Package 'UCS'

August 15, 2005

Version 0.5

Title The UCS/R libraries

Author Stefan Evert <evert@ims.uni-stuttgart.de>

Maintainer Stefan Evert <evert@ims.uni-stuttgart.de>

Depends R (>= 1.6.0), graphics, stats, boot

Description All libraries from the UCS/R system

License Artistic License or GPL (same terms and conditions as Perl)

URL http://www.collocations.de/

R topics documented:

`Cbeta` *The Beta Function (sfunc)*

Description

Computes the (complete) Beta function and its base 10 logarithm.

Usage

```
Cbeta(a, b, log=FALSE)
```

Arguments

`a, b`	numeric vectors
`log`	if `TRUE`, returns the base 10 logarithm of the Beta function (default: `FALSE`)

Details

This is just a front-end to the built-in `beta` and `lbeta` functions, provided mainly for consistent naming. Note that the logarithmic version is scaled to base 10 logarithms, according to the UCS conventions.

Value

The Beta function with arguments (`a`, `b`), or its base 10 logarithm (if `log=TRUE`).

See Also

beta, Ibeta, Rbeta, Cgamma, Igamma, Rgamma

Examples

```
x <- 5
y <- 3
((x+y+1) * beta(x+1,y+1))^-1 # == choose(x+y, x)
```

Cgamma *The Gamma Function (sfunc)*

Description

Computes the (complete) Gamma function and its base 10 logarithm.

Usage

```
Cgamma(a, log=FALSE)
```

Arguments

a	a numeric vector
log	if TRUE, returns the base 10 logarithm of the Gamma function (default: FALSE)

Details

This is just a front-end to the built-in gamma and lgamma functions, provided mainly for consistent naming. Note that the logarithmic version is scaled to base 10 logarithms, according to the UCS conventions.

Value

The Gamma function evaluated at a, or its base 10 logarithm (if log=TRUE).

See Also

gamma, Igamma, Rgamma, Cbeta, Ibeta, Rbeta

Examples

```
Cgamma(5 + 1) # = factorial(5)
```

EV — *Expected Vocabulary Size of a LNRE Model (zm, fzm)*

Description

Computes the expected vocabulary size of a LNRE model (Baayen, 2001) at sample size N.

Usage

```
EV(model, N)
```

Arguments

`model`	an object of class `"zm"` or `"fzm"`, representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model
`N`	a vector of positive integers, representing sample sizes

Details

The expected vocabulary size $E[V(N)]$ is the expected number of types at sample size N, according to the LNRE model `model` (see Baayen, 2001).

Value

a numeric vector of the same length as `N`

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`zm`, `fzm`, `EVm`, `VV`, `VVm`

EVm — *Expected Frequency Spectrum of a LNRE Model (zm, fzm)*

Description

Computes the expected frequency spectrum, relative frequency spectrum, and conditional parameter distribution of a LNRE model (Baayen, 2001) at sample size N.

Usage

```
EVm(model, m, N, rho=1, relative=FALSE, ratio=FALSE, lower=TRUE)
```

Arguments

`model`	an object of class `"zm"` or `"fzm"`, representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model
`m`	a vector of positive integers, representing frequency ranks
`N`	a vector of positive integers, representing sample sizes; either `m` or `N` should be a single number
`rho`	a vector of numbers in the range $[0, 1]$. If `length(rho) > 1`, both `m` and `N` should be single numbers. See below for details.
`relative`	if `TRUE`, computes the relative frequency spectrum (see below for details)
`ratio`	if `TRUE`, computes the ratio between consecutive elements in the expected frequency spectrum
`lower`	if `rho` is specified, controls whether the lower or upper conditional parameter distribution is computed

Details

The expected frequency spectrum consists of the numbers $E[V_m(N)]$, which stand for the expected number of types in frequency class m at sample size N, according to the LNRE model `model` (see Baayen, 2001).

If `relative=TRUE`, the relative frequency spectrum $E[V_m(N)]/E[V(N)]$ is returned. If `ratio=TRUE`, the ratios between consecutive expected class sizes, $E[V_{m+1}(N)]/E[V_m(N)]$, are returned.

When `rho` is specified, the conditional parameter distribution $E[V_{m,\rho}(N)]$ is returned, i.e. the expected number of types in frequency class m at sample size N with probability parameter $\pi \leq \rho$. If `relative=TRUE`, the expected proportion $E[R_{m,\rho}] \approx E[V_{m,\rho}(N)]/E[V(N)]$ is returned instead. With `lower=FALSE`, computes the upper conditional parameter distribution $E[V_{m,>\rho}(N)]$ or proportion $E[R_{m,>\rho}(N)]$. See Evert (2004, Ch. 4) for details.

Value

a numeric vector of appropriate length (determined either by `m`, `N`, or `rho`)

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`zm`, `fzm`, `EVm`, `VV`, `VVm`

Ibeta *The Incomplete Beta Function (sfunc)*

Description

Computes the incomplete Beta function and its inverse. The Beta value can be scaled to a base 10 logarithm.

Usage

```
Ibeta(x, a, b, log=FALSE)

Ibeta.inv(y, a, b, log=FALSE)
```

Arguments

a, b	non-negative numeric vectors, the parameters of the incomplete Beta function
x	a numeric vector with values in the range $[0, 1]$, the point at which the incomplete Beta function is evaluated
y	a numeric vector, the values of the incomplete Beta function (or their base 10 logarithms if log=TRUE)
log	if TRUE, the Beta values are base 10 logarithms (default: FALSE)

Details

The incomplete Beta function is defined by the Beta integral

$$B(x; a, b) = \int_0^x t^{a-1}(1-t)^{b-1}\, dt$$

Value

Ibeta returns the incomplete Beta function with parameters (a,b) evaluated at point x.

Ibeta.inv returns the point x at which the incomplete Beta function with parameters (a,b) evaluates to y.

See Also

Cgamma, Igamma, Rgamma, Cbeta, Rbeta

Igamma *The Incomplete Gamma Function (sfunc)*

Description

Computes the incomplete Gamma function and its inverse. Both the lower and the upper incomplete Gamma function are supported, and the Gamma value can be scaled to a base 10 logarithm.

Usage

```
Igamma(a, x, lower=TRUE, log=FALSE)

Igamma.inv(a, y, lower=TRUE, log=FALSE)
```

Arguments

a	a non-negative numeric vector, the parameter of the incomplete Gamma function
x	a non-negative numeric vector, the point at which the incomplete Gamma function is evaluated
y	a numeric vector, the values of the incomplete Gamma function (or their base 10 logarithms if log=TRUE)
lower	if TRUE, computes the lower incomplete Gamma function (default). Otherwise, computes the upper incomplete Gamma function.
log	if TRUE, the Gamma values are base 10 logarithms (default: FALSE)

Details

The upper incomplete Gamma function is defined by the Gamma integral

$$\Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}\,dt$$

The lower incomplete Gamma function is defined by the complementary Gamma integral

$$\gamma(a,x) = \int_0^x t^{a-1}e^{-t}\,dt$$

Value

Igamma returns the (lower or upper) incomplete Gamma function with parameter a evaluated at point x.

Igamma.inv returns the point x at which the (lower or upper) incomplete Gamma function with parameter a evaluates to y.

See Also

Cgamma, Rgamma, Cbeta, Ibeta, Rbeta

`Rbeta` *The Regularized Beta Function (sfunc)*

Description

Computes the regularized Beta function and its inverse. The Beta value can be scaled to a base 10 logarithm.

Usage

```
Rbeta(x, a, b, log=FALSE)

Rbeta.inv(y, a, b, log=FALSE)
```

Arguments

`a, b`	non-negative numeric vectors, the parameters of the regularized Beta function
`x`	a numeric vector with values in the range $[0, 1]$, the point at which the regularized Beta function is evaluated
`y`	a numeric vector, the values of the regularized Beta function (or their base 10 logarithms if `log=TRUE`)
`log`	if `TRUE`, the Beta values are base 10 logarithms (default: `FALSE`)

Details

The regularized Beta function scales the incomplete Beta function to the interval $[0, 1]$, by dividing through $B(a, b)$, i.e.

$$I(x; a, b) = \frac{B(x; a, b)}{B(a, b)}$$

Value

`Rbeta` returns the regularized Beta function with parameters (`a`,`b`) evaluated at point `x`.

`Rbeta.inv` returns the point `x` at which the regularized Beta function with parameters (`a`,`b`) evaluates to `y`.

See Also

`Cgamma`, `Igamma`, `Rgamma`, `Cbeta`, `Ibeta`

`Rgamma` *The Regularized Gamma Function (sfunc)*

Description

Computes the regularized Gamma function and its inverse. Both the lower and the upper regularized Gamma function are supported, and the Gamma value can be scaled to a base 10 logarithm.

Usage

```
Rgamma(a, x, lower=TRUE, log=FALSE)

Rgamma.inv(a, y, lower=TRUE, log=FALSE)
```

Arguments

`a`	a non-negative numeric vector, the parameter of the incomplete Gamma function
`x`	a non-negative numeric vector, the point at which the incomplete Gamma function is evaluated
`y`	a numeric vector, the values of the regularized Gamma function (or their base 10 logarithms if `log=TRUE`)
`lower`	if `TRUE`, computes the lower regularized Gamma function (default). Otherwise, computes the upper regularized Gamma function.
`log`	if `TRUE`, the Gamma values are base 10 logarithms (default: `FALSE`)

Details

The regularized Gamma functions scale the corresponding incomplete Gamma functions to the interval $[0, 1]$, by dividing through $\Gamma(a)$. Thus, the lower regularized Gamma function is given by

$$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)}$$

and the upper regularized Gamma function is given by

$$Q(a, x) = \frac{\Gamma(a, x)}{\Gamma(a)}$$

Value

`Rgamma` returns the (lower or upper) regularized Gamma function with parameter `a` evaluated at point `x`.

`Rgamma.inv` returns the point `x` at which the (lower or upper) regularized Gamma function with parameter `a` evaluates to `y`.

See Also

`Cgamma`, `Igamma`, `Cbeta`, `Ibeta`, `Rbeta`

Examples

```
## P(X >= k) for Poisson distribution with mean alpha
alpha <- 5
k <- 10
Rgamma(k, alpha) # == ppois(k-1, alpha, lower=FALSE)
```

UCS *Introduction to UCS/R*

Description

UCS/R consists of a set of R libraries related to the visualisation of cooccurrence data and the evaluation of association measures. The current functionaliy includes: evaluation graphs for association measures (in terms of precision and recall), measures for inter-annotator agreement, and two population models for word frequency distributions.

Usage

```
source("/path/to/UCS/System/R/lib/ucs.R")
ucs.library()
```

Details

UCS/R is initialised by `source`ing the file 'ucs.R' in the 'lib/' subdirectory of the UCS/R directory tree. This will make the UCS/R documentation available in the R process and provide the `ucs.library` command, which is used to load individual UCS/R modules. Enter `ucs.library()` now to display a list of available modules (see the `ucs.library` manpage for details).

Currently, the following modules are available. The listing below also indicates the most important manpages for each module. Throughout the documentation, it is assumed that you are familiar with the UCS/Perl naming conventions and data set file format.

- `sfunc`: **Special Mathematical Functions**

  Convenience interfaces to the Gamma function (`Cgamma`), the incomplete (and regularized) Gamma function and its inverse (`Igamma`, `Rgamma`), the Beta function (`Cbeta`), the incomplete (and regularized) Beta function and its inverse (`Ibeta`, `Rbeta`), and binomial confidence intervals (`binom.conf.interval`).

  All these functions are computed from the `pgamma` and `pbeta` distributions (and the corresponding quantile functions) in the standard library of R.

- `base`: **Basic Functions for Loading and Managing UCS data sets**

  This module provides functions for loading UCS data set files (`read.ds.gz`), listing annotated association measures (`ds.find.am`, `am.key2var`), ranking by association scores (`order.by.am`, `add.ranks`), and computing precision/recall tables for the evaluation of association measures (`precision.recall`).

  The module also includes a listing of all built-in association measures in the UCS/Perl system, including add-on packages (`builtin.ams`).

- `plots`: **Evaluation Graphs for Association Measures**
  This module plots precision-, recall-, and precision-by-recall graphs for the empirical evaluation of association measures (all combined in a single function, `evaluation.plot`). The graphs are highly configurable, either locally in each function call or by setting global defaults (`ucs.par`). The `evaluation.plot` function supports confidence intervals, significance tests for result differences, and evaluation based on random samples (see Evert, 2004, Ch. 5). A simple text-mode version of the precision/recall-based evaluation is provided by the `evaluation.table` function in the `base` module.
- `iaa`: **Measures of Inter-Annotator Agreement**
  Computes Cohen's kappa statistic with standard deviation (Fleiss, Cohen & Everitt, 1969) or confidence interval for proportion of true agreement (Krenn, Evert & Zinsmeister, 2004) from a 2×2 contingency table (see `iaa.kappa` and `iaa.pta`)
- `gam`: **Generalised association measures (GAMs)**
  This module implements extensions of several association measures to continuous functions on a real-valued coordinate space (generalised association measures, GAMs). For details and terminology, please refer to Evert (2004, Sec. 3.3). The functions in this module compute GAM scores and iso-surfaces in standard or ebo-coordinates, and can add jitter to a given data set. New GAMs can easily be added with the `register.gam` function. Relevant help pages are `builtin.gams`, `gam.score`, `gam.iso`, `gamma.nbest`, `add.jitter`, `add.gams`, `add.ebo`, and `gam.helpers`.
- `eo`: **Visualise GAMs in the (e,o) plane**
  This module implements 2-D visualisation of data sets and GAMs by plotting point clouds and iso-lines in the (e,o) plane (see Evert 2004, Sec. 3.3). The recommended starting point is the documentation of the `eo.setup` function, which intialises a new (e,o) plot. Other relevant help pages are `eo.par`, `eo.points`, `eo.iso`, `eo.iso.diff`, `eo.legend` and `eo.mark`.
- `lexstats`: **Utilities for lexical statistics**
  This module contains miscellaneous utility functions for word frequency distributions, including: an interface to file formats used by the `lexstats` software (Baayen 2001); a range of common plots; goodness-of-fit evaluation for LNRE populations models (cf. the `zm` and `fzm` modules below). Currently, the most useful functions in this module are `read.spectrum`, `spectrum.plot`, and `lnre.goodness.of.fit`.
- `zm`: **The Zipf-Mandelbrot (ZM) Population Model**
  This module implements a simple population model for word frequency distributions (Baayen, 2001) based on the Zipf-Mandelbrot law. See (Evert, 2004a) for details. Relevant help pages are `zm`, `EV`, `EVm`, `VV`, `VVm`, `write.lexstats`, and `lnre.goodness.of.fit`.
- `fzm`: **The Finite Zipf-Mandelbrot (fZM) Population Model**
  This module implements the finite Zipf-Mandelbrot model, an extension of the ZM model (Evert, 2004a). Relevant help pages are `fzm`, `EV`, `EVm`, `VV`, `VVm`, `write.lexstats`, and `lnre.goodness.of.fit`.

The command `help(package=UCS)` will give you a full index of available UCS/R help pages. Use `help.search()` for full-text search.

Note

The correct `source` path for the file 'ucs.R' can be set automatically with the UCS/R tool `ucs-config`. Simply insert the statement

```
source("ucs.R")
```

on a separate line in your R script file (say, 'my-script.R') and run the shell command

```
ucs-config my-script.R
```

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

Evert, Stefan (2004a). A simple LNRE model for random character sequences. In *Proceedings of JADT 2004*, Louvain-la-Neuve, Belgium, pages 411–422.

Fleiss, Joseph L.; Cohen, Jacob; Everitt, B. S. (1969). Large sample standard errors of kappa and weighted kappa. *Psychological Bulletin*, **72**(5), 323–327.

Krenn, Brigitte; Evert, Stefan; Zinsmeister, Heike (2004). Determining intercoder agreement for a collocation identification task. In preparation.

See Also

`ucs.library`, the UCS/R tutorial ('tutorial.R' in the 'script/' subdirectory) and the UCS/Perl documentation.

VV *Variance of the Vocabulary Size of a LNRE Model (zm, fzm)*

Description

Computes the variance of the vocabulary size of a LNRE model (Baayen, 2001) at sample size N.

Usage

```
VV(model, N)
```

Arguments

`model`	an object of class `"zm"` or `"fzm"`, representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model
`N`	a vector of positive integers, representing sample sizes

Details

The variance $V[V(N)]$ is computed according to Baayen (2001, 120f). See the `EV` help page for some more information on the vocabulary size $V(N)$.

Value

a numeric vector of the same length as `N`

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

zm, fzm, VVm, EV, EVm

VVm *Variances of the Frequency Spectrum of a LNRE Model (zm, fzm)*

Description

Computes the variances of the frequency spectrum and conditional parameter distribution of a LNRE model (Baayen, 2001) at sample size N.

Usage

```
VVm(model, m, N, rho=1, relative=FALSE, lower=TRUE)
```

Arguments

model	an object of class "zm" or "fzm", representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model
m	a vector of positive integers, representing frequency ranks
N	a vector of positive integers, representing sample sizes; either m or N should be a single number
rho	a vector of numbers in the range $[0, 1]$. If length(rho) > 1, both m and N should be single numbers. See below for details.
relative	if TRUE, computes variances for the relative conditional parameter distribution (see below for details). May only be used when rho is specified.
lower	if rho is specified, controls whether variances are computed for the lower or for the upper conditional parameter distribution

Details

The variance $V[V_m(N)]$ is computed according to Baayen (2001, 120f).

When rho is specified, the variances of the conditional parameter distribution $V[V_{m,\rho}(N)]$ or the corresponding proportions $V[R_{m,\rho}(N)]$ are returned, depending on the value of relative. With lower=FALSE, computes variances for the upper conditional parameter distribution $V[V_{m,>\rho}(N)]$ or proportion $V[R_{m,>\rho}(N)]$. See Evert (2004, Ch. 4) for details.

The EVm help page provides more information about $V_m(N)$, $V_{m,\rho}(N)$, $R_{m,\rho}(N)$, $V_{m,>\rho}(N)$ and $R_{m,>\rho}(N)$.

Note that this function does *not* compute variances for the relative frequency spectrum ($V[V_m(N)/V(N)]$) or the ratio between consecutive spectrum elements ($V[V_{m+1}(N)/V_m(N)]$).

Value

a numeric vector of appropriate length (determined either by m, N, or rho)

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

zm, fzm, VVm, EV, EVm

`add.gams` *Annotate Data Set with GAM Scores (gam)*

Description

Annotates data set with GAM scores, possibly overwriting existing scores of a standard AM. Optionally, *jitter* annotated in the data set can be taken into account when computing the scores.

Usage

```
add.gams(ds, names, jitter=FALSE)
```

Arguments

`ds`	a UCS data set object
`name`	a character vector specifying the names of generalised association measures to be annotated in the data set
`add.jitter`	if `TRUE`, random jitter (which must be annotated in the data set) is added to the frequency signatures before computing GAM scores (see details below)

Details

The `add.gams` function uses the standard variable names for AM scores (e.g. `am.t.score` for the `t.score` measure), so that existing scores for the respective standard AMs in the data set will be overwritten. Rankings for the GAM scores can then be computed in the normal way using the `add.ranks` function.

With `jitter=TRUE`, a small amount of random jitter is added to the frequency signatures in order to avoid ties in the rankings and facilitate visualisation of the data set. The necessary jitter vectors have to be stored in special variables in the data set first, which is most easily achieved with the `add.jitter` function.

Value

a copy of the data set `ds` annotated with GAM scores for the specified measures

See Also

gam.score, gam.iso, builtin.gams, add.ranks, add.jitter

Examples

```
ds <- add.ranks(add.gams(ds, c("t.score", "chi.squared.corr")))

ds <- add.jitter(ds)
gam.names <- ds.find.am(ds)
gam.names <- gam.names[ is.builtin.gam(gam.names) ]
ds <- add.gams(ds, gam.names, jitter=TRUE)
ds <- add.ranks(ds, gam.names, randomise=FALSE, overwrite=TRUE)
```

`add.jitter` *Random Jitter for Frequency Signatures in Data Set (gam)*

Description

Add random *jitter* to the frequency signatures in a data set, in order to avoid ties in rankings according to GAM scores and to facilitate visualisation of the data set with eo and ebo plots. The `add.ebo` function is used to re-compute ebo-coordinates from the jittered frequency signatures.

Usage

```
add.jitter(ds, amount=0.5, overwrite=FALSE)

has.jitter(ds, fail=FALSE)

add.ebo(ds, jitter=FALSE)
```

Arguments

`ds`	a UCS data set object
`amount`	amount of jitter to be added; the jitter vector for each coordinate (`f, f1, f2`) has a uniform distribution over the range `[-amount, +amount]`
`overwrite`	if `TRUE`, overwrite existing jitter vectors in the data set
`fail`	if `TRUE`, abort with an error message unless the data set contains jitter vectors
`jitter`	if `TRUE`, use the jittered frequency signatures to compute ebo-coordinates (default: unjittered integer frequencies)

Details

The `add.jitter` function adds jitter vectors for the joint and marginal frequencies (`f, f1, f2`) to a data set, i.e. uniformly distributed random numbers in the range `[-amount, +amount]`. These vectors are stored in variables `x.jitter.f`, `x.jitter.f1` and `x.jitter.f2`, where they can be used by `add.ebo`, `add.gams` and other functions. `has.jitter` tests for the presence of these variables.

`add.ebo` computes ebo-coordinates from the frequency signatures and stores them in the standard variables `e, b, o`. Unlike the values computed with UCS/Perl tools, `add.ebo` uses jitter vectors in this computation when the option `jitter=TRUE` is passed.

Value

`add.jitter` and `add.ebo` return a copy of the data set `ds` with the request variables added. `has.jitter` returns `TRUE` if the jitter variables are present in `ds`, and `FALSE` otherwise.

See Also

`add.gams`, `gamma.nbest`

Examples

```
ds <- add.jitter(ds, amount=0.2)

ds <- add.ebo(ds, jitter=TRUE)  # recompute ebo coordinates with jitter
```

`add.ranks` *Compute Rankings for Annotated Association Measures (base)*

Description

Add rankings (with or without ties) for specified association measures to a data set object.

Usage

```
add.ranks(ds, keys=ds.find.am(ds), randomise=TRUE, overwrite=TRUE)
```

Arguments

`ds`	a UCS data set object
`keys`	a character vector giving the names of one or more association measures. When it is omitted, rankings are computed for all annotated measures.
`randomise`	if `TRUE`, ties are broken randomly (default). Otherwise, tied rows are assigned the same rank, which is the first free one (as in the Olympic Games). See below for prerequisites.
`overwrite`	if `TRUE`, existing rankings are overwritten (default). Otherwise, association measures for which ranks are already annotated are silently skipped. If you modify association scores within R, be sure to call `add.ranks` with `overwrite=TRUE`.

Details

Since `add.ranks` is based on the `order.by.am` function, the prerequisites are the same: the data set must contain association scores for the `random` measure if `randomise=TRUE` and an `id` variable if `randomise=FALSE`. See the `order.by.am` manpage for further information.

Value

Invisibly returns a copy of `ds` annotated with the requested rankings. The rankings are stored in variables `r.*`, where `*` stands for the name of an association measure (according to the UCS naming conventions, cf. the `am.key2var` manpage).

See Also

`order.by.am`, `am.key2var`, `ds.find.am`, `read.ds.gz`

Examples

```
## from the UCS/R tutorial
GLAW <- read.ds.gz("glaw.scores.ds.gz")
GLAW <- add.ranks(GLAW)

## combine into single command
GLAW <- add.ranks(read.ds.gz("glaw.scores.ds.gz"))
```

am.key2var *UCS Variable Names for Association Scores and Rankings (base)*

Description

These functions implement the UCS naming conventions for variables storing association scores and the corresponding ranking. `is.valid.key` checks whether a given string is valid as a name for an association measure. `am.key2var` translates a valid AM name into the corresponding variables (for scores or ranking), and `am.var2key` extracts the AM name from such a variable.

Usage

```
is.valid.key(key, warn=FALSE)

am.key2var(key, rank=FALSE)

am.var2key(var)
```

Arguments

`key`	a character vector, giving the names of one or more association measures
`var`	a character vector of variable names, which must be either association scores or rankings (but both types can be mixed in the vector)
`warn`	if `TRUE`, issues a warning if the vector `key` contains invalid AM names. All invalid entries are listed in the warning message.
`rank`	if `TRUE`, return names of the ranking variables corresponding to the specified association measures. otherwise, return names of variables for association scores.

Value

`is.valid.key` returns a logical vector, `am.var2key` returns a list of AM names ("keys"), and `am.key2var` returns a list of variable names (either for association scores or rankings, depending on the `rank` parameter).

See Also

`builtin.ams` for information about built-in association measures, and the `ucsfile` manpage in UCS/Perl for a description of the UCS naming conventions (enter the shell command `ucsdoc ucsfile`).

Examples

```
am.key2var(c("t.score", "MI"), rank=TRUE)
am.var2key(c("am.t.score", "r.MI"))
```

`binom.conf.interval`

Binomial Confidence Intervals

Description

Computes confidence intervals for the success probability of a binomial distribution efficiently. Unlike `binom.test`, this function can be applied to vectors.

Usage

```
binom.conf.interval(k, size, limit=c("lower","upper"),
                    conf.level=0.05, one.sided=FALSE)
```

Arguments

`k`	a vector of non-negative integers. Each element represents the number of successes out of `size` trials, i.e. the observed value of a random variable with binomial distribution.
`size`	a vector of positive integers. Each element represents the number of trials of a binomial distribution.
`limit`	if `"upper"`, the upper boundaries of the confidence intervals are returned. If `"lower"`, the lower boundaries are returned. Note that this works both for one-sided and for two-sided confidence intervals.
`conf.level`	the required confidence level, or rather the significance level of the corresponding binomial test (note that this behaviour differs from the built-in `binom.test` function). The default `conf.level=0.05` stands for 95% confidence.
`one.sided`	if `TRUE`, computes one-sided confidence interval (either lower or upper, depending on the value of `limit`). If `FALSE`, a two-sided confidence interval is computed (default).

Details

If `one.sided=TRUE`, the underlying test is one-sided (with alternative `"less"` or `"greater"`, depending on the `limit` parameter), and the non-trivial boundary of the confidence interval is returned.

If `one.sided=FALSE`, the underlying test is two-sided and the requested boundary of the two-sided confidence interval is returned. For efficiency reasons, the `binom.conf.interval` function cheats a little and computes one-sided confidence intervals with significance level `conf.level / 2`.

Value

A numeric vector with the requested boundary of confidence intervals for the unknown success probabilities of binomial variables.

See Also

`binom.test`

builtin.ams *UCS/Perl Built-in Association Measures (base)*

Description

`builtin.ams` returns a character vector listing the built-in association measures of the UCS/Perl system (including the standard add-on packages), `is.builtin.am` checks whether a specified measure belongs to this set, and `am.key2desc` returns a short description of the specified measure.

Usage

```
builtin.ams()

is.builtin.am(key)

am.key2desc(key)
```

Arguments

`key`	a character vector specifying the names of one or more association measures

Value

`builtin.ams` returns a character vector containing the names of all built-in association measures, `is.builtin.am` returns a logical vector, and `am.key2desc` returns a character vector with a short description of each of the measures in `key`.

See Also

The information provided by these functions is obtained from the UCS/Perl tool `ucs-list-am`. See the `ucsam` manpage in UCS/Perl for further information about built-in association measures (using the shell command `ucsdoc ucsam`).

Examples

```
print(builtin.ams())
am.key2desc("chi.squared.corr")
```

builtin.gams *Built-in Generalised Association Measures (gam)*

Description

List available GAMs (generalised association measures) that can be computed with functions such as `gam.score`, `add.gams` and `gam.iso`, or test whether a specific GAM is available. Additional GAMs can be defined with the `register.gam` function.

Usage

```
builtin.gams()

is.builtin.gam(names)

register.gam(name, equation, iso.equation=NULL)
```

Arguments

`names`	a character vector specifying the names of GAMs whose availability is tested
`name`	a single character string specifying the name of a GAM that is defined or re-defined
`equation`	a function that computes GAM scores from standard or ebo-coordinates (see below for details)
`iso.equation`	an optional function that computes iso-surfaces in standard or ebo-coordinates (see below for details)

Details

The names of built-in GAMs are identical to those of the corresponding standard AMs (e.g. `t.score` and `chi.squared.corr`).

The `equation` argument of `register.gam`, i.e. the equation defining a new GAM), must be a function with the signature `(o, e, b, f, f1, f2, N)`. This function can computes GAM scores either from the ebo-coordinates `e, b, o` or from the standard coordinates `f, f1, f2, N`. It is always invoked with all seven arguments, which are guaranteed to be vectors of the same length, and must return a vector of corresponding GAM scores.

When an explicit equation for iso-surfaces $\{g = \gamma\}$ exists, it can be made available through the optional argument `iso.equation`, which expects a function with the signature `(gamma, e, b, f1, f2, N)`. Again, all six arguments are guaranteed to be vectors of the same length, and the function must return the corresponding o (or f) coordinates that satisfy the condition $g(o, e, b) = \gamma$ (or $g(f, f_1, f_2, N) = \gamma$). When the `iso.equation` function is available for a GAM, it will be used by `gam.iso` for greater speed and accuracy. Otherwise, the iso surface is determined by a binary search algorithm (which has a unique solution for any semi-sound GAM).

The signatures of the `equation` and `iso.equation` functions are checked by `register.gam`, which will abort with an error message if they are not correct.

Value

`builtin.gams` returns a character vector listing the names of available GAMs. `is.builtin.gam` returns a logical vector indicating which of the GAMs in the vector `names` are available.

See Also

`builtin.ams`, `gam.score`, `add.gams`, `gam.iso`, `gam.helpers`

Examples

```
print(builtin.gams())

all(is.builtin.gam(c("MI", "t.score", "chi.squared")))
```

```
register.gam("MI5",
             eq  = function (o, e, b, f, f1, f2, N) { log10(o^5 / e) },
             iso = function (gamma, e, b, f1, f2, N) { 10^(gamma/5) * e^(1/5) })
```

`ds.find.am` *List Association Scores and Rankings in Data Set (base)*

Description

`am.in.ds` tests whether a specified association measure is annotated in a data set, `ds.find.am` lists all annotated association measures, and `ds.match.am` searches the data set for AMs whose names may be abbreviated to a unique prefix. All three functions look either for association scores or for rankings.

Usage

```
am.in.ds(ds, keys, rank=FALSE, fail=FALSE)

ds.find.am(ds, rank=FALSE)

ds.match.am(ds, abbrevs, rank=FALSE)
```

Arguments

`ds`	a UCS data set, read from a data set file with the `read.ds.gz` function
`keys`	a character vector of AM names
`abbrevs`	a character vector of AM names, each of which may be abbreviated to a unique prefix (within the data set)
`rank`	if `TRUE`, the functions look for annotated rankings; otherwise, they look for annotated association scores (default)
`fail`	if `TRUE`, the function aborts with an error message unless all specified AMs are annotated in the data set

Details

If any of the `abbrevs` do not have a unique match in the data set, `ds.match.am` aborts with an error message (listing all strings that failed to match uniquely).

Value

`am.in.ds` returns a logical vector of the same length as `keys`. `ds.find.am` and `ds.match.am` return a character vector containing the names of the annotated association measures.

See Also

`read.ds.gz`, `am.var2key`

Examples

```
GLAW <- read.ds.gz("glaw.scores.ds.gz")
print(ds.find.am(GLAW))
```

`eo.iso` *Draw Iso-Line of a GAM in the (e,o) Plane (eo)*

Description

Draw an iso-line of a generalised association measure (GAM) in the (e,o) plane, either for a specified cutoff threshold γ or an n-best iso-line for a given data set `ds`. Optionally, the corresponding acceptance region can be shaded or filled with solid colour.

Usage

```
eo.iso(gam, gamma=0, b=1, N=1e6, n.best=NULL, ds=NULL,
       style=1, fill=solid, solid=FALSE,
       steps=eo.par("steps"), jitter=eo.par("jitter"), bw=bw,
       col=eo.par("col"), lty=eo.par("lty"), lwd=eo.par("lwd"),
       angle=eo.par("angle"), density=eo.par("density"),
       solid.col=eo.par("solid"))
```

Arguments

`gam`	a character string giving the name of a generalised association measure (GAM). Use the function `builtin.gams` from the `gam` module to obtain a list of available GAMs.
`gamma`	a cutoff threshold that determines the iso-line to be drawn (by the implicit equation $\{g = \gamma\}$). Use the `n.best` and `ds` parameters instead of `gamma` in order to obtain an n-best iso-line for the data set `ds`.
`b, N`	optional balance (`b`) and sample size (`N`) parameters for GAMs that are not central or size-invariant, respectively. The default `b=1` yields the centralised version of a non-central GAM (for details, see Evert 2004, Sec. 3.3)
`n.best, ds`	When these parameters are specified, the cutoff threshold `gamma` will automatically be determined so as to yield an n-best acceptance region for the data set `ds`.
`jitter`	If `TRUE`, use jittered coordinates for computing the n-best cutoff threshold (see above). In this case, the data set has to be annotated with the `add.jitter` function first.
`style`	an integer specifying the style (colour, line type and width) in which iso-lines will be drawn. The number of styles available depends on the global parameter settings (`eo.par`). The "factory settings" define 5 different styles for iso-lines.
`fill`	If `TRUE`, fill in the acceptance region bounded by the given iso-line with shading lines, according to the chosen `style` and `bw mode`. See `eo.par` for details on shading styles.
`solid`	If `TRUE`, fill the acceptance region with solid colour rather than shading lines, also according to the chosen `style` and `bw` mode. Setting `solid=TRUE` implies `fill=TRUE`.
`steps`	an integer specifying how many equidistant steps are used for drawing iso-lines. The default value is set with `eo.par`.

`bw` If `TRUE`, the iso-lines are drawn in B/W mode, otherwise in colour mode. This parameter defaults to the state specified with the initial `eo.setup` call, but can be overridden manually.

`col, lty, lwd` can be used to override the default style parameters for iso-lines, which are determined automatically from the global settings (`eo.par`) according to the selected `style` and `bw` mode.

`angle, density` can be used to override the default style parameters for shaded acceptance region, which are determined automatically from the global settings (`eo.par`) according to the selected `style` and `bw` mode.

`solid.col` can be used to override the default colour for solid filled acceptance regions, which is determined automatically from the global settings (`eo.par`) according to the selected `style` and `bw` mode.

Details

See the `eo.setup` help page for a description of the general procedure used to create (e,o) plots. This help page also has links to other (e,o) plotting functions. The "factory setting" styles are described on the `eo.par` help page.

The cutoff threshold γ can either be specified explicitly (with the `gamma` parameter) or implicitly as an n-best threshold (with `n.best`, `ds`, and optional `jitter`). The latter method produces the same result as

```
gam.iso(gam, gamma=gamma.nbest(ds, gam, n.best, jitter), ...)
```

Visualisation by (e,o) iso-lines is most suitable for GAMs that are both central and size-invariant (see Evert 2004, Sec. 3.3). For non-central measures, the `eo.iso` function uses a balance value of $b = 1$, yielding a centralised version of the GAM. Note that many non-central GAMs (especially those based on statistical tests, such as `log.likelihood` and `chi.squared`) have only a weak dependency on the balance b, so that their centralised iso-surfaces (i.e. extrusions of the iso-lines along the b-axis) are very similar to the original iso-surfaces. Other GAMs (most notably `Dice` and similar measures) are highly dependent on b, though. For measures that are not size-invariant, the sample size is arbitrarily set to $N = 10^6$, which is in a realistic range for real-life data sets. You may wish to modify the default value in order to match a data set shown in the plot (this is *not* done automatically when the `ds` parameter is specified), or to demonstrate the dependency of iso-lines on N.

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`eo.par`, `eo.setup`, `eo.iso.diff`

Examples

```
## an example can be found on the "eo.setup" help page
```

`eo.iso.diff` *Highlight Differences between Two Acceptance Regions in the (e,o) Plane (eo)*

Description

Compare the acceptance regions of two GAMs by shading the two difference sets (cf. Evert 2004, Sec. 5.2.2) in different fill styles. This function should be followed by two `eo.iso` calls to draw the iso-lines bounding the difference regions.

Usage

```
eo.iso.diff(gam1, gam2, gamma1=0, gamma2=0, b=1, N=1e6,
            n.best1=NULL, n.best2=NULL, ds=NULL,
            style1=4, style2=5, solid=FALSE, bw=bw,
            steps=eo.par("steps"), jitter=eo.par("jitter"),
            col1=eo.par("col"), angle1=eo.par("angle"),
            density1=eo.par("density"), solid.col1=eo.par("solid"),
            col2=eo.par("col"), angle2=eo.par("angle"),
            density2=eo.par("density"), solid.col2=eo.par("solid"))
```

Arguments

`gam1, gam2` character strings giving the names of two generalised association measures (GAMs). Use the function `builtin.gams` from the `gam` module to obtain a list of available GAMs.

`gamma2, gamma2` cutoff thresholds that determines the two acceptance regions ($\{g_1 = \gamma_1\}$ and $\{g_1 = \gamma_1\}$) to be compared. You can use `n.best` and `ds` parameters (see below) to compute n-best thresholds automatically.

`b, N` optional balance (`b`) and sample size (`N`) parameters for GAMs that are not central or size-invariant, respectively. The default `b=1` yields the centralised version of a non-central GAM (for details, see Evert 2004, Sec. 3.3). Note that the same values are used for both GAMs.

`n.best1, n.best2, ds` When `n.best1` is specified, the cutoff threshold `gamma1` will automatically be determined so as to yield an n-best acceptance region for the data set `ds`. In the same way, `n.best2` computes `gamma2` as an n-best acceptance threshold. Note that the data set `ds` is used for both n-best thresholds.

`jitter` If `TRUE`, use jittered coordinates for computing n-best cutoff thresholds (see above). In this case, the data set has to be annotated with the `add.jitter` function first.

`style1, style2` integer values specifying fill styles for the two difference regions. `style1` is used for the region D_1 of the (e,o) plane accepted by `gam1` but not `gam2`, and `style2` for the region D_2 accepted by `gam2` but not `gam1`. Style parameters include the colour, angle and density of shading lines, or the solid fill colour if `solid=TRUE`. See the `eo.par` help page for more information about available fill styles.

`solid` If `TRUE`, fill the difference regions with solid colour rather than shading lines, also according to the chosen `styles` and `bw` mode.

`bw` If `TRUE`, the regions are drawn in B/W mode, otherwise in colour mode. This parameter defaults to the state specified with the initial `eo.setup` call, but can be overridden manually.

`steps` an integer specifying how many equidistant steps are used for the (combined) boundaries of the difference regions. The default value is set with `eo.par`.

`col1, col2` can be used to override the default colours for shading lines, which are determined automatically from the global settings (`eo.par`) according to the selected `styles` and `bw` mode.

`angle1, angle2` can be used to override the default angles of shading lines, which are determined automatically from the global settings (`eo.par`) according to the selected `styles` and `bw` mode.

`density1, density2` can be used to override the default densities of shading lines, which are determined automatically from the global settings (`eo.par`) according to the selected `styles` and `bw` mode.

`solid.col1, solid.col2` can be used to override the default solid fill colours (with `solid=TRUE`), which are determined automatically from the global settings (`eo.par`) according to the selected `styles` and `bw` mode.

Details

See the `eo.setup` help page for a description of the general procedure used to create (e,o) plots. This help page also has links to other (e,o) plotting functions. The "factory setting" styles are described on the `eo.par` help page.

See the `eo.iso` help page for details about iso-lines, acceptance regions and n-best cutoff thresholds.

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`eo.par`, `eo.setup`, `eo.iso`

Examples

```
## setup code (see "eo.setup" example for a detailed explanation)
ucs.library("eo")
ds <- add.jitter(read.ds.gz("dickens.ds.gz"))
select <- rbinom(nrow(ds), 1, .1) == 1
ds <- ds[select,]

## comparison of 300-best acceptance regions for Poisson and MI measures
eo.setup(xlim=c(-3,2), ylim=c(0,2), aspect=FALSE)
eo.iso.diff("Poisson.pv", "MI", n.best1=300, n.best2=300, ds=ds, solid=TRUE, jitter=TRUE)
eo.points(ds, style=1, jitter=TRUE)
```

```
eo.iso("Poisson.pv", n.best=300, ds=ds, style=4)
eo.iso("MI", n.best=300, ds=ds, style=5)
eo.legend.diff(3, c("Poisson+ / MI-","Poisson- / MI+"), solid=TRUE)
eo.close()
```

`eo.legend` *Draw Legend Box for Point Cloud or Iso-Lines (eo)*

Description

Draw a legend box in one of the corners of the active (e,o) plot, showing labels for one or more styles of data set points, iso-lines or shaded/filled acceptance regions.

Usage

```
eo.legend.points(corner, legend, styles, bw=bw, cex.mul=2.5, ...)

eo.legend.iso(corner, legend, styles, bw=bw, fill=solid, solid=FALSE,
              lw.add=0, density.mul=2, ...)

eo.legend.diff(corner, legend, style1=4, style2=5,
               bw=bw, solid=FALSE, density.mul=2, ...)
```

Arguments

`corner`	an integer specifying the corner of the plot where the legend box will be drawn (`1` = top left, `2` = top right, `3` = bottom right, `4` = bottom left)
`legend`	a character vector specifying labels for the legend box. For the `eo.legend.diff` function, it must have length 2 (labels for the difference regions D_1 and D_2).
`styles`	an integer vector specifying display styles for the items in the legend box (see the `eo.par` help page for more information about display styles). Note that `styles` must have exactly the same length as `legend`
`style1, style2`	display styles for the first and second difference region (D_1 and D_2). The defaults are set to match those of `eo.iso.diff`.
`bw`	If `TRUE`, the points, lines or shading/colour boxes in the legend are drawn in B/W mode; otherwise, they are drawn in colour mode. This parameter defaults to the state specified with the initial `eo.setup` call, but can be overridden manually.
`fill`	If `TRUE`, show the shadings of acceptance regions instead of iso-line styles in the legend.
`solid`	If `TRUE`, show solid colours instead of shadings for acceptance regions in the legend. Setting `solid=TRUE` implies `fill=TRUE`.
`cex.mul`	numeric factor by which plot symbols are scaled in the legend box (with respect to their size in the plot)
`lw.add`	numeric value added to line widths in the legend box. Only needed when widths of iso-lines are too thin to be clearly visible in the legend box.

density.mul	numeric factor by which the density of shading lines is multiplied in the legend box in order to improve visibility of the shading style
...	Any additional parameters are passed through to the legend function used to draw the legend box.

Details

See the eo.setup help page for a description of the general procedure used to create (e,o) plots. This help page also has links to other (e,o) plotting functions. The "factory setting" styles are described on the eo.par help page.

eo.legend.points displays a legend box for point clouds plotted with eo.points; eo.legend.iso a legend box for iso-lines or acceptance regions drawn with eo.iso; and eo.legend.diff a legend box for differences between two acceptance regions that have been highlighted with eo.iso.diff (this is just a convenience wrapper around eo.legend.iso).

Note that legend boxes can only be created for the default styles set with eo.par since it is not possible to override the style parameters manually.

See Also

eo.par, eo.setup, eo.points, eo.iso, eo.iso.diff

Examples

```
## an example can be found on the "eo.setup" help page
```

---

eo.mark *Mark Individual Pair Typess in Point Cloud (eo)*

Description

Mark individual pair types from a data set in a point cloud plotted with the eo.points function.

Usage

```
eo.mark(ds, select, style=1, bw=bw, cex=1.5, lwd=3,
        jitter=eo.par("jitter"))
```

Arguments

ds	a data set containing pair types that have been plotted as a point cloud, some or all of which will be marked
select	an expression that will be evaluated on the data set ds to determine the pair types that will be marked. In order to mark the point representing the word pair *black box*, e.g., specify select=(l1 == "black" & l2 == "box").
style	an integer specifying the style from which the colour of the markers is taken. Note that the symbol (a thick ring) and its size are hard-coded in the function and cannot be changed globally.

bw	If TRUE, the markers are drawn in B/W mode, otherwise in colour mode. This parameter only affects the colour of the marker rings. It defaults to the state specified with the initial eo.setup call, but can be overridden manually.
cex, lwd	size and thickness of the marker rings. The default values are suitable for the "factory setting" styles used for data set points (see eo.points).
jitter	If TRUE, the coordinates of pair types are jittered for the plot. This parameter must have the same value as in the eo.points call that was used to plot the point cloud, otherwise marker placement will be incorrect. When jitter=TRUE, the data set has to be annotated with the add.jitter function first. The default value is set with eo.par.

Details

See the eo.setup help page for a description of the general procedure used to create (e,o) plots. This help page also has links to other (e,o) plotting functions. The "factory setting" styles are described on the eo.par help page.

See Also

eo.par, eo.setup, eo.points

---

eo.par *Graphics Parameters for (e,o) Plots (eo)*

Description

Set default graphics parameters for (e,o) plots, similar to ucs.par in the plots module and par for general graphics parameters. Parameter values can be set by specifying them as arguments in name=value form, or by passing a single list of named values. The current values can be queried by giving their names as character strings.

Usage

```
eo.par(...)

.eo.PAR
```

Arguments

...	either character strings (or vectors) specifying the names of parameters to be queried, or parameters to be set in name=value form, or a single list of named values. Valid parameter names are described below.

Details

The current default parameters are stored in the global variable `.eo.PAR`. They can be queried by giving their names as one or more character vectors to `eo.par`. `eo.par()` (no arguments) returns all `eo` graphics parameters.

Parameters are set by specifying their names and the new values as `name=value` pairs. Such a list can also be passed as a single argument to `eo.par`, which is typically used to restore previous parameter values (that have been saved in a list variable).

In order to restore the "factory settings", reload the module with the command `ucs.library("eo", reload=TRUE)`.

Value

When parameters are set, their former values are returned in an invisible named list. Such a list can be passed as a single argument to `eo.par` to restore the previous settings.

When a single parameter is queried, its value is returned directly. When two or more parameters are queried, the result is a named list.

Note the inconsistency, which is the same as for `par`: setting one parameter returns a list, but querying one parameter returns a vector (or a scalar, i.e. a vector of length 1).

Graphics Parameters for (e,o) Plots

`bw` If `TRUE`, (e,o) plots are created in B/W mode by default.

`xlim, ylim` Integer vectors of length 2, specifying default ranges for the e-axis (`xlim`) and o-axis (`ylim`) in orders of magnitude (i.e., base 10 logarithms: -2 corresponds to .01, 0 corresponds to 1, and 3 corresponds to 1000). When the default values are not set, every call to the `eo.setup` function must either specify `xlim` and `ylim` values or a data set, from which suitable ranges are computed.

`aspect` If `TRUE`, an aspect ratio of 1:1 is enforced for every (e,o) plot, i.e. the axis ranges are extended as necessary (assuming a square plotting region). The factory setting is `TRUE`.

`log.marks` If `TRUE`, tick marks on the axes are labelled in logarithmic units, i.e. orders of magnitude. Otherwise, absolute numbers are used. The factory setting is `FALSE`. (Note that (e,o) plots are always drawn in logarithmic scale.)

`steps` An integer specifying the number of equidistant steps used for drawing iso-lines. The factory setting is 100.

`jitter` If `TRUE`, always uses jittered coordinates for plotting data sets and computing n-best thresholds. Note that all data sets must be annotated with the `add.jitter` function first. The factory setting is `FALSE`.

`cex` Overall character expansion factor (for tick marks, axis labels and legends). The factory setting is `1.3`.

`col` A character of integer vector specifying line colours for the different styles of iso-lines in colour mode (see the `par` manpage for details on colour specification). Values are recycled to match the length of the `lty` and `lwd` parameters when necessary. The factory setting defines 5 styles in black, blue, red, magenta and cyan.

`lty` A character or integer vector specifying line types for the different styles of iso-lines in colour mode (see the `par` manpage for details). Values are recycled to match the length of the `col` and `lwd` parameters when necessary.

`lwd` A numeric vector specifying line widths for the different styles of iso-lines in colour mode. Values are recycled to match the length of the `col` and `lty` parameters when necessary.

angle, density Numeric vectors specifying the angle and density of shading lines when the acceptance region bounded by a given iso-line is filled. These vectors should supportas many styles as `col`, `lty` and `lwd` above. Details on shading lines can be found on the `polygon` help page.

solid A character or integer vector specifying background colours for the different styles of iso-lines when the acceptance region is filled with solid colour (rather than shading lines).

bw.col, bw.lty, bw.lwd Colour, line type and line width for iso-lines in B/W mode (corresponding to `col`, `lty` and `lwd` in colour mode). The factory setting defines 5 styles with solid, dashed, grey, dotted and dark grey dot-dash lines.

bw.angle, bw.density, bw.solid Angle and density of shading lines, as well as solid colour, for filled acceptance regions in B/W mode (corresponding to `angle`, `density` and `solid` in colour mode)

pt.pch A character or integer vector specifying plot symbols for the different styles of data set points in colour mode (see the `points` help page for a full list of available plot symbols). Values are recycled to match the length of the `pt.cex` and `pt.col` parameters when necessary. The factory setting defines 5 styles with black, green, red, yellow and orange dots.

pt.cex A numeric vector specifying character expansion factors for the different styles of data set points in colour mode. Values are recycled to match the length of the `pt.pch` and `pt.col` parameters when necessary.

pt.col A character or integer vector specifying colours for the different styles of data set points in colour mode (see the `par` help page for details on colour specification). Values are recycled to match the length of the `pt.pch` and `pt.cex` parameters when necessary.

bw.pt.pch, bw.pt.cex, bw.pt.col Plot symbol, character expansion and colour for data set points in B/W mode (corresponding to `pt.pch`, `pt.cex` and `pt.col` in colour mode). The factory setting defines 5 styles with black dots, circles, + crosses, triangles and x crosses.

See Also

`eo.setup`, `eo.iso`, `eo.iso.diff`, `eo.points`, `eo.legend`, `ucs.par`, `par`

Examples

```
print(names(ucs.eo()))          # list available parameters

eo.par("col", "lty", "lwd")     # the default styles for iso-lines
eo.par(c("col", "lty", "lwd"))  # works as well

## temporary changes to graphics paramters:
par.save <- eo.par(bw=TRUE, steps=200)
## (e,o) plots use the modified parameters here
eo.par(par.save)                # restore previous values

ucs.library("eo", reload=TRUE)  # reload module for factory defaults
```

`eo.points` *Draw Data Set as Point Cloud in (e,o) Plane (eo)*

Description

Plot (selected) pair types from a data set as a point cloud in the (e,o) plane. Points can be drawn in any of the styles defined in the global defaults (`eo.par`), as determined by the `style` parameter.

Usage

```
eo.points(ds, style=1, select=NULL, bw=bw, jitter=eo.par("jitter"),
          pch=par("pt.pch"), cex=par("pt.cex"), col=par("pt.col"), ...)
```

Arguments

`ds`	a data set containing the pair types to be plotted as a point cloud
`style`	an integer specifying the style (shape, size and colour) in which points will be drawn. The number of styles available depends on the global parameter settings (`eo.par`). The "factory settings" define 5 different styles for points.
`select`	an optional expression, which is evaluated on the data set `ds` to select a subset of the pair types for plotting (e.g. `select=(f <= 10 & b.TP)` to display pair types with joint frequency $f \leq 10$ that are marked as true positives).
`bw`	If `TRUE`, the points are drawn in B/W mode, otherwise in colour mode. This parameter defaults to the state specified with the initial `eo.setup` call, but can be overridden manually.
`jitter`	If `TRUE`, the coordinates of pair types are jittered for the plot, i.e. a small random displacement is added to each point so that the point cloud has a more homogeneous appearance. In order to use this option, the data set has to be annotated with the `add.jitter` function first. The default value is set with `eo.par`.
`pch, cex, col`	The style parameters for points are determined automatically from the global settings (`eo.par`), according to the selected `style` and `bw` mode. They can be overridden by specifying explicit values in the function call.
`...`	Any additional parameters are passed through to the `points` function that draws the point cloud.

Details

See the `eo.setup` help page for a description of the general procedure used to create (e,o) plots. This help page also has links to other (e,o) plotting functions. The "factory setting" styles are described on the `eo.par` help page.

See Also

`eo.par`, `eo.setup`

Examples

```
## an example can be found on the "eo.setup" help page
```

`eo.setup` *Initialise and Finalise an (e,o) Plot (eo)*

Description

`eo.setup` initialises a new (e,o) plot window, which can then be drawn into with calls to `eo.iso`, `eo.points` and similar functions. The plot has to be finalised with `eo.close` before a new plot can be generated.

A detailed explanation of (e,o) plots and their interpretation can be found in Section 3.3 of Evert (2004).

Usage

```
eo.setup(xlim=eo.par("xlim"), ylim=eo.par("ylim"), ds=NULL,
         bw=eo.par("bw"), file=NULL,
         aspect=eo.par("aspect"), log.marks=eo.par("log.marks"),
         cex=eo.par("cex"), ...)

eo.close()
```

Arguments

`xlim, ylim`	integer vectors of length 2, specifying ranges for the e-axis (`xlim`) and o-axis (`ylim`) in orders of magnitude (i.e., base 10 logarithms: -2 corresponds to .01, 0 corresponds to 1, and 3 corresponds to 1000). If `xlim` and `ylim` are not given and no default values have been set with `eo.par`, the `ds` parameter has to be specified. Note that (e,o) plots are always drawn in logarithmic scale.
`ds`	A data set from which suitable ranges for the e-axis and o-axis are computed. The automatically determined values are overridden by explicit `xlim` and `ylim` parameters.
`bw`	If `TRUE`, the (e,o) plot is drawn in B/W mode, otherwise in colour mode. The default value is set with `eo.par`.
`file`	a character string giving the name of a PostScript file. If specified, the (e,o) plot is saved to `file` in EPS format rather than displayed on screen. Note that this file will only be written after `eo.close` has been called.
`aspect`	If `TRUE`, an aspect ratio of 1:1 is enforced by extending the axis ranges as necessary (assuming that the plotting region is square). The default value is set with `eo.par`.
`log.marks`	If `TRUE`, tick marks on the axes are labelled in logarithmic units, i.e. orders of magnitude. Otherwise, absolute numbers are used. The default value is set with `eo.par`. (Recall that (e,o) plots are always drawn in logarithmic scale.)

`cex`	overall character expansion factor (for tick marks, axis labels and legends). The default value is set with `eo.par`.
`...`	Any additional parameters are passed through to the `plot` function used to set up the plot region and axes.

Details

An (e,o) plot is typically created in four stages:

- Set up the plot with `eo.setup`, defining suitable ranges for the e-axis. These ranges and some other state information (e.g. whether the plot is drawn in colour or B/W mode) are recorded in the global variable `.eo.STATE`.
- Draw data sets as point clouds with `eo.points` and iso-lines for GAMs with `eo.iso`. Differences between two acceptance regions can be highlighted with `eo.iso.diff`. The `eo.mark` function can be used to mark individual points with circles.
- Draw legend boxes in the corners of the plot with `eo.legend.points`, `eo.legend.iso` and `eo.legend.diff`.
- Finalise the plot with `eo.close`. When a `file` argument has been specified in the `eo.setup` call, the plot will be saved to a PostScript file at this stage.

Default values for `xlim`, `ylim`, `bw`, `aspect`, `log.marks` and `cex` can be set with the `eo.par` function. See the `eo.par` help page for "factory settings" of these parameters, as well as default line and point styles in colour and B/W mode.

Note that (e,o) plots are always drawn in logarithmic scale and tick marks are shown for orders of magnitude (full powers of ten). The `log.marks` parameter only determines whether the labels on these tick marks show linear (.1, 1, 10, 100, ...) or logarithmic (-1, 0, 1, 2, ...) values.

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`eo.par`, `eo.points`, `eo.iso`, `eo.iso.diff`, `eo.mark`, `eo.legend`

Examples

```
ucs.library("eo")

## load data set file, add jitter, and reduce to random 10
ds <- add.jitter(read.ds.gz("dickens.ds.gz"))
select <- rbinom(nrow(ds), 1, .1) == 1
ds <- ds[select,]

## 1) set up new (e,o) plot with suitable axis ranges
eo.setup(ds=ds)  # note that y axis is extend to enforce 1:1 aspect

## 2) add data set as point cloud and three iso-lines
eo.points(ds, style=5, jitter=TRUE)
eo.iso("Poisson.pv", 3, style=1)  # p-value = 1e-3
eo.iso("z.score", 3.09, style=2)  # corresponding one-sided z-score
```

```
eo.iso("t.score", 3.09, style=3)  # same as t-score with df=Inf

## 3) add legend boxes in top right (2) and bottom right (3) corner
eo.legend.points(2, "pair type", 5)
eo.legend.iso(3, c("Poisson", "z-score", "t-score"), 1:3)

## 4) finalise the (e,o) plot
eo.close()
```

`evaluation.file` *Evaluation Graphs for Association Measures (plots)*

Description

The `evaluation.plot` function is often invoked twice with the same parameter settings, once for on-screen display, and once for saving the plot to a PostScript file. `evaluation.file` automates this process, automatically switching between colour mode for the screen version and B/W mode for the PostScript version.

Usage

```
evaluation.file(ds, keys, file, bw=NULL, ...)
```

Arguments

`ds`	a UCS data set object (passed to `evaluation.plot`)
`keys`	a character vector specifying the names of association measures to be evaluated (passed to `evaluation.plot`)
`file`	a character string giving the name of a file to which the PostScript version of the plot will be saved
`bw`	if `TRUE`, both versions will be in B/W; if `FALSE`, both versions will be in colour. If unspecified, `evaluation.file` switches automatically from colour mode (for the screen version) to B/W mode (for the PostScript file), which is the most common use.

Details

PostScript versions can be suppressed by setting

```
ucs.par(do.file=FALSE)
```

In this case, `evaluation.file` will only draw the screen versions of the graphs, which is convenient when experimenting and while fine-tuning the plots.

See Also

`evaluation.plot`, `ucs.par`, and the tutorial script 'tutorial.R' in the 'script/' directory.

`evaluation.plot` *Evaluation Graphs for Association Measures (plots)*

Description

An implementation of evaluation graphs for the empirical evaluation of association measures in terms of precision and recall, as described in (Evert, 2004, Ch. 5). Graphs of precision, recall and local precision for n-best lists, as well as precision-by-recall graphs are all provided by a single function `evaluation.plot`.

Usage

```
evaluation.plot(ds, keys, tp=ds$b.TP,
                x.min=0, x.max=100, y.min=0, y.max=100,
                x.axis=c("n.best", "proportion", "recall"),
                y.axis=c("precision", "local.precision", "recall"),
                n.first=ucs.par("n.first"), n.step=ucs.par("n.step"),
                cut=NULL, window=400,
                show.baseline=TRUE, show.nbest=NULL, show.npair=NULL,
                conf=FALSE, conf.am=NULL, conf.am2=NULL,
                test=FALSE, test.am1=NULL, test.am2=NULL,
                test.step=ucs.par("test.step"), test.relevant=0,
                usercode=NULL,
                file=NULL, aspect=1, plot.width=6, plot.height=6,
                cex=ucs.par("cex"), lex=ucs.par("lex"), bw=FALSE,
                legend=NULL, bottom.legend=FALSE,
                title=NULL, ...)
```

Arguments

`ds`	a UCS data set object, read in from a data set file with the `read.ds.gz` function. `ds` must contain rankings for the association measures listed in the `keys` parameter (use `add.ranks` to add such rankings to a data set object).
`keys`	a character vector naming up to 10 association measures to be evaluated. Each name may be abbreviated to prefix that must be unique within the measures annotated in `ds`. Use the `ds.find.am` function to obtain a list of measures annotated in the data set, and see the `ucsam` manpage in UCS/Perl for detailed information about the association measures supported by the UCS system (with the shell command `ucsdoc ucsam`).
`tp`	a logical vector indicating true positives, parallel to the rows of the data set `ds`. If `tp` is not specified, the data set must contain a variable named `b.TP` which is used instead.
`x.min, x.max`	the limits of the x-axis in the plot, used to "zoom in" to an interesting region. The interpretation of the values depends on the `x.axis` parameter below. For `x.axis="n.best"` (the default case), `x.min` and `x.max` refer to n-best lists. Otherwise, they refer to percentages ranging from 0 to 100. By default, the full data set is shown.
`y.min, y.max`	the limits of the y-axis in the plot, used to "zoom in" to an interesting region. The values are always interpreted as percentages, ranging from 0 to 100. By default, `y.max` is fitted to the evaluation graphs (unless `y.axis="recall"`, where `y.max` is always set to 100).

`x.axis` select variable shown on x-axis. Available choices are the n-best list size n (`"n.best"`, the default), the same as a proportion of the full data set (`"proportion"`), and the recall as a percentage (`"recall"`). The latter produces precision-by-recall graphs. Unless you are silly enough to specify `y.axis="recall"` at the same time, that is.

`y.axis` select variable shown on x-axis. Available choices are the precision (`"precision"`, the default), an estimate for local precision (`"local.precision"`, see details below), and the recall (`"recall"`). All three variables are shown as percentages ranging from 0 to 100.

`n.first` the smallest n-best list to be evaluated. Shorter n-best lists typically lead to highly unstable evaluation graphs. The standard setting is 100, but a higher value may be necessary for random sample evaluation (see details below). If `n.first` is not specified, the default supplied by `ucs.par` is used.

`n.step` the step width for n-best lists in the evaluation graphs. Initially, precision and recall are computed for all n-best lists, but only every `n.step`-th one is plotted, yielding graphs that look less jagged and reducing the size of generated PostScript files (see the `file` parameter below). If `n.step` is not specified, the default supplied by `ucs.par` is used.

`cut` for each association measure, pretend that the data set consists only of the `cut` highest-ranked candidates according to this measure. This trick can be used to perform an evaluation of n-best lists without having to annotate the full data set. The candidates from all relevant n-best lists are combined into a single data set file and `cut` is set to n.

`window` number of candidates to consider when estimating local precision (default: 400), i.e. with the option `y.axis="local"`. Values below 400 or above 1000 are rarely useful. See below for details.

`show.baseline` if `TRUE`, show baseline precision as dotted horizontal line with label (this is the default). Not available when `y.axis="recall"`.

`show.nbest` integer vector of n-best lists that will be indicated as thin vertical lines in the plot. When `x.axis="recall"`, the n-best lists are shown as diagonal lines.

`show.npair` when `x.axis="proportion"`, the total number of candidates in `ds` is shown in the x-axis label. Set `show.npair=NULL` to suppress this, or set it to an integer value in order to lie about the number of candidates (rarely useful).

`conf` if `TRUE`, confidence intervals are shown as coloured or shaded regions around one or two precision graphs. In this case, the parameter `conf.am` must also be specified. Alternatively, `conf` can be set to a number indicating the significance level to be used for the confidence intervals (default: 0.05, corresponding to 95% confidence). See below for details. Note that `conf` is only available when `y.axis="precision"`.

`conf.am` name of the association measure for which confidence intervals are displayed (may be abbreviated to a prefix that is unique within `keys`)

`conf.am2` optional second association measure, for which confidence intervals will also be shown

`test` if `TRUE`, significance tests are carried out for the differences between the evaluation results of two association measures, given as `test.am1` and `test.am2` below. Alternatively, `test` can be set to a number indicating the significance level to be used for the tests (default: 0.05). n-best lists where the result difference is significant are indicated by arrows between the respective evaluation

graphs (when x.axis="recall") or by coloured triangles (otherwise). See details below. Note that test is *not* available when y.axsis="local".

test.am1 — the first association measure for significance tests (may be abbreviated to a prefix that is unique within keys). Usually, this is the measure that achieves better performance (but tests are always two-sided).

test.am2 — the second association measure for significance tests (may be abbreviated to a prefix that is unique within keys)

test.step — the step width for n-best lists where significance tests are carried out, as a multiple of n.step. The standard setting is 10 since the significance tests are based on the computationally expensive fisher.test functio and since the triangles or arrows shown in the plot are fairly large. If test.step is not specified, the default supplied by ucs.par is used.

test.relevant — a positive number, indicating the estimated precision differences that are considered "relevant" and that are marked by dark triangles or arrows in the plot. See below for details.

usercode — a callback function that is invoked when the plot has been completed, but before the legend box is drawn. This feature is mainly used to add something to a plot that is written to a PostScript file. The usercode function is invoked with parameters region=c(x.min,x.max,y.min,y.max) and pr, a list of precision/recall tables (as returned by precision.recall) for each of the measures in keys.

file — a character string giving the name of a PostScript file. If specified, the evaluation plot will be saved to file rather than displayed on screen. See evaluation.file for a function that combines both operations.

aspect — a positive number specifying the desired aspect of the plot region (only available for PostScript files). In the default case x.axis="n.best", aspect refers to the absolute size of the plot region. Otherwise, it specifies the size ratio between percentage points on the x-axis and the y-axis. Setting aspect modifies the height of the plot (plot.height).

plot.width, plot.height — the width and height of a plot that is written to a PostScript file, measured in inches. plot.height may be overridden by the aspect parameter, even if it is set explicitly.

cex — character expansion factor for labels, annotations, and symbols in the plot (see par for details). If cex is not specified, the default supplied by ucs.par is used.

lex — added to the line widths of evaluation graphs and some decorations (note that this is not an expansion factor). If lex is not specified, the default supplied by ucs.par is used.

bw — if TRUE, the evaluation plot is drawn in black and white, which is mostly used in conjunction with file to produce figures for articles (defaults to FALSE). See below for details.

legend — a vector of character strings or expressions, used as labels in the legend of the plot (e.g. to show mathematical symbols instead of the names of association measures). Use legend=NULL to suppress the display of a legend box.

bottom.legend — if TRUE, draw legend box in bottom right corner of plot (default is top right corner).

`title`	a character vector or expression to be used as the main title of the plot (optional)
`...`	any other arguments are set as local graphics parameters (using `par`) before the evaluation plot is drawn

Details

When `y.axis="local.precision"`, the evaluation graphs show **local precision**, i.e. an estimate for the density of true positives around the n-th rank according to the respective association measure. Local precision is smoothed using a kernel density estimate with a Gaussian kernel (from the `density` function), based on a symmetric window covering approximately `window` candidates (default: 400). Consequently, the resulting values do not have a clear-cut interpretation and should not be used to evaluate the performance of association measures. They are rather a means of exploratory data analysis, helping to visualise the relation between association scores and the true positives in a data set (see Evert, 2004, Sec. 5.2 for an example).

In order to generalise evaluation results beyond the specific data set on which they were obtained, it is necessary to compute confidence intervals for the observed precision values and to test whether the observed result differences are significant. See (Evert, 2004, Sec. 5.3) for the methods used and the interpretation of their results.

Confidence intervals are computed by setting `conf=TRUE` and selecting an association measure with the `conf.am` parameter. The confidence intervals are displayed as a coloured or shaded region around the precision graph of this measure (confidence intervals are not available for graphs of recall or local precision). The default confidence level of 95% will rarely need to be changed. Optionally, a second confidence region can be displayed for a measure selected with the `conf.am2` parameter.

Significance tests for the result differences are activated by setting `test=TRUE` (not available for graphs of local precision). The evaluation results of two association measures (specified with `test.am1` and `test.am2`) are compared for selected n-best lists, and significant differences are marked by coloured triangles or arrows (when `x.axis="recall"`). The default significance level of 0.05 will rarely need to be changed. Use the `test.step` parameter to control the spacing of the triangles or arrows.

A significant difference indicates that measure A is truly better than measure B, rather than just as a coincidence in a single evaluation experiment. Formally, this "true performance" can be defined as the average precision of a measure, obtained by averaging over many similar evaluation experiments. Thus, a significant difference means that the average precision of A is higher than that of B, but it does not indicate how great the difference is. A tiny difference (say, of half a percent point) is hardly **relevant** for an application, even if there is significant evidence for it. If the `test.relevant` parameter is set, the `evaluation.plot` function attempts to estimate whether there is significant evidence for a relevant difference (of at least a many percent points as given by the value of `test.relevant`), and marks such cases by darker triangles or arrows. This feature should be considered experimental and used with caution, as the computation involves many approximations and guesses (exact statistical inference for the difference in true precision not being available).

It goes without saying that confidence regions and significance tests do not allow evaluation results to be generalised to a different extraction task (i.e. another type of cooccurrences or another definition of true positives), or even to the same task under different conditions (such as a source corpus from a different domain, register, time, or a corpus of different size). The unpredictability of the performance of association measures for different extraction tasks or under different conditions has been confirmed by various evaluation studies.

Generally, evaluation plots can be drawn in two modes: **colour** (`bw=FALSE`, the default) or **black and white** (`bw=TRUE`). The styles of evaluation graphs are controlled by the respective settings in `ucs.par`, while the appearance of various other elements is hard-coded in the `evaluation.plot`

function. In particular, confidence regions are either filled with a light background colour (colour mode) or shaded with diagonal lines (B/W mode). The triangles or arrows used to mark significant differences are yellow or red (indicating relevance) in colour mode, and light grey or dark grey (indicating relevance) in B/W mode. B/W mode is mainly used to produce PostScript files to be included as figures in articles, but can also be displayed on-screen for testing purposes.

The `evaluation.plot` function supports **evaluation based on random samples**, or RSE for short (Evert, 2004, Sec. 5.4). Missing values (`NA`) in the `tp` vector (or the `b.TP` variable in `ds`) are interpreted as unannotated candidates. In this case, precision, recall and local precision are computed as maxmium-likelihood estimates based on the annotated candidates. Confidence intervals and significance tests, which should not be absent from any RSE, are adjusted accordingly. A confidence interval for the baseline precision is automatically shown (by thin dotted lines) when RSE is detected. Note that n-best lists (as shown on the x-axis) still refer to the full data set, not just to the number of annotated candidates.

Note

The following functions are provided for compatibility with earlier versions of UCS/R: `precision.plot`, `recall.plot`, and `recall.precision.plot`. They are simple front-ends to `evaluation.plot` with the implicit parameter settings `y.axis="recall"` and `y.axis="precision", x.axis="recall"` for the latter two.

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`ucs.par`, `evaluation.file`, `read.ds.gz`, and `precision.recall`. The R script 'tutorial.R' in the 'script/' directory provides a gentle introduction to the wide range of possibilities offered by the `evaluation.plot` function.

`evaluation.table` *Precision/Recall Tables for the Evaluation of Association Measures (base)*

Description

A simple text-mode version of the precision/recall-based evaluation provided by the `plots` module. Returns a table of precision or recall values for a selected range of association measures on selected n-best lists. **This is a preliminary version of the function - both interface and functionality may change in future releases.**

Usage

```
evaluation.table(ds, keys, n, tp=ds$b.TP, recall=FALSE)
```

Arguments

ds	a UCS data set object, read in from a data set file with the read.ds.gz function. ds must contain rankings for the association measures listed in the keys parameter (use add.ranks to add such rankings to a data set object).
keys	a character vector specifying the names of association measures to be evaluated. Each name may be abbreviated to prefix that must be unique within the measures annotated in ds. Use the ds.find.am function to obtain a list of measures annotated in the data set, and see the ucsam manpage in UCS/Perl for detailed information about the association measures supported by the UCS system (with the shell command ucsdoc ucsam).
n	a vector of n-best sizes for which precision or recall values are computed
tp	a logical vector indicating true positives, parallel to the rows of the data set ds. If tp is not specified, the data set must contain a variable named b.TP which is used instead.
recall	if TRUE, returns table of recall values, otherwise table of precision values (default)

Value

A data frame whose rows correspond to n-best lists. In addition to the column labelled n, which gives the n-best lists for which the evaluation was carried out, there is one column for each selected association measure. The column is labelled with the name of the measure and lists the corresponding precision or recall values, depending on the recall parameter.

See Also

evaluation.plot, precision.recall

---

fzm — *The Finite Zipf-Mandelbrot LNRE Model (fzm)*

Description

Object constructor for a finite Zipf-Mandelbrot (fZM) LNRE model with parameters α, A and B (Evert, 2004a). Either the parameters are specified explicitly, or one or more of them can be estimated from an observed frequency spectrum.

Usage

```
fzm(alpha, A, B)

fzm(alpha, A, N, V)

fzm(alpha, N, V, spc, m.max=15, stepmax=10, debug=FALSE)

fzm(N, V, spc, m.max=15, stepmax=10, debug=FALSE)
```

Arguments

`alpha`	a number in the range $(0, 1)$, the shape parameter α of the fZM model. `alpha` can automatically be estimated from `N`, `V`, and `spc`.
`A`	a small positive number $A \ll 1$, the parameter A of the fZM model. `A` can automatically be estimated from `N`, `V`, and `spc`.
`B`	a large positive number $B \gg 1$, the parameter B of the fZM model. `B` can automatically be estimated from `N` and `V`.
`N`	the sample size, i.e. number of observed tokens
`V`	the vocabulary size, i.e. the number of observed types
`spc`	a vector of non-negative integers representing the class sizes V_m of the observed frequency spectrum. The vector is usually read from a file in `lexstats` format with the `read.spectrum` function.
`m.max`	the number of ranks from `spc` that will be used to estimate the α parameter
`stepmax`	maximal step size of the `nlm` function used for parameter estimation. It should not be necessary to change the default value.
`debug`	if `TRUE`, print debugging information during the parameter estimation process. This feature can be useful to find out why parameter estimation fails.

Details

The fZM model with parameters $\alpha \in (0, 1)$ and $C > 0$ is defined by the type density function

$$g(\pi) := C \cdot \pi^{-\alpha-1}$$

for $A \leq \pi \leq B$. The normalisation constant C is determined from the other parameters by the condition

$$\int_A^B \pi \cdot g(\pi) d\pi = 1$$

The parameters α and A are estimated simultaneously by nonlinear minimisation (`nlm`) of a multinomial chi-squared statistic for the observed against the expected frequency spectrum. Note that this is different from the multivariate chi-squared test used to measure the goodness-of-fit of the final model (Baayen, 2001, Sec. 3.3).

See Evert (2004, Ch. 4) for further mathematical details, especially concerning the expected vocabulary size, frequency spectrum and conditional parameter distribution, as well as their variances.

Value

An object of class `"fzm"` with the following components:

`alpha`	value of the α parameter
`A`	value of the A parameter
`B`	value of the B parameter
`C`	value of the normalisation constant C
`C`	population size S predicted by the model
`N`	number of observed tokens (if specified)
`V`	number of observed types (if specified)
`spc`	observed frequency spectrum (if specified)

This object `prints` a short summary, including the population size S and a comparison of the first ranks of the observed and expected frequency spectrum (if available).

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

Evert, Stefan (2004a). A simple LNRE model for random character sequences. In *Proceedings of JADT 2004*, Louvain-la-Neuve, Belgium, pages 411–422.

See Also

`zm`, `EV`, `EVm`, `VV`, `VVm`, `write.lexstats`, `lnre.goodness.of.fit`, `read.spectrum`, and `spectrum.plot`

`gam.helpers` *Helper Functions for GAM Equations (gam)*

Description

`gam.yates` and `gam.yates.inv` implement an invertible version of the discounting function used by Yates' correction. `signed.sqrt`, `b.star`, `b.norm` and `e.bar` are standard abbreviations used in the definition of generalised association measures in terms of ebo-coordinates.

Usage

```
gam.yates(d)
gam.yates.inv(d.corr)

signed.sqrt(x)

b.star(b)
b.norm(b)
e.bar(e, b, N)
```

Arguments

`d`	difference between observed and expected frequency, to which the generalised Yates' correction is applied
`d.corr`	difference between observed and expected frequency with generalised Yates' correction applied, from which the original difference can uniquely be reconstructed
`x`	a vector of positive or negative real numbers
`b`	a vector of *balance* (b) values in the ebo coordinate system
`e`	a vector of *expectation* (e) values in the ebo coordinate system
`N`	sample size N

Details

The standard discounting function for Yates' correction is $d^* := d - 1/2$ for $d \geq 0$ and $d^* := d + 1/2$ for $d < 0$, where d is the difference between observed and expected frequency. This definition does not lead to a continuous and invertible function of d, so a GAM with Yates' correction applied does not satisfy the soundness conditions. The generalised Yates' correction implemented by `gam.yates` and `gam.yates.inv` is a monotonic (and hence invertible) function that is identical to the standard discounting function for $|d| \geq 1$ and uses linear interpolation for $-1 < d < 1$.

The functions `signed.sqrt`, `b.star`, `b.norm` and `e.bar` compute the standard abbreviation $\pm\sqrt{x}$, b^*, $\|b\|$ and $\bar{e}$ ("e bar") used by Evert (2004) for the definition of GAMs in terms of ebo-coordinates.

Value

all functions return a vector of real numbers

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

Examples

```
d <- runif(20, -2, 2)
d.corr <- gam.yates(d)
all(d == gam.yates.inv(d.corr))

signed.sqrt(-4:4)
```

---

gam.iso *Compute Iso-Surfaces for GAMs (gam)*

Description

Computes iso-surfaces for a generalised association measure (GAM) in standard or ebo-coordinates.

Usage

```
gam.iso(name, gamma, f1, f2, N, bsearch.min=NULL, bsearch.max=NULL)
gam.iso(name, gamme, e, b=1, N=1e6, bsearch.min=NULL, bsearch.max=NULL)
```

Arguments

`name`	name of a generalised association measure (GAM)
`gamma`	a numerical constant that determines the desired iso-surface $\{g = \gamma\}$
`f1, f2, N`	numerical vectors specifying the `f1` and `f2` coordinates of points in the standard coordinate space, as well as the sample size `N`
`e, b`	numerical vectors specifying the `e` and `b` coordinates of points in the ebo-coordinate space (if the *balance* b is not specified, it defaults to `1`)
`N`	optional numerical vector specifying the sample size N when computing iso-surfaces for a GAM that is not size-invariant in ebo-coordinates (defaults to `1e6`)

`bsearch.min` initial lower boundary for binary search algorithm, when no explicit equation for the iso-surface is available

`bsearch.max` initial upper boundary for the binary search algorithm

Details

Note that all function arguments except for `name` must be passed explicitly by name in order to distinguish the two operating modes of `gam.iso` (standard vs. ebo-coordinates).

When ebo-coordinates are used, the argument `N` (*sample size*) can safely be omitted for any size-invariant GAM (in ebo-coordinates). For other GAMs, a default value of `1e6` will be used, corresponding to the typical size of a co-occurrence data set. The argument `b` (*balance*) can be omitted for any central GAMs. Otherwise, it defaults to a value of `1`, corresponding to the centralized version of the respective GAM.

Use `gamma.nbest` to compute a suitable γ values for n-best surfaces.

When no explicit equation for the iso-surface of a GAM is available, the `gam.iso` function uses a binary search algorithm to solve the implicit equation $\{g = \gamma\}$. Since some GAMs are only defined for valid frequency signatures (where all four cells of the contingency table are non-negative), the binary search for the `o` coordinate is confined to the range from 0 to $\min\{f_1, f_2\}$. When no solution can be found in this range, `gam.iso` returns `NA` for the corresponding points. For GAMs where it is safe to search a larger range (notably `Poisson.pv` and `log.likelihood`), the boundaries of the search interval can be adjusted with the `bsearch.min` and `bsearch.max` parameters. Note that most other GAMs have explicit iso-equations, so these parameters are rarely needed.

Value

a vector of real numbers representing the `f` or `o` coordinates of the respective iso-surface; these are the values of `f` or `o` that solve the implicit equation $\{g = \gamma\}$ for the specified values of `f1, f2, N` or `e, b` (and `N`); this vector may contain missing values (`NA`) for points where no solution is found (see "Details" for more information)

See Also

`gam.score`, `builtin.gams`, `gamma.nbest`

Examples

```
e <- 10^seq(-2, 1, .1)          # compute iso-line on logarithmic scale
o <- gam.iso("t.score", 2, e=e)

x <- 10^seq(0, 2, .1)           # compute iso-surface over rectangular grid
g <- expand.grid(f1=x, f2=x)
g$f <- gam.iso("t.score", 2, f1=g$f1, f2=g$f2, N=1000)
library(lattice)
wireframe(f ~ f1 * f2, log(g))
```

`gam.score` *Compute GAM Scores in Standard or EBO-Coordinates (gam)*

Description

Computes scores of a generalised association measure (GAM) in standard or ebo-coordinates.

Usage

```
gam.score(name, f, f1, f2, N)
gam.score(name, o, e, b=1, N=1e6)
```

Arguments

`name`	name of a generalised association measure (GAM)
`f, f1, f2, N`	numerical vectors specifying the (generalised) frequency signatures of candidates
`o, e, b`	numerical vectors specifying the ebo-coordinates of candidates (if the *balance* b is not specified, it defaults to `1`)
`N`	optional numerical vector specifying the sample size N when computing scores of a GAM that is not size-invariant in ebo-coordinates (defaults to `1e6`)

Details

Note that all function arguments except for `name` must be passed explicitly by name in order to distinguish the two operating modes of `gam.score` (standard vs. ebo-coordinates).

The components of the generalised frequency signature `(f, f1, f2, N)` can be arbitrary positive real numbers.

When ebo-coordinates are used, the argument `N` (*sample size*) can safely be omitted for any size-invariant GAM (in ebo-coordinates). For other GAMs, a default value of `1e6` will be used, corresponding to the typical size of a co-occurrence data set. The argument `b` (*balance*) can be omitted for any central GAMs. Otherwise, it defaults to a value of `1`, corresponding to the centralized version of the respective GAM.

The `gam.score` function automatically converts between standard and ebo-coordinates, depending on the requirements of the GAM implementation.

Value

a vector of real numbers representing generalised association scores

See Also

`add.gams`, `gam.iso`, `builtin.gams`

Examples

```
gam.score("t.score", f=1:10, f1=(1:10)*5, f2=100, N=1000)

gam.score("t.score", o=1:10, e=(1:10)/2)
```

`gamma.nbest` *Compute Gamma Threshold for N-Best Acceptance Region (gam)*

Description

Computes a suitable value of γ such that the acceptance region $\{g \geq \gamma\}$ contains exactly n candidates from a given data set.

Usage

```
gamma.nbest(ds, name, n, jitter=FALSE)
```

Arguments

`ds`	a UCS data set object
`name`	name of a generalised association measure (GAM)
`n`	an integer, specifying the number of candidates to be included in the acceptance region
`jitter`	if `TRUE`, random jitter is added to the coordinates of candidates for computation of the n-best threshold

Details

When `jitter=TRUE`, the data set `ds` must contain jitter vectors stored in special variables. Such jitter variables can easily be added with the `add.jitter` function.

Value

a real number specifying a suitable threshold γ, i.e. the data set `ds` contains exactly `n` candidates with a GAM score $g \geq \gamma$ (for the specified measure `name`)

See Also

`add.jitter`, `gam.score`, `add.gams`, `gam.iso`, `builtin.gams`

Examples

```
e <- 10^seq(-2, 1, .1)          # 100-best iso-line for UCS data set ds
gamma <- gamma.nbest(ds, "t.score", 100)
o <- gam.iso("t.score", gamma, e=e)
```

`iaa.kappa` *Inter-Annotator Agreement: Cohen's Kappa (iaa)*

Description

Compute the kappa statistic (Cohen, 1960) as a measure of intercoder agreement on a binary variable between two annotators, as well as a confidence interval according to Fleiss, Cohen & Everitt (1969). The data can either be given in the form of a 2×2 contingency table or as two parallel annotation vectors.

Usage

```
iaa.kappa(x, y=NULL, conf.level=0.95)
```

Arguments

`x`	either a 2×2 contingency table in matrix form, or a vector of logicals
`y`	a vector of logicals; ignored if `x` is a matrix
`conf.level`	confidence level of the returned confidence interval (default: 0.95, corresponding to 95% confidence)

Value

A data frame with a single row and the following variables:

`kappa`	sample estimate for the kappa statistic
`sd`	sample estimate for the standard deviation of the kappa statistic
`kappa.min, kappa.max`	two-sided asymptotic confidence interval for the "true" kappa, based on normal approximation with estimated variance

The single-row data frame was chosen as a return structure because it `prints` nicely, and results from different comparisons can easily be combined with `rbind`.

References

Cohen, Jacob (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, **20**, 37–46.

Fleiss, Joseph L.; Cohen, Jacob; Everitt, B. S. (1969). Large sample standard errors of kappa and weighted kappa. *Psychological Bulletin*, **72**(5), 323–327.

See Also

`iaa.pta`

Examples

```
## kappa should be close to zero for random codings
p <- 0.1                        # proportion of true positives
x <- runif(1000) < p            # 1000 candidates annotated randomly
y <- runif(1000) < p
iaa.kappa(x, y)
```

`iaa.pta` *Inter-Annotator Agreement: Estimates for the Proportion of True Agreement (iaa)*

Description

Compute confidence interval estimates for the proportion of true agreement between two annotators on a binary variable, as described by Krenn, Evert & Zinsmeister (2004). `iaa.pta.conservative` computes a conservative estimate that is rarely useful, while `iaa.pta.homogeneous` relies on additional assumptions. The data can either be given in the form of a 2×2 contingency table or as two parallel annotation vectors.

Usage

```
iaa.pta.conservative(x, y=NULL, conf.level=0.95, debug=FALSE)

iaa.pta.homogeneous(x, y=NULL, conf.level=0.95, debug=FALSE)
```

Arguments

`x`	either a 2×2 contingency table in matrix form, or a vector of logicals
`y`	a vector of logicals; ignored if `x` is a matrix
`conf.level`	confidence level of the returned confidence interval (default: 0.95, corresponding to 95% confidence)
`debug`	if `TRUE`, show which divisions of the data are considered when computing the confidence interval (see Krenn, Evert & Zinsmeister, 2004)

Details

This approach to measuring intercoder agreement is based on the assumption that the observed **surface agreement** in the data can be divided into **true agreement** (i.e. candidates where both annotators make the same choice *for the same reasons*) and **chance agreement** (i.e. candidates on which the annotators agree purely by coincidence). The goal is to estimate the proportion of candidates for which there is true agreement between the annotators, referred to as PTA.

The two functions differ in how they compute this estimate. `iaa.pta.conservative` considers all possible divisions of the observed data into true and chance agreement, leading to a conservative confidence interval. This interval is almost always too large to be of any practical value.

`iaa.pta.homogeneous` makes the additional assumption that the average proportion of true positives is the same for the part of the data where the annotators reach true agreement and for the part where they agree only by chance. Note that there is no *a priori* reason why this should be the case. Interestingly, the confidence intervals obtained in this way for the PTA correspond closely to those for Cohen's kappa statistic (`iaa.kappa`).

Value

A numeric vector giving the lower and upper bound of a confidence interval for the proportion of true agreement (both in the range $[0, 1]$).

Note

`iaa.pta.conservative` is a computationally expensive operation based on Fisher's exact test. (It doesn't use `fisher.test`, though. If it did, it would be even slower than it is now.) In most circumstances, you will want to use `iaa.pta.homogeneous` instead.

References

Krenn, Brigitte; Evert, Stefan; Zinsmeister, Heike (2004). Determining intercoder agreement for a collocation identification task. In preparation.

See Also

`iaa.kappa`

Examples

```
## how well do the confidence intervals match the true PTA?
true.agreement <- 700           # 700 cases of true agreement
chance <- 300                   # 300 cases where annotations are independent
p <- 0.1                        # average proportion of true positives
z <- runif(true.agreement) < p  # candidates with true agreement
x.r <- runif(chance) < p        # randomly annotated candidates
y.r <- runif(chance) < p
x <- c(z, x.r)
y <- c(z, y.r)
cat("True PTA =", true.agreement / (true.agreement + chance), "\n")
iaa.pta.conservative(x, y)      # conservative estimate
iaa.pta.homogeneous(x, y)       # estimate with homogeneity assumption
```

`lnre.goodness.of.fit`

Perform Goodness-of-Fit Evaluation of LNRE Model

Description

Evaluate the goodness-of-fit of a LNRE model with a multivariate chi-squared test (Baayen, 2001, Sec. 3.3).

Usage

```
lnre.goodness.of.fit(model, m.max=15)
```

Arguments

`model`	an object representing a LNRE model whose parameters have been estimated from observed word frequency data. Currently, the Zipf-Mandelbrot (ZM, class `"zm"`) and the finite Zipf-Mandelbrot (fZM, class `"fzm"`) models are supported.
`m.max`	highest frequency rank to be included in the evaluation (limited by the number of ranks stored in the `model` object).

Details

This function performs a multivariate chi-squared test to evaluate the goodness-of-fit of an LNRE model (Baayen 2001, p. 119-122).

All LNRE models that follow the UCS/R conventions are supported. In particular, they must specify the number of parameters estimated from the observed data (in the `n.param` component), and they must provide appropriate implementations of the `EV`, `EVm`, and `VV` methods. Currently available LNRE models are objects of class `"zm"` or `"fzm"`. The `model` object must include observed frequency data (in components `N`, `V`, and `spc`), which is usually achieved by estimating the model parameters from the observed frequency spectrum.

Value

A data frame with one row and three columns:

`X2`	the value of the multi-variate χ^2 test statistic
`df`	the degrees of freedom of the approximate χ^2 distribution of the test statistic under the null hypothesis
`p`	the p-value for the test

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`zm`, `fzm`

`order.by.am` *Sort Rows of a Data Set by Association Scores (base)*

Description

Sort the rows of a data set according to the annotated scores of an association measure (in descending order). Ties in the ordering are broken randomly by default, using the `random` association measure to yield a reproducible ordering.

Usage

```
order.by.am(ds, am, randomise=TRUE)
```

Details

With `randomise=TRUE`, the data set must contain a variable named `am.random`, which is used to break ties in the ordering. Otherwise, tied rows are arranged according to their ID values, and the corresponding `id` variable must be annotated in the data set.

The `random` association measure is used for breaking ties (rather than random numbers generated on the fly) in order to ensure that the ordering is reproducible. If this measure has not been annotated in a data set file, you can easily add the required variable to a data set `ds` with the command

```
ds$am.random <- runif(nrow(ds))
```

You should probably use `set.seed` to ensure a reproducible ordering.

Value

an integer vector of row numbers, which can be used as a row index for the data set object

See Also

read.ds.gz, add.ranks

---

precision.recall *Compute Precision and Recall for N-Best Lists (base)*

Description

Computes precision and recall of n-best lists for a UCS data set annotated with true positives and rankings (based on association scores). This function forms the basis for the evaluation graphs in the plots packages.

Usage

```
precision.recall(ds, am, tp=ds$b.TP, step=1, first=1, cut=0, window=0)
```

Arguments

ds	a UCS data set object
am	a character string giving the name of an association measure. The corresponding ranking must be annotated in the data set (usually with the add.ranks function).
tp	a logical vector, which must be parallel to the rows of the data set. TRUE values indicate true positives (see details below for the use of missing values). If tp is omitted, the data set must contain a Boolean variable b.TP which is used instead.
step	step width for n-best lists considered, i.e. precision and recall are computed for every step-th value of n only (default: 1)
first	smallest n-best list for which precision and recall are computed (default: 1)
cut	pretend that data set consists only of the first cut rows in the ranking, i.e. treat cut-best list as full data set (for percentage and recall).
window	if specified, local precision is estimated, considering a window of approximately the given size around each value of n (uses the density function for smoothing). Useful window sizes range from 400 to 1000.

Details

The precision.recall function supports evaluation based on random samples (cf. Evert, 2004, Sec. 5.4). Any NA values in the tp parameter (or the b.TP variable) are interpreted as unannotated candidates. Precision and recall values are computed from the annotated candidates only (as are the tp, fp, and lp variables in the returned data frame). For a random sample evaluation, confidence intervals should always be supplied with the raw precision values, and result differences should be tested for significance. Such tests are implemented by the evaluation.plot function, for instance.

Value

An invisible data frame with rows corresponding to n-best lists and the following variables:

`n`	the number of candidates in the n-best list
`perc`	the same as a percentage of the full data set (or the `cut` highest-ranking candidates if specified)
`tp`	the number of true positives in the n-best list
`fp`	the number of false positives in the n-best list
`precision`	the precision of the n-best list, i.e. the number of TPs divided by n
`recall`	the recall of the n-best list, i.e. the number of TPs divided by the total number of TPs in the data set
`lp`	if `window` is specified, an estimate for the *local precision*, i.e. the density of TPs in the vicinity of the n-th rank. Averages over a symmetric window of approximately the specified total size by convolution with a Gaussian kernel (using the `density` function).

References

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

See Also

`add.ranks`, `read.ds.gz`, `evaluation.plot`

`read.ds.gz` *Load UCS data set file (base)*

Description

Load a UCS data set file, which is uncompressed on the fly if necessary.

Usage

```
read.ds.gz(filename)
```

Arguments

`filename`	name, partial or full path of the data set file to be loaded.

Details

When the specified file is not found in the current directory, it is automatically searched in the standard UCS data library (the 'DataSet/' directory and its subdirectories). Should there be multiple matches, a warning is issued and the first match is used. You may specify partial paths to identify the desired file unambiguously (e.g. `"Distrib/dickens.ds.gz"`). The automatic search facility is suppressed when `filename` is an explicit absolute or relative path (starting with `/` or `./`).

`gzip`-compressed data set files, whose name must end in `.gz`, are automagically decompressed.

Value

A data frame with column names (i.e. variables) corresponding to those in the data set file. `l1` and `l2` are read as character vectors, all other string variables (`f.*`) are converted into factors, and Boolean variables (`b.*`) are converted into logicals.

Any comments and global variables in the file header are discarded.

Examples

```
## load GLAW data set from UCS distribution
GLAW <- read.ds.gz("glaw.ds.gz")
```

`read.spectrum` *Read Frequency Spectrum File (lexstats)*

Description

Read a word frequency spectrum from a `.spc` file in `lexstats` format (see Baayen, 2001). Returns spectrum as integer vector, possibly including zeroes, whose m-th element gives the number of types V_m with frequency rank m. Also computes sample size N and vocabulary size V.

Usage

```
read.spectrum(file, m.max=Inf, expected=FALSE)
```

Arguments

`file`	a character string giving the name of a frequency spectrum file in `lexstats` format (usually with the extension `.spc`)
`m.max`	maximum length of frequency spectrum, i.e. frequency ranks $m > m_{\max}$ are discarded. Setting `m.max` is a good idea if there are high-frequency types, so that the spectrum is sparse. For most applications, only the first 10 to 100 ranks are of interest.
`expected`	if `TRUE`, reads expected class sizes (in the `EVm` column) rather than the observed ones (in the `Vm` column). This is only possible when the `.spc` file was generated by a LNRE model, of course.

Value

A list with the following components:

`spc`	an integer vector containing the class sizes V_m
`N`	the sample size computed from the spectrum
`V`	the vocabulary size computed from the spectrum

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`spectrum.plot`, `zm`, `fzm`

spectrum.plot *Comparative Plot of Word Frequency Spectra (lexstats)*

Description

Comparative plot of up to five word frequency spectra (see Baayen, 2001), either as a side-by-side barplot or as points and lines on a logarithmic scale.

Usage

```
spectrum.plot(spc, m.max=Inf, log=FALSE, y.min=100, y.max=0,
              xlab="m", ylab="V_m / E[V_m]",
              legend=NULL,
              pch=c(1, 3, 15, 2, 20),
              lwd=1,
              lty=c("solid", "dashed", "dotdash", "dotted", "twodash"),
              col=if (log) c("black") else c("black", "grey50", ...))
```

Arguments

spc	a list containing up to five frequency spectrum vectors. Such spectrum vectors can be read in from a file in lexstats format with read.spectrum or generated by a ZM or fZM model with the EVm method.
m.max	number of frequency ranks to be shown in plot. If unspecified, it is determined by the shortest spectrum vector in spc.
log	if TRUE, display frequency spectra as points and lines on a logarithmic scale. If FALSE, display spectra as side-by-side barplot on a linear scale (default). The latter is only useful when m.max is comparatively small.
y.min, y.max	range of y-axis. y.max is automatically computed to fit the data in spc. y.min is only used when log=TRUE and defaults to 100.
legend	a vector of character strings or expressions specifying the labels to be shown in a legend box. If legend is missing, no legend box will be displayed.
xlab, ylab	character strings giving labels for the x-axis and y-axis
pch, lwd, lty	vectors of plot symbols, line widths, and line types (only used if log=TRUE. Values are recycled if necessary. See the par manpage for possible ways of specifying these attributes.
col	a vector of colours for the lines (log=TRUE) or bars (log=FALSE) in the plot. Values are recycled if necessary. Colours are specified as described in the par manpage.

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

read.spectrum, zm, fzm, EVm

`ucs.library` *Load UCS/R Modules*

Description

Since the UCS/R functions are imported into the global namespace, they are collected in various modules that can be loaded separately on demand. `ucs.library` loads a specified module. When called without arguments, it prints a listing of available modules.

Usage

```
ucs.library(name, all=FALSE, reload=FALSE)
```

Arguments

`name`	a character string giving the name of a *single* UCS/R module to be loaded. If omitted, a list of all available modules is displayed (see below).
`all`	if `TRUE`, all available modules are loaded
`reload`	if `TRUE`, force module to be loaded even if it has already been imported (useful when developing UCS/R modules)

Details

Like the `library` and `package` functions, `ucs.library(module)` checks whether the requested module has already been loaded by a previous `ucs.library` call. Set `reload=TRUE` in order to skip this test and force re-loading a module (especially while developing or debugging module code).

Value

Calling the `ucs.library` function without arguments returns a list of all available UCS/R modules as an object of class `"UCSLibList"`, which prints as a nicely formatted listing including one-line descriptions. Use `names(ucs.library())` to obtain a plain vector of module names.

See Also

`UCS` for an overview of the UCS/R modules

Examples

```
print(ucs.library())  # list of available modules

ucs.library("base")   # load and manage UCS data sets
ucs.library("plots")  # evaluation graphs

ucs.library(all=TRUE) # load all modules
```

ucs.par *Graphics Parameters for Evaluation Graphs (plots)*

Description

Set default graphics parameters for the `evaluation.plot` function, similar to `par` for general graphics parameters. The current parameter values are queried by giving their names as character strings. The values can be set by specifying them as arguments in `name=value` form, or by passing a single list of named values.

Usage

```
ucs.par(...)

.ucs.PAR
```

Arguments

`...`	either character strings (or vectors) specifying the names of parameters to be queried, or parameters to be set in `name=value` form, or a single list of named values. Valid parameter names are described below.

Details

The current default parameters are stored in the global variable `.ucs.PAR`. They can be queried by giving their names as one or more character vectors to `ucs.par`. `ucs.par()` (no arguments) returns all UCS graphics parameters.

Parameters are set by specifying their names and the new values as `name=value` pairs. Such a list can also be passed as a single argument to `ucs.par`, which is typically used to restore previous parameter values (that have been saved in a list variable).

In order to restore the "factory settings", reload the module with the command `ucs.library("plots", reload=TRUE)`.

Value

When parameters are set, their former values are returned in an invisible named list. Such a list can be passed as a single argument to `ucs.par` to restore the parameter values.

When a single parameter is queried, its value is returned directly. When two or more parameters are queried, the result is a named list.

Note the inconsistency, which is the same as for `par`: setting one parameter returns a list, but querying one parameter returns a vector (or a scalar, i.e. a vector of length 1).

UCS Graphics Parameters

`col` A character or integer vector specifying line colours for up to 10 evaluation graphs (see the `par` manpage for details). Values are recycled if necessary.

`lty` A character or integer vector specifying line styles for up to 10 evaluation graphs (see the `par` manpage for details). Values are recycled if necessary.

`lwd` A numeric vector specifying line widths for up to 10 evaluation graphs (see the `par` manpage for details). Values are recycled if necessary.

bw.col The line colours used in B/W mode (see the evaluation.plot manpage for details).

bw.lty The line styles used in B/W mode.

bw.lwd The line widths in B/W mode.

n.first The smallest n-best list to be evaluated (default: 100). Shorter n-best lists typically lead to highly unstable evaluation graphs. It may be necessary to set n.first to a higher value for evaluation based on random samples (cf. evaluation.plot).

n.step The step width for n-best lists in evaluation graphs (default: 1). The default setting evaluates all possible n-best lists. Higher values speed up computation, make graphs look less jagged, and reduce the size of PostScript files. A useful range is $5 \ldots 20$, depending on the size of the data set file.

test.step Step width for n-best lists where significance tests for result differences are applied, as a multiple of n.step (default: 10). Since these tests are time-consuming and significant differences are indicated by fairly large symbols in the plot, values below 5 are rarely useful.

cex A character expansion factor for labels, annotations, and symbols in evaluation plots (see par for details).

lex This parameter can be used to increase the line widths of evaluation graphs and some decorations. Not that lex is not an expansion factor, but is simply *added* to all line widths in the plot.

do.file If FALSE, evaluation.file will not generate PostScript files, which is useful while testing and fine-tuning plots (default: TRUE).

See Also

evaluation.plot, evaluation.file, par

Examples

```
print(names(ucs.par()))          # list available parameters

ucs.par("col", "lty", "lwd")     # the default line styles
ucs.par(c("col", "lty", "lwd")) # works as well

## temporary changes to graphics paramters:
par.save <- ucs.par(n.first=200, n.step=5)
## plots use the modified parameters here
ucs.par(par.save)                # restore previous values

ucs.library("plots", reload=TRUE) # reload module for factory defaults
```

---

write.lexstats *Write Data Files for Goodness-of-Fit Evaluation of LNRE Model (zm, fzm)*

Description

Creates three data files in lexstats format, which can be used to compare and LNRE model with other models from the lexstats package and evaluate its goodness-of-fit by a multivariate chi-squared test (Baayen, 2001, Sec. 3.3), using the lnreChi2 program (Baayen, 2001).

Usage

```
write.lexstats(model, file)
```

Arguments

`model`	an object of class `"zm"` or `"fzm"`, representing a Zipf-Mandelbrot (ZM) or finite Zipf-Mandelbrot (fZM) LNRE model. The object must include observed word frequency data (in components `N`, `V`, and `spc`), usually because the model parameters have been estimated from the observed frequency spectrum.
`file`	a character string giving the basename of the files that will be created

Details

This functions creates files in `lexstats` format with the extensions `.spc`, `.sp2`, and `.ev2`, which are required by the `lnreChi2` tool (Baayen, 2001, 270).

In addition, the basename `file` is extended with the string `"_bZM"` (for a ZM model) or `"_bfZM"` (for a fZM model), so that the `lnreChi2` tool can correctly identify the number of degrees of freedom (reduced by two estimated parameters for the ZM model, and three estimated parameters for the fZM model).

Value

The full basename of the created files (obtained by adding a model-specific suffix to the `file` parameter).

Note

The combination of `write.lexstats` and the external `lnreChi2` program to evaluate the goodness-of-fit of a LNRE model has been superseded by the built-in `lnre.goodness.of.fit` function (in the `lexstats` module). This function implements the multivariate chi-squared test as described by Baayen (2001, Sec. 3.3) in R without relying on external software.

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

See Also

`zm`, `fzm`, `EV`, `EVm`, `lnre.goodness.of.fit`

`zm` — *The Zipf-Mandelbrot LNRE Model (zm)*

Description

Object constructor for a Zipf-Mandelbrot (ZM) LNRE model with parameters α and C (Evert, 2004a). Either the parameters are specified explicitly, or one or both of them can be estimated from an observed frequency spectrum.

Usage

```
zm(alpha, C)

zm(alpha, N, V)

zm(N, V, spc, m.max=15, stepmax=10, debug=FALSE)
```

Arguments

`alpha`	a number in the range $(0, 1)$, the shape parameter α of the ZM model. `alpha` can automatically be estimated from `N`, `V`, and `spc`.
`C`	a positive number, the parameter C of the ZM model. `C` can automatically be estimated from `N` and `V`.
`N`	the sample size, i.e. number of observed tokens
`V`	the vocabulary size, i.e. the number of observed types
`spc`	a vector of non-negative integers representing the class sizes V_m of the observed frequency spectrum. The vector is usually read from a file in `lexstats` format with the `read.spectrum` function.
`m.max`	the number of ranks from `spc` that will be used to estimate the α parameter
`stepmax`	maximal step size of the `nlm` function used for parameter estimation. It should not be necessary to change the default value.
`debug`	if `TRUE`, print debugging information during the parameter estimation process. This feature can be useful to find out why parameter estimation fails.

Details

The ZM model with parameters $\alpha \in (0, 1)$ and $C > 0$ is defined by the type density function

$$g(\pi) := C \cdot \pi^{-\alpha-1}$$

for $0 \leq \pi \leq B$, where the upper bound B is determined from C by the normalisation condition

$$\int_0^\infty \pi \cdot g(\pi) d\pi = 1$$

The parameter α is estimated by nonlinear minimisation (`nlm`) of a multinomial chi-squared statistic for the observed against the expected frequency spectrum. Note that this is different from the multivariate chi-squared test used to measure the goodness-of-fit of the final model (Baayen, 2001, Sec. 3.3).

See Evert (2004, Ch. 4) for further mathematical details, especially concerning the expected vocabulary size, frequency spectrum and conditional parameter distribution, as well as their variances.

Value

An object of class `"zm"` with the following components:

`alpha`	value of the α parameter
`B`	value of the upper bound B (a normalisation device)
`C`	value of the C parameter
`N`	number of observed tokens (if specified)

`V`	number of observed types (if specified)
`spc`	observed frequency spectrum (if specified)

This object `prints` a short summary, including a comparison of the first ranks of the observed and expected frequency spectrum (if available).

References

Baayen, R. Harald (2001). *Word Frequency Distributions.* Kluwer, Dordrecht.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart.

Evert, Stefan (2004a). A simple LNRE model for random character sequences. In *Proceedings of JADT 2004*, Louvain-la-Neuve, Belgium, pages 411–422.

See Also

`fzm`, `EV`, `EVm`, `VV`, `VVm`, `write.lexstats`, `lnre.goodness.of.fit`, `read.spectrum`, and `spectrum.plot`

Index