics:

B	Normalizing constant for the hyperdirichlet distribution
Extract.hyper2	Extract or replace parts of a hyper2 object
NBA	Basketball dataset
Ops.hyper2	Arithmetic Ops Group Methods for hyper2 objects
Print	Print methods
T20	Indian Premier League T20 cricket
as.ordertable	Convert an order table with DNS entries to a nice order table
character_to_number	Convert a character vector to a numeric vector
chess	Chess playing dataset
counterstrike	Counterstrike
cplusplus	Wrappers to c calls
curling	Curling at the Winter Olympics, 1998-2018
dirichlet	Dirichlet distribution and generalizations
equalp.test	Hypothesis testing
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
hyper2	Basic functions in the hyper2 package
hyper2-package	The Hyperdirichlet Distribution, Mark 2
icons	Dataset on climate change due to O'Neill
increment	Increment and decrement operators
interzonal	1963 World Chess Championships
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 functionality
ordertable	Order tables
ordertable2points	Calculate points from an order table
ordertable2supp	Translate order tables to support functions
ordertrans	Order transformation
pentathlon	Pentathlon
powerboat	Powerboat dataset
profile	Profile likelihood and support
pwa	Player with advantage
ranktable	Convert rank tables to and from order tables
rhyper2	Random 'hyper2' objects
rowing	Rowing dataset, sculling
rp	Random samples from the prior of a 'hyper2' object
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
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

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.

**Author(s)**

NA

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, <https://www.jstatsoft.org/v33/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.

**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 coercable 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)
pie(maxp(ordertable2supp(as.ordertable(F1_table_2017[1:9,])))
```

---

 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 $\text{disallowed}(p)$ evaluating to FALSE
e,p	A vector; see details
ip	A vector of probabilities corresponding to $\text{indep}(p)$ 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 <code>adaptIntegrate()</code>
normalize	Boolean, indicates whether return value of <code>mean_hyper2()</code> is normalized to have unit sum
...	Further arguments passed to <code>adaptIntegrate()</code>

### 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  $\prod_{i=1}^n p_i^{powers[i]}$  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  $p^{\text{power}}$
- 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

```
data(chess)
mean_hyper2(chess)
maxp(chess)
```

```
# disallowed argument typically means slow run times; use high tol for speed
probability(chess,disallowed=function(p){p[1]>p[2]},tol=0.1)
probability(chess,disallowed=function(p){p[1]<p[2]},tol=0.1)
```

---

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 `ggr1()`, 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 `ggr1()`.

### Author(s)

Robin K. S. Hankin

### See Also

[rank\\_likelihood](#)

### Examples

```
x <- sample(9)
names(x) <- sample(letters[1:9])
H <- rank_likelihood(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

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.

### References

- <https://en.chessbase.com/>

### See Also

[karpov\\_kasparov\\_anand](#)

### Examples

```
data(chess)
```

```
maxp(chess)
```

```
mgf(chess, c(Anand=2))
```

---

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=XKWz1G4jDnI>
- [https://en.wikipedia.org/wiki/FaZe\\_Clan](https://en.wikipedia.org/wiki/FaZe_Clan)
- <https://en.wikipedia.org/wiki/Cloud9>

## Examples

```
data(counterstrike)
dotchart(maxp(counterstrike))
```

---

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](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)
GD(alpha, beta, beta0=0)
GD_wong(alpha, beta)
rdirichlet(n,H)
is.dirichlet(H)
rp_unif(n,H)
```

**Arguments**

<code>powers</code>	In function <code>dirichlet()</code> a (named) vector of powers
<code>alpha, beta</code>	A vector of parameters for the Dirichlet or generalized Dirichlet distribution
<code>beta0</code>	In function <code>GD()</code> , an arbitrary parameter
<code>H</code>	Object of class <code>hyper2</code>
<code>n</code>	Number of observations

**Details**

These functions are really convenience functions.

Function `rdirichlet()` returns random samples drawn from a Dirichlet 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 vector of parameters  $\alpha$  [**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`.

**Note**

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

**Author(s)**

Robin K. S. Hankin

**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))
```

---

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](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

**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, a numeric vector is returned but the elements may be in any order (the brackets and indeed the powers are not stored in any particular order).

The replace method, `H[L] <-value`, the index `L` is a list specifying the brackets to be overwritten; argument `value` is not recycled unless it is of length 1.

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"] %<>% inc
chess[c("Topalov", "Kasparov")] %<>% dec

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

---

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, total=1)
indep(x)
```



**Arguments**

x	Numeric vector
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  $\delta p$ ; 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))
```

---

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 <code>formula1_points_systems()</code>
-----	---

**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 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.

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 likelihood function `formula1` is `ordertable2supp(F1_table_2017)`.

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

## 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](https://en.wikipedia.org/w/index.php?title=2017_Formula_One_World_Championship&oldid=839923210) [Online; accessed 14-May-2018]

## See Also

[ordertable2supp](#)

## 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

```
ggr1(H, ...)
general_grouped_rank_likelihood(H, ...)
choose_losers(H, all_players, losers)
choose_winners(H, all_players, winners)
elimination(all_players)
rank_likelihood(M, times=1)
rankvec_likelihood(v)
race(v)
```

**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
M	In function rank_likelihood(), a matrix with each row corresponding to a race (or judge). The columns correspond to the finishing order; thus $a=M[i, j]$ means that competitor a finished in place j in race i
times	Vector specifying the number of times each row is observed
v	A character vector specifying ranks. Thus $c("b", "c", "a")$ means that b came first, c second, and a third

**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 tournaments.

Function ggrrl() is an easily-typed alias for general\_grouped\_rank\_likelihood().

Functions choose\_winners() and choose\_losers() take a hyper2 object H (a likelihood function) and return a list of hyper2 objects. The evaluate may be found with function maxplist().

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  $i^{\text{th}}$  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() takes a matrix M with rows corresponding to a judge (or race); column names are interpreted as competitor names. qA named vector is coerced to a one-row matrix. Each row of M is an order statistic: thus  $c(3, 4, 2, 1)$  means that person 3 came first, person 4 came second, person 2 came third and person 1 came last. Note that in data frames like F1\_table\_2017, each *column* is a race.

Function rankvec\_likelihood() 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 ggrrl() are given in inst/loser.Rmd and inst/masterchef.Rmd.

**Author(s)**

Robin K. S. Hankin

**See Also**

[rrank, ordertable2supp](#)

**Examples**

```

W <- hyper2(pnames=letters[1:7])
W1 <- ggr1(W, 'a', letters[2:6], 'g') # 24-element list
W2 <- ggr1(W, 'b', letters[3:7], 'a') # 24-element list

like_single_list(rep(1/7,6),W1)
like_series(rep(1/7,6),list(W1,W2))

# run 10 races:
r1 <- rrank(10,p=(7:1)/28)
colnames(r1) <- letters[1:7]

# Likelihood function for r1:
W <- rank_likelihood(r1)

H <- hyper2()
for(i in 1:10){H <- H+race(sample(letters[1:9],sample(6,1),replace=FALSE))}
equalp.test(H)$p.value # should not be significant (null is true)

H <- hyper2(pnames=letters[1:5])
H2 <- choose_losers(H,letters[1:4],letters[1:2]) # {a,b} vs {c,d}; {a,b} lost
maxplist(H2)

```

---

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 <code>hessian(border=TRUE)</code>
give	Boolean with default FALSE meaning for function <code>is_ok_hessian()</code> 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` must be a vector of length  $n - 1$ , typically `indep(p)`; and the returned gradient is also a vector of length  $n - 1$ . The hard part of this is accounting for the fillup value. The function returns the derivative of the loglikelihood with respect to the  $n - 1$  independent components of  $(p_1, \dots, p_n)$ , namely  $(p_1, \dots, p_{n-1})$ . The fillup value  $p_n$  is calculated as  $1 - (p_1 + \dots + p_{n-1})$ .

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 returned gradient is also a 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::diag(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`.

## 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  $(p_1, \dots, p_n)$ , namely  $(p_1, \dots, p_{n-1})$ . The fillup value  $p_n$  is calculated as  $1 - (p_1, \dots, p_{n-1})$ .

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 exactly 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	<i>Dataset on communication breakdown in handover between physicians</i>
----------	--

---

### 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 **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  $2 \times 2$  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  $2 \times 2$  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*

---

**Description**

First few terms in a hyperdirichlet distribution

**Usage**

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

**Arguments**

<code>x</code>	Object of class hyper2
<code>...</code>	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 <- (10:1)/55  
names(p) <- letters[1:10]  
H <- rank_likelihood(rrank(100,p))  
head(H)
```

---

`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 <i>only if</i> 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.

Function is\_valid\_hyper2() tests for valid input, returning a Boolean. 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  $p = (p_1, \dots, p_n)$  where  $p_1 + \dots + p_n = 1$ . Note that function lhyper2() takes the vector  $p = (p_1, \dots, p_{n-1})$ .

**Note**

The empty string is a perfectly valid name

**Author(s)**

Robin K. S. Hankin

**See Also**

[Ops.hyper2](#), [Extract.hyper2](#), [loglik](#), [hyper2-package](#)



**Examples**

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

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

---

 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)`.

**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

**LF** 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. O'Neill 2007. *An Iconic Approach to Communicating Climate Change*, University of East Anglia, School of Environmental Science (in prep)
- 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

## 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("b",c("a", "b"), "c",c("b", "c")),c(2,4,3,5))
> H
a * (a + b)^4 * b^2 * (b + c)^5 * 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`). Alternatively we could use `magrittr` pipes:

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

The `inc` and `dec` operators are used in `inst/rowing_analysis.R`; and the `trial()` function is used in `inst/kka_draws.R`.

### 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

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_re`

### See Also

[chess,karpov\\_kasparov\\_anand](#)

### Examples

```
pie(interzonal_maxp)

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

---

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 <http://eigentaste.berkeley.edu/dataset/>. Object `jester_maxp` is the result of running `maxp(jester)`.

Objects `jester` 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.

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.

## 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

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

loglik(indep(jester_maxp), jester)
```

---

karate	<i>Karate dataset</i>
--------	-----------------------

---

## Description

Dataset from the 2018 World Karate Championships, men's 67kg.

## 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 and `karate_maxp` the evaluate.

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](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 matrices of game results

## Details

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

Likelihood functions `karpov_kasparov_anand`, `kka_3draws` and `kka_3whites` are created by files `inst/karpov_kasparov_anand.R`, `inst/kka_3draws` and `inst/kka_3whites`, which are heavily documented and include some analysis. 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 list 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.

The three matrices `plays_white_wins`, `plays_white_draws`, and `plays_white_loses` tabulate this information in a coherent way; and array `kka_array` presents the same information in a 3D array (but the names of the dimnames are lost).

All data drawn from `chessgames.com`, specifically

<https://www.chessgames.com/perl/ezsearch.pl?search=karpov+vs+kasparov>

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

```
data(karpov_kasparov_anand)
maxp(karpov_kasparov_anand)
pie(maxp(karpov_kasparov_anand))
```

---

keep	<i>Keep or discard players</i>
------	--------------------------------

---

### 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 <code>wikitables_to_ranktable()</code> , 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.Rd`.**

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**

```

data("oneill")
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	<i>Length method for hyper2 objects</i>
---------------	---

---

**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	<i>Log likelihood functions</i>
--------	---------------------------------

---

**Description**

Returns a log-likelihood for a given hyper2 object at a specific point



**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
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 takes the vector  $p = (p_1, \dots, p_{n-1})$  and returns the (log) likelihood. Note that the ‘fillup’ value is automatically appended; the elements of  $p$  are (linearly!) independent.

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

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

Function `loglik_single()` is a helper function that takes a single point in probability space.

Functions `like_single_list()` and `like_series()` are intended for use with `ggr1()`.

**Note**

*Likelihood* is defined up to an arbitrary multiplicative constant.

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

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)
```

```
data(masterchef)
```

```
a1 <- rep(1/13, 12)           # equal strengths
a2 <- indep(masterchef_maxp) # MLE
```

```

like_series(a1,masterchef)
like_series(a2,masterchef)

W <- hyper2(pnames=letters[1:6])
W1 <- ggr1(W, 'a', letters[2:5], 'f') # 24-element list
W2 <- ggr1(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

```

---

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](https://en.wikipedia.org/w/index.php?title=MasterChef_Australia_(series_6)&oldid=758432561) (accessed January 5, 2017).

## See Also

[ggr1](#)

## Examples

```
data(masterchef)

a1 <- rep(1/13,12)           # equal strengths
a2 <- indep(masterchef_maxp) # MLE
a3 <- indep(masterchef_constrained_maxp) # constrained MLE

like_series(a1, masterchef)
like_series(a2, masterchef)
like_series(a3, masterchef)
```

---

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=10,
     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_simplex(H, n=100, show=FALSE, give=FALSE, ...)
equalp(H)
```

**Arguments**

H	A hyper2 object
Hlist	A list with elements all hyper2 objects
startp	A vector of probabilities
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
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 <code>constrOptim()</code> with slightly differing start points, to avoid the stupid <code>wmmin is not finite</code> bug, <a href="#">bugzilla id 17703</a>
...	Further arguments which <code>maxp()</code> passes to <code>constrOptim()</code>

**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 `ggr1()`). 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()`.

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 unit-sum 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 used for speed reasons. Use the default `n=10` or greater in production work.

This functionality is peculiarly susceptible to off-by-one errors.

## Author(s)

Robin K. S. Hankin

## See Also

[gradient,fillup](#)

## Examples

```
maxp(icons)

W <- hyper2(pnames=letters[1:5])
W1 <- ggr1(W, 'a', letters[2:3], 'd') # W1 is a suplist object
maxp(W1)
```

---

moto	<i>MotoGP dataset</i>
------	-----------------------

---

**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](https://en.wikipedia.org/w/index.php?title=2019_MotoGP_season&oldid=939711064)

**See Also**

[ordertable2supp](#)

**Examples**

```
pie(moto_maxp)
```

---

mult_grid	<i>Kronecker matrix functionality</i>
-----------	---------------------------------------

---

**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 `ggr1()`.

**Author(s)**

Robin K. S. Hankin

**See Also**

[ggr1](#)

**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.

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  $H1 + H1 + \dots + H1$ , corresponding to repeated 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**

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

**Details**

If two independent datasets have hyper2 objects  $H1$  and  $H2$ , then the R 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).

### 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)
```

---

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`.

Consider `pentathlon_ordertable`:

```
> pentathlon_table
      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 `ordertables`. 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.Rd`). 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.Rd`).

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_2014
      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()`.

### Author(s)

Robin K. S Hankin

**See Also**

[ordertable2supp](#), [rrank](#), [ranktable](#), [as.ordertable](#)

**Examples**

```
ordertable_to_ranktable(soling_table)
ordertable2supp(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**

<code>o</code>	Order table
<code>points</code>	A numeric vector indicating number of points awarded for first, second, third, etc placing
<code>totals</code>	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)

ordertable2points(ranktable_to_ordertable(rrank(9, volvo_maxp)), 1)
```

---

ordertable2supp      *Translate order tables to support functions*

---

## Description

Wikipedia gives a nice summary in table form of Formula 1 racing results on pages like [https://en.wikipedia.org/wiki/2017\\_Formula\\_One\\_World\\_Championship](https://en.wikipedia.org/wiki/2017_Formula_One_World_Championship) (at *World Drivers' Championship standings*) but the data format is commonly used for many sports [see `ordertable.Rd`] and function `ordertable2supp()` translates such tables into a hyper2 support function and also a order table.

Both functions interpret zero to mean “Did not finish” (wikipedia usually signifies DNF as a blank).

## Usage

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

## Arguments

<code>x</code>	Data frame, see details
<code>d</code>	A named numeric vector giving order; zero entries are interpreted as that competitor coming last (due to, e.g., not finishing)
<code>incomplete</code>	Boolean, with FALSE meaning to insist that each rank 1, 2, ..., $n$ is present [zero meaning did not place] and default TRUE allowing for gaps. See examples.
<code>noscore</code>	Character vector giving the abbreviations for a non-finishing status such as “did not finish” or “disqualified”. A missing argument is interpreted as <code>c("Ret", "WD", "DNS", "DSQ", "DNF")</code> .

## Details

Function `ordertable2supp()` is intended for use on order tables such as found at [https://en.wikipedia.org/wiki/2019\\_Moto3\\_season](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)
```

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

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.

### Value

Returns a `hyper2` object

### Author(s)

Robin K. S. Hankin

### See Also

[ordertable](#)

### Examples

```

data(formula1)
ordertable2supp(F1_table_2017)

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

## ordervec2supp() may be added [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	<i>Order transformation</i>
------------	-----------------------------

---

### Description

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

### Usage

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

### Arguments

<code>x</code>	A (generalized) order vector
<code>players</code>	A character vector specifying the order in which the players will be listed; if missing, use <code>sort(names(x))</code>
<code>ox, oy</code>	Rank vectors
<code>...</code>	Further arguments, passed to <code>plot()</code>

### 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 file `inst/ordertrans.Rmd` provides further discussion and examples.

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

**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.Rd`). 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)
ordertrans(x, letters[1:9])

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

ordertrans(o)

#rL <- maxp(volvo2014) # ranks Likelihood
#rL[] <- rank(-maxp(volvo2014))

#r1 <- volvo_table_2014[,1] # ranks race 1
#names(r1) <- rownames(volvo_table_2014)
#ordertransplot(rL, r1)
```

---

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](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](https://en.wikipedia.org/w/index.php?title=2018_F1_Powerboat_World_Championship&oldid=920386507)

### See Also

[ordertable2supp](#)

### Examples

```
data(powerboat)
jj <-
  read.table(system.file("powerboat.txt", package="hyper2"))
identical(jj, powerboat_table) # should be TRUE
```

---

Print

*Print methods*

---

### Description

Print methods for hyper2 objects

### Usage

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

### Arguments

<code>x</code>	An object of class hyper2
<code>...</code>	Further arguments, currently ignored

### Value

Returns the hyper2 object it was sent, invisibly. Used mainly for its 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()`.

### Author(s)

Robin K. S. Hankin

**Examples**

```
data(chess)
chess
```

---

 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	Element to be tested, character accepted
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 <code>profile_support_single()</code> .
...	Arguments passed to <code>maxp()</code>

**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**

```
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))
```

---

pwa	<i>Player with advantage</i>
-----	------------------------------

---

### 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 <code>hyper2</code> object
pwa	A list of the players with the supposed advantage; may be character in the case of a named <code>hyper2</code> 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

[ordvec2supp](#)

**Examples**

```

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

formula1 %>% pwa("Hamilton", "pole")

## Three five-way races:
H1 <- ordervec2supp(c(a = 1, b = 5, c = 3, d = 4, e = 2))
H2 <- ordervec2supp(c(a = 2, b = 3, c = 1, d = 4, e = 5))
H3 <- ordervec2supp(c(a = 4, b = 3, c = 0, d = 1, e = 2))

## Now it is revealed that a,b,c had some advantage in races 1,2,3 respectively:
Ht <- pwa(H1, 'a') + pwa(H2, 'b') + pwa(H2, 'c')

specificp.test(Ht, "S")

```

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, ...)
ranktable_to_printable_object(x)
## 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 `wiki_table_to_ranktable()` or `ordertable2supp()` to produce a likelihood function.

The print method is discussed in vignette `inst/ordertable_to_ranktable.Rmd`.

**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)
xorder <- ranktable_to_ordertable(xrank)

## Can convert back and forth:
identical(xrank,ordertable_to_ranktable(ranktable_to_ordertable(xrank)))

maxp(ordertable2supp(xorder)) # should be close to p

ordertable_to_ranktable(skating_table)
```

---

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](#)

**Examples**

```
dotchart(maxp(rhyper2(n=8,s=10)))
dotchart(maxp(rhyper2(n=26,s=2,race=FALSE)))
```

---

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.

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%93Men%27s\\_single\\_sculls&oldid=753517240](https://en.wikipedia.org/w/index.php?title=Rowing_at_the_2016_Summer_Olympics_%E2%80%93Men%27s_single_sculls&oldid=753517240) (accessed December 7, 2016).

## See Also

[ggr1](#)

## Examples

```
dotchart(rowing_maxp)
```

---

rp	<i>Random samples from the prior of a hyper2 object</i>
----	---

---

## 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,...)
```

## 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()
...	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 likelihood function based on random observations, use `rhyper2()`.

**Author(s)**

Robin K. S. Hankin

**See Also**

[maxp](#), [loglik](#), [dirichlet](#), [rhyper2](#)

**Examples**

```
rp(10, icons)

plot(loglik(indep(rp(100, icons)), icons))
```

---

 rrank

*Random ranks*


---

**Description**

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

**Usage**

```
rrank(n = 1, p, pnames=NULL, fill = FALSE, rnames=NULL)
## S3 method for class 'ranktable'
print(x, ...)
```

**Arguments**

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

**Value**

If `n=1`, return a vector; if `n>1` return 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(2, 5, 1, 3, 4)`, then competitor 2 came first, competitor 5 came second, competitor 1 came third, and so on.

Note that function `rrank()` returns an object of class `ranktable`, which has its own special print method. 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 Plackett-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.

Function `rrank()` is designed for `rank_likelihood()`, which needs rank data, not order data. Vignette “`skating_analysis`” gives another discussion.

Note that function `rrank()` returns an object of class “`rrank`”, which has its own print method that returns NA, intentionally. This can be confusing.

### Author(s)

Robin K. S. Hankin

### See Also

[ordertrans](#), [rank\\_likelihood](#), [skating](#)

### Examples

```

ptrue <- (4:1)/10
names(ptrue) <- letters[1:4]
rrank(100, p=ptrue)

r1 <- rrank(100, p=ptrue)
r2 <- rrank(10, p=ptrue)

```

```
H <- rank_likelihood(r1)
maxp(H) # should be close to ptrue

H <- H + rank_likelihood(rrank(100,maxp(H)))
maxp(H)
```

---

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

```
data(skating)
```

## Details

There are three datasets loaded by `data("skating")`: `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_analysis.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 `rank_likelihood()` 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.

## Author(s)

Robin K. S. Hankin

## References

- [https://en.wikipedia.org/wiki/Figure\\_skating\\_at\\_the\\_2002\\_Winter\\_Olympics#Full\\_results\\_2](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)
```

```
r0 <- seq_len(nrow(skating_table))
names(r0) <- rownames(skating_table)
ordertransplot(r0,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](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)
```

---

suplic                        *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 `ggr1()`.

### Usage

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

**Arguments**

<code>e1, e2</code>	Objects of class <code>suplist</code> , here interpreted as a list of possible likelihood functions (who should be added)
<code>x, ..., na.rm</code>	In the <code>sum()</code> method, objects to be summed; <code>na.rm</code> is currently ignored
<code>L</code>	A list of <code>hyper2</code> 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  $S = \text{list}(H_1, H_2, \dots, H_n)$  where the  $H_i$  are `hyper2` objects, then  $\text{Prob}(p|H_1) + \dots + \text{Prob}(p|H_n)$ . 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  $\mathcal{L}_{H_1 \cup H_2}(p) = C_{12} (\text{Prob}(p|H_1) + \text{Prob}(p|H_2))$  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  $S_1$  and  $S_2$ . The idea is that the support functions “add” in the following sense. If  $S_1 = \text{list}(H_1, \dots, H_r)$  and  $S_2 = \text{list}(I_1, \dots, I_s)$  where  $H_x, I_x$  are `hyper2` objects, then the likelihood function for “S1+S2” is the likelihood function for  $S_1$  followed by (independent)  $S_2$ . Formally

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

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

However,  $S_1 + S_2$  is typically a large and unwieldy object, and can be very slow to evaluate. These functions are here because they provide slick R idiom.

**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 <- ggr1(W, 'a', letters[2:3], 'd') # 2-element list
W2 <- ggr1(W, 'e', letters[1:3], 'd') # 6-element list
W3 <- ggr1(W, 'c', letters[4:5], 'a') # 2-element list

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

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

like_single_list(indep(equalp(W)),W1+W2+W3)

dotchart(maxplist(W1+W1+W3),pch=16)

```

---

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**

```

dotchart(T20_maxp)
dotchart(maxp(T20_toss))

```

---

table_tennis	<i>Match outcomes from repeated table tennis matches</i>
--------------	--

---

**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	<i>Match outcomes from repeated doubles tennis matches</i>
--------	--

---

**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, <https://www.jstatsoft.org/v33/i11/>

### Examples

```
data(tennis)
dotchart(maxp(tennis))
eigen(maxp(tennis, give=TRUE, hessian=TRUE)$hessian, TRUE, TRUE)$values

maxp(tennis_ghost)
```

---

tests

*Hypothesis testing*

---

### Description

Tests different nulls against a free alternative



**Usage**

```

equalp.test(H, ...)
samep.test(H, i, give=FALSE, ...)
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
...	Further arguments passed by equalp.test() to maxp() and ignored by print.hyper2test()
i, specificp	Function specificp.test() tests the hypothesis that $p_i$ is equal to specificp. Function samep.test() has i an integer vector of strengths. The null is that $p[i[1]] == p[i[2]] == \dots == p[i[r]]$
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 the 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() tests  $H_0: p_1 = p_2 = \dots = p_n = \frac{1}{n}$
- Function specificp.test(H, i, p) tests  $H_0: p_i = p$
- Function samep.test() tests  $H_0: p_{i_1} = p_{i_2} = \dots = p_{i_k}$ .
- 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`. 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.

**See Also**

[maxp](#)

**Examples**

```
equalp.test(volvo)
samep.test(icons,c("NB","L"))
```

---

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](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**

```
pie(maxp(universities))
```

---

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.

### 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, <https://www.jstatsoft.org/v33/i11/>

### 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](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)
```

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 <code>size(H)-1</code>
...	Further arguments, passed to <code>maxp()</code>

**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 am investigating this.

**Author(s)**

Robin K. S. Hankin

**See Also**

[discard\\_flawed](#), [maxp](#)

**Examples**

```
zapweak(icons)      # removes noone
#Takes too long
zapweak(rowing)    # removes everyone...
```

# Index

- \* **datasets**
  - chess, 9
  - counterstrike, 10
  - curling, 12
  - handover, 22
  - icons, 25
  - interzonal, 28
  - jester, 28
  - karpov\_kasparov\_anand, 30
  - masterchef, 34
  - NBA, 39
  - rowing, 54
  - skating, 58
  - T20, 62
  - table\_tennis, 63
  - tennis, 63
  - volleyball, 68
- \* **package**
  - hyper2-package, 3
- \* **symbolmath**
  - Ops.hyper2, 40
  - suplist, 60
- [.hyper2 (Extract), 15
- [<- .hyper2 (Extract), 15
  
- accessor (cplusplus), 11
- additional\_strength (pwa), 51
- addL (cplusplus), 11
- allequal (maxp), 36
- allowers (rowing), 54
- as.hyper2 (hyper2), 23
- as.ordertable, 5, 43
- as.suplist (suplist), 60
- assign\_lowlevel (Extract), 15
- assigner (cplusplus), 11
  
- B, 6
- basketball (NBA), 39
- black\_wins (karpov\_kasparov\_anand), 30
- bordered\_hessian (gradient), 20
- brackets (hyper2), 23
  
- chameleon (pwa), 51
- char2num (character\_to\_number), 8
  
- character\_to\_number, 8
- chess, 9, 28, 30
- chess\_maxp (chess), 9
- chess\_table (chess), 9
- choose\_losers (ggol), 18
- choose\_winners (ggol), 18
- collusion (interzonal), 28
- Connor (dirichlet), 13
- counterstrike, 10
- counterstrike\_likelihood (counterstrike), 10
- counterstrike\_maxp (counterstrike), 10
- cplusplus, 11
- curacao (interzonal), 28
- curacao3 (interzonal), 28
- curling, 12
- curling1 (curling), 12
- curling1\_maxp (curling), 12
- curling2 (curling), 12
- curling2\_maxp (curling), 12
- curling\_maxp (curling), 12
- curling\_table (curling), 12
  
- dec (increment), 26
- decrement (increment), 26
- dhyper2 (B), 6
- dhyper2\_e (B), 6
- differentiate (cplusplus), 11
- differentiate\_n (cplusplus), 11
- Dirichlet (dirichlet), 13
- dirichlet, 13, 56
- discard (keep), 31
- discard\_flawed, 70
- discard\_flawed (keep), 31
- discard\_flawed2 (keep), 31
- doubles (tennis), 63
- doubles\_ghost (tennis), 63
- doubles\_noghost (tennis), 63
- drawn\_games (karpov\_kasparov\_anand), 30
- drop (keep), 31
- drop\_flawed (keep), 31
  
- e\_to\_p (B), 6
- elimination (ggol), 18

- equal (cplusplus), 11
- equality (cplusplus), 11
- equalp, 17
- equalp (maxp), 36
- equalp.test (tests), 64
- equalprobs (maxp), 36
- euro (eurovision), 14
- euro2009 (eurovision), 14
- Eurovision (eurovision), 14
- eurovision, 14
- Eurovision2009 (eurovision), 14
- eurovision2009 (eurovision), 14
- eurovision2009\_votingtable (eurovision), 14
- eurovision\_maxp (eurovision), 14
- Eurovision\_song\_contest (eurovision), 14
- eurovision\_table (eurovision), 14
- evaluate (cplusplus), 11
- extra\_strength (pwa), 51
- Extract, 15, 61
- extract (Extract), 15
- Extract.hyper2, 24
- Extract.hyper2 (Extract), 15
- extractor (Extract), 15
  
- F1 (formula1), 17
- F1\_2014 (formula1), 17
- F1\_2015 (formula1), 17
- F1\_2016 (formula1), 17
- F1\_2017 (formula1), 17
- F1\_2018 (formula1), 17
- F1\_2019 (formula1), 17
- F1\_points\_2017 (formula1), 17
- F1\_table\_2016 (formula1), 17
- F1\_table\_2017 (formula1), 17
- F1\_table\_2018 (formula1), 17
- F1\_table\_2019 (formula1), 17
- fillup, 16, 37
- formula1, 17
- formula1\_2017\_table (formula1), 17
- formula1\_points\_2017 (formula1), 17
- formula1\_points\_systems (formula1), 17
- formula1\_table\_2017 (formula1), 17
- formula\_1 (formula1), 17
- formula\_one (formula1), 17
  
- GD (dirichlet), 13
- gd (dirichlet), 13
- GD\_wong (dirichlet), 13
- general\_grouped\_order\_likelihood (ggol), 18
- general\_grouped\_rank\_likelihood (ggol), 18
  
- ggol, 18
- ggol, 34, 39, 55
- ggol (ggol), 18
- gradient, 17, 20, 37
- gradientn (gradient), 20
  
- handoff (handover), 22
- handover, 22
- handover\_maxp (handover), 22
- handover\_table (handover), 22
- head.hyper2, 23
- hessian (gradient), 20
- hessian\_bordered (gradient), 20
- hessian\_lowlevel (gradient), 20
- hyper2, 13, 16, 23, 60
- hyper2-package, 3
- hyper2\_accessor (cplusplus), 11
- hyper2\_add (Ops.hyper2), 40
- hyper2\_addL (cplusplus), 11
- hyper2\_assigner (cplusplus), 11
- hyper2\_differentiate (cplusplus), 11
- hyper2\_equal (cplusplus), 11
- hyper2\_evaluate (cplusplus), 11
- hyper2\_identityL (cplusplus), 11
- hyper2\_overwrite (cplusplus), 11
- hyper2\_prod (Ops.hyper2), 40
- hyper2\_sum\_numeric (Ops.hyper2), 40
  
- icons, 25, 35
- icons\_matrix (icons), 25
- icons\_maxp (icons), 25
- icons\_table (icons), 25
- identityL (cplusplus), 11
- inc (increment), 26
- increment, 26
- indep (fillup), 16
- interzonal, 28
- interzonal\_collusion (interzonal), 28
- interzonal\_collusion\_maxp (interzonal), 28
- interzonal\_maxp (interzonal), 28
- interzonal\_table (interzonal), 28
- is.dirichlet (dirichlet), 13
- is.hyper2 (hyper2), 23
- is\_constant (hyper2), 23
- is\_ok\_hessian (gradient), 20
- is\_valid\_hyper2 (hyper2), 23
  
- Jacobian (B), 6
- jester, 28
- jester\_maxp (jester), 28
  
- karate, 29



- karate\_maxp (karate), 29
- karate\_table (karate), 29
- karpov\_kasparov\_anand, 9, 28, 30
- keep, 31
- keep\_flawed (keep), 31
- keep\_flawed2 (keep), 31
- kka (karpov\_kasparov\_anand), 30
- kka\_3draws (karpov\_kasparov\_anand), 30
- kka\_3whites (karpov\_kasparov\_anand), 30
- kka\_array (karpov\_kasparov\_anand), 30
  
- length (length.hyper2), 32
- length.hyper2, 32
- like\_series (loglik), 32
- like\_single\_list (loglik), 32
- loglik, 7, 24, 32, 50, 56, 61
- loglik\_single (loglik), 32
  
- malpractice (handover), 22
- MasterChef (masterchef), 34
- masterchef, 34
- masterchef\_constrained\_maxp (masterchef), 34
- masterchef\_maxp (masterchef), 34
- matrix2supp, 26, 35
- maxjest (jester), 28
- maxp, 33, 36, 56, 66, 70
- maxp\_simplex (maxp), 36
- maxp\_single (maxp), 36
- maxplist (maxp), 36
- mean (B), 6
- mean\_hyper2 (B), 6
- mgf (B), 6
- Mosimann (dirichlet), 13
- moto, 38
- moto\_maxp (moto), 38
- moto\_table (moto), 38
- motoGP (moto), 38
- motoGP\_2019 (moto), 38
- mult\_grid, 38
  
- NBA, 39
- NBA\_likelihood (NBA), 39
- NBA\_maxp (NBA), 39
- NBA\_table (NBA), 39
  
- oneill (icons), 25
- Ops (Ops.hyper2), 40
- Ops.hyper2, 16, 24, 40, 61
- Ops.suplist (suplist), 60
- order\_obs (ordertable2supp), 44
- order\_table (ordertable), 41
- ordertable, 6, 41, 43, 45, 48, 67
- ordertable2points, 43
- ordertable2supp, 6, 18, 19, 31, 38, 43, 44, 49, 53, 59, 69
- ordertable\_to\_ranktable (ranktable), 52
- ordertrans, 46, 57
- ordertransplot (ordertrans), 46
- ordervec2supp, 51
- ordervec2supp (ordertable2supp), 44
- overwrite (cplusplus), 11
- overwrite\_lowlevel (Extract), 15
  
- p\_to\_e (B), 6
- pair\_grid (mult\_grid), 38
- pentathlon, 47
- pentathlon\_maxp (pentathlon), 47
- pentathlon\_ordertable (pentathlon), 47
- pentathlon\_table (pentathlon), 47
- ping\_pong (table\_tennis), 63
- plays\_white\_draws (karpov\_kasparov\_anand), 30
- plays\_white\_loses (karpov\_kasparov\_anand), 30
- plays\_white\_wins (karpov\_kasparov\_anand), 30
- pnames (hyper2), 23
- pnames<- (hyper2), 23
- powerboat, 48
- powerboat2018 (powerboat), 48
- powerboat\_2018 (powerboat), 48
- powerboat\_maxp (powerboat), 48
- powerboat\_table (powerboat), 48
- powers (hyper2), 23
- Print, 49
- print (Print), 49
- print.equalptest (tests), 64
- print.hyper2test (tests), 64
- print.ranktable (rrank), 56
- print.ranktablesummary (ranktable), 52
- print.summary.hyper2 (summary.hyper2), 60
- probability (B), 6
- profile, 50
- profile\_likelihood (profile), 50
- profile\_likelihood\_single (profile), 50
- profile\_support (profile), 50
- profile\_support\_single (profile), 50
- proflike (profile), 50
- profsup (profile), 50
- profsupp (profile), 50
- pwa, 51
  
- race (ggol), 18
- rank\_likelihood, 8, 57

- rank\_likelihood (ggol), 18
- ranktable, 43, 52
- ranktable\_to\_ordertable (ranktable), 52
- ranktable\_to\_printable\_object (ranktable), 52
- rankvec\_likelihood (ggol), 18
- rdirichlet (dirichlet), 13
- retain (keep), 31
- retain\_flawed (keep), 31
- rhyper2, 53, 56
- rock\_paper\_scissors (chess), 9
- rowing, 54
- rowing\_maxp (rowing), 54
- rowing\_minimal (rowing), 54
- rowing\_minimal\_maxp (rowing), 54
- rp, 13, 54, 55
- rp\_unif (dirichlet), 13
- rrank, 19, 43, 47, 53, 56
- saffy (matrix2supp), 35
- samep.test (tests), 64
- sculling (rowing), 54
- sculls2016 (rowing), 54
- size (hyper2), 23
- skating, 57, 58
- skating\_maxp (skating), 58
- skating\_table (skating), 58
- soling, 59
- soling2000 (soling), 59
- soling2000\_qf (soling), 59
- soling2000\_rr1 (soling), 59
- soling2000\_rr2 (soling), 59
- soling\_after (soling), 59
- soling\_after\_maxp (soling), 59
- soling\_maxp (soling), 59
- soling\_qf (soling), 59
- soling\_rr1 (soling), 59
- soling\_rr2 (soling), 59
- soling\_table (soling), 59
- soling\_table\_2000 (soling), 59
- specificp.ge.test (tests), 64
- specificp.gt.test (tests), 64
- specificp.le.test (tests), 64
- specificp.lt.test (tests), 64
- specificp.ne.test (tests), 64
- specificp.test (tests), 64
- stockholm1962 (interzonal), 28
- sum.hyper2 (Ops.hyper2), 40
- sum.suplist (suplist), 60
- summary.hyper2, 60
- summary.ranktable (ranktable), 52
- suplist, 60
- suplist\_add (suplist), 60
- T20, 62
- T20\_maxp (T20), 62
- T20\_table (T20), 62
- T20\_toss (T20), 62
- T20\_toss\_maxp (T20), 62
- table\_tennis, 63
- table\_tennis\_serve (table\_tennis), 63
- tennis, 63
- tennis\_ghost (tennis), 63
- tennis\_ghost\_maxp (tennis), 63
- tennis\_maxp (tennis), 63
- tennis\_noghost (tennis), 63
- tests, 64
- tidy, 31, 66
- training\_strength (pwa), 51
- trial (increment), 26
- universities, 67
- universities\_maxp (universities), 67
- universities\_table (universities), 67
- vb (volleyball), 68
- vb\_synthetic (volleyball), 68
- volley (matrix2supp), 35
- volleyball, 35, 40, 68
- volleyball\_matrix (volleyball), 68
- volleyball\_maxp (volleyball), 68
- volleyball\_results (volleyball), 68
- volleyball\_table (volleyball), 68
- volvo, 69
- volvo2014 (volvo), 69
- volvo\_maxp (volvo), 69
- volvo\_ocean\_race (volvo), 69
- volvo\_table (volvo), 69
- volvo\_table\_2014 (volvo), 69
- wet\_strength (pwa), 51
- white\_strength (pwa), 51
- white\_wins (karpov\_kasparov\_anand), 30
- wikitable\_to\_ranktable (ranktable), 52
- zacslis (counterstrike), 10
- zapweak, 29, 70