Package ‘zipfR’

August 30, 2007

Type Package

Title Statistical models for word frequency distributions

Version 0.6-4

Depends R (>= 2.3.0)

Date 2007-08-30

Author Stefan Evert <stefan.evert@uos.de>, Marco Baroni <marco.baroni@unitn.it>

Maintainer Stefan Evert <stefan.evert@uos.de>

Description Statistical models and utilities for the analysis of word frequency distributions. The utilities include functions for loading, manipulating and visualizing word frequency

License GPL

URL http://purl.org/stefan.evert/zipfR/

R topics documented:

`Brown` *Brown Corpus Frequency Data (zipfR)*

Description

`Brown.tfl`, `Brown.spc` and `Brown.emp.vgc` are `zipfR` objects of classes `tfl`, `spc` and `vgc`, respectively.

These data were extracted from the Brown corpus (see Kucera and Francis 1967).

Details

`Brown.emp.vgc` is the *empirical* vocabulary growth curve, reflecting the `V` and `V(1)` development in the non-randomized corpus.

We removed numbers and other forms of non-linguistic material before collecting word counts from the Brown.

References

Kucera, H. and Francis, W.N. (1967). *Computational analysis of present-day American English.* Brown University Press, Providence.

See Also

The datasets documented in BrownSubsets pertain to various subsets of the Brown (e.g., informative prose, adjectives only, etc.)

Examples

```
data(Brown.tfl)
summary(Brown.tfl)

data(Brown.spc)
summary(Brown.spc)

data(Brown.emp.vgc)
summary(Brown.emp.vgc)
```

BrownSubsets *Brown Corpus Subset Frequency Data (zipfR)*

Description

Objects of classes spc and vgc that contain frequency data for various subsets of words from the Brown corpus (see Kucera and Francis 1967).

Details

`BrownAdj.spc`, `BrownNoun.spc` and `BrownVer.spc` are frequency spectra of all the Brown corpus words tagged as adjectives, nouns and verbs, respectively. `BrownAdj.emp.vgc`, `BrownNoun.emp.vgc` and `BrownVer.emp.vgc` are the corresponding observed vocabulary growth curves (tracking the development of `V` and `V(1)`, like all the files with suffix `.emp.vgc` below).

`BrownImag.spc` and `BrownInform.spc` are frequency spectra of the Brown corpus words subdivided into the two main stylistic partitions of the corpus, i.e., imaginative and informative prose, respectively. `BrownImag.emp.vgc` and `BrownInform.emp.vgc` are the corresponding observed vocabulary growth curves.

`Brown100k.spc` is the spectrum of the first 100,000 tokens in the Brown (useful, e.g., for extrapolation experiments in which we want to estimate a `lnre` model on a subset of the data available). The corresponding observed growth curve can be easily obtained from the one for the whole Brown (`Brown.emp.vgc`).

Notice that we removed numbers and other forms of non-linguistic material before collecting any data from the Brown.

References

Kucera, H. and Francis, W.N. (1967). *Computational analysis of present-day American English*. Brown University Press, Providence.

See Also

The data described in Brown pertain to the Brown as a whole.

Examples

```
data(BrownAdj.spc)
summary(BrownAdj.spc)

data(BrownAdj.emp.vgc)
summary(BrownAdj.emp.vgc)

data(BrownInform.spc)
summary(BrownInform.spc)

data(BrownInform.emp.vgc)
summary(BrownInform.emp.vgc)

data(Brown100k.spc)
summary(Brown100k.spc)
```

`Dickens` *Dickens' Frequency Data (zipfR)*

Description

Objects of classes `spc` and `vgc` that contain frequency data for a collection of Dickens's works from Project Gutenberg, and for 3 novels (Oliver Twist, Great Expectations and Our Mutual Friends).

Details

`Dickens.spc` has a frequency spectrum derived from a collection of Dickens' works downloaded from the Gutenberg archive (A Christmas Carol, David Copperfield, Dombey and Son, Great Expectations, Hard Times, Master Humphrey's Clock, Nicholas Nickleby, Oliver Twist, Our Mutual Friend, Sketches by BOZ, A Tale of Two Cities, The Old Curiosity Shop, The Pickwick Papers, Three Ghost Stories). `Dickens.emp.vgc` contains the corresponding observed vocabulary growth (`V` and `V(1)`).

`DickensOliverTwist.spc` and `DickensOliverTwist.emp.vgc` contain spectrum and observed growth curve (`V` and `V(1)` of the early novel Oliver Twist (1837-1839).

`DickensGreatExpectations.spc` and `DickensGreatExpectations.emp.vgc` contain spectrum and observed growth curve (`V` and `V(1)`) of the late novel Great Expectations (1860-1861).

`DickensOurMutualFriend.spc` and `DickensOurMutualFriend.emp.vgc` contain spectrum and observed growth curve (`V` and `V(1)`) of Our Mutual Friend, the last novel completed by Dickens (1864-1865).

Notice that we removed numbers and other forms of non-linguistic material before collecting the frequency data.

References

Project Gutenberg: `http://www.gutenberg.org/`

Charles Dickens on Wikipedia: `http://en.wikipedia.org/wiki/Charles_Dickens`

Examples

```
data(Dickens.spc)
summary(Dickens.spc)

data(Dickens.emp.vgc)
summary(Dickens.emp.vgc)

data(DickensOliverTwist.spc)
summary(DickensOliverTwist.spc)

data(DickensOliverTwist.emp.vgc)
summary(DickensOliverTwist.emp.vgc)
```

`EV-EVm` *Expected Frequency Spectrum (zipfR)*

Description

`EV` and `EVm` are generic methods for computing the expected vocabulary size $E[V]$ and frequency spectrum $E[V_m]$ according to a LNRE model (i.e. an object belonging to a subclass of `lnre`).

When applied to a frequency spectrum (i.e. an object of class `spc`), these methods perform binomial interpolation (see `EV.spc` for details), although `spc.interp` and `vgc.interp` might be more convenient binomial interpolation functions for most purposes.

Usage

```
EV(obj, N, ...)
EVm(obj, m, N, ...)
```

Arguments

`obj`	an LNRE model (i.e. an object belonging to a subclass of `lnre`) or frequency spectrum (i.e. an object of class `spc`)
`m`	positive integer value determining the frequency class m to be returned (or a vector of such values)
`N`	sample size N for which the expected vocabulary size and frequency spectrum are calculated (or a vector of sample sizes)
`...`	additional arguments passed on to the method implementation (see respective manpages for details)

Value

`EV` returns the expected vocabulary size $E[V(N)]$ in a sample of N tokens, and `EVm` returns the expected spectrum elements $E[V_m(N)]$, according to the LNRE model given by `obj` (or according to binomial interpolation).

See Also

See lnre for more information on LNRE models, a listing of available models, and methods for parameter estimation.

The variances of the random variables $V(N)$ and $V_m(N)$ can be computed with the methods VV and VVm.

See EV.spc and EVm.spc for more information about the usage of these methods to perform binomial interpolation (but consider using spc.interp and vgc.interp instead).

Examples

```
## see lnre() documentation for examples
```

EV-EVm.spc *Binomial Interpolation (zipfR)*

Description

Compute the expected vocabulary size $E[V(N)]$ (with function EV.spc) or expected frequency spectrum $E[V_m(N)]$ (with function EVm.spc) for a random sample of size N from a given frequency spectrum (i.e., an object of class spc). The expectations are calculated by binomial interpolation (following Baayen 2001, pp. 64-69).

Note that these functions are not user-visible. They can be called implicitly through the generic methods EV and EVm, applied to an object of type spc.

Usage

```
## S3 method for class 'spc':
EV(obj, N, allow.extrapolation=FALSE, ...)

## S3 method for class 'spc':
EVm(obj, m, N, allow.extrapolation=FALSE, ...)
```

Arguments

obj	an object of class spc, representing a frequency spectrum
m	positive integer value determining the frequency class m for which $E[V_m(N)]$ be returned (or a vector of such values)
N	sample size N for which the expected vocabulary size or frequency spectrum are calculated (or a vector of sample sizes)
allow.extrapolation	if TRUE, the requested sample size N may be larger than the sample size of the frequency spectrum obj, for binomial *extrapolation*. This obtion should be used with great caution (see "Details" below).
...	additional arguments passed on from generic methods will be ignored

Details

These functions are naive implementations of binomial interpolation, using Equations (2.41) and (2.43) from Baayen (2001). No guarantees are made concerning their numerical accuracy, especially for extreme values of m and N.

According to Baayen (2001), pp. 69-73., the same equations can also be used for binomial *extrapolation* of a given frequency spectrum to larger sample sizes. However, they become numerically unstable in this case and will typically break down when extrapolating to more than twice the size of the observed sample (Baayen 2001, p. 75). Therefore, extrapolation has to be enabled explicitly with the option `allow.extrapolation=TRUE` and should be used with great caution.

Value

`EV` returns the expected vocabulary size $E[V(N)]$ for a random sample of N tokens from the frequency spectrum `obj`, and `EVm` returns the expected spectrum elements $E[V_m(N)]$ for a random sample of N tokens from `obj`, calculated by binomial interpolation.

References

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

See Also

`EV` and `EVm` for the generic methods and links to other implementations

`spc.interp` and `vgc.interp` are convenience functions that compute an expected frequency spectrum or vocabulary growth curve by binomial interpolation

---

`ItaPref` *Italian Ri- and Ultra- Prefix Frequency Data (zipfR)*

Description

`ItaRi.spc` and `ItaRi.emp.vgc` are `zipfR` objects of classes `tfl`, `spc` and `vgc`, respectively. They contain frequency data for all verbal lemmas with the prefix ri- (similar to English re-) in the Italian la Repubblica corpus.

`ItaUltra.spc` and `ItaUltra.emp.vgc` contain the same kinds of data for the adjectival prefix ultra-.

Details

`ItaRi.emp.vgc` and `ItaUltra.emp.vgc` are *empirical* vocabulary growth curves, reflecting the `V` and `V(1)` development in the non-randomized corpus.

The data were manually checked, as described for ri- in Baroni (to appear).

References

Baroni, M. (to appear) I sensi di ri-: Un'indagine preliminare. In Maschi, R., Penello, N. and Rizzolatti, P. (eds.), *Miscellanea di studi linguistici offerti a Laura Vanelli*. Udine, Forum.

la Repubblica corpus: http://sslmit.unibo.it/repubblica/

Examples

```
data(ItaRi.spc)
summary(ItaRi.spc)

data(ItaRi.emp.vgc)
summary(ItaRi.emp.vgc)

data(ItaUltra.spc)
summary(ItaUltra.spc)

data(ItaUltra.emp.vgc)
summary(ItaUltra.emp.vgc)
```

`N-V-Vm` *Access Methods for Observed Frequency Data (zipfR)*

Description

`N`, `V` and `Vm` are generic methods that can (and should) be used to access observed frequency data for objects of class `tfl`, `spc`, `vgc` and `lnre`. The precise behaviour of the functions depends on the class of the object, but in general `N` returns the sample size, `V` the vocabulary size, and `Vm` one or more selected elements of the frequency spectrum.

Usage

```
N(obj, ...)
V(obj, ...)
Vm(obj, m, ...)
```

Arguments

`obj`	an object of class `tfl` (type frequency list), `spc` (frequency spectrum), `vgc` (vocabulary growth curve) or `lnre` (LNRE model)
`m`	positive integer value determining the frequency class m to be returned (or a vector of such values).
`...`	additional arguments passed on to the method implementation (see respective manpages for details)

Details

For `tfl` and `vgc` objects, the `Vm` method allows only a single value `m` to be specified.

Value

For a frequency spectrum (class `spc`), `N` returns the sample size, `V` returns the vocabulary size, and `Vm` returns individual spectrum elements.

For a type frequency list (class `tfl`), `N` returns the sample size and `V` returns the vocabulary size corresponding to the list. `Vm` returns a single spectrum element from the corresponding frequency spectrum, and may only be called with a single value `m`.

For a vocabulary growth curve (class `vgc`), `N` returns the vector of sample sizes and `V` the vector of vocabulary sizes. `Vm` may only be called with a single value `m` and returns the corresponding vector from the `vgc` object (if present).

For a LNRE model (class `lnre`) estimated from an observed frequency spectrum, the methods `N`, `V` and `Vm` return information about this frequency spectrum.

See Also

For details on the implementations of these methods, see `N.tfl`, `N.spc`, `N.vgc`, etc. When applied to an LNRE model, the methods return information about the observed frequency spectrum from which the model was estimated, so the manpages for `N.spc` are relevant in this case.

Expected vocabulary size and frequency spectrum for a sample of size N according to a LNRE model can be computed with the analogous methods `EV` and `EVm`. The corresponding variances are obtained with the `VV` and `VVm` methods, which can also be applied to expected or interpolated frequency spectra and vocabulary growth curves.

Examples

```
## load Brown spc and tfl
data(Brown.spc)
data(Brown.tfl)

## you can extract N, V and Vm (for a specific m)
## from either structure
N(Brown.spc)
N(Brown.tfl)

V(Brown.spc)
V(Brown.tfl)

Vm(Brown.spc,1)
Vm(Brown.tfl,1)

## you can extract the same info also from a lnre model estimated
## from these data (NB: these are the observed quantities; for the
## expected values predicted by the model use EV and EVm instead!)
model <- lnre("gigp",Brown.spc)
N(model)
V(model)
Vm(model,1)

## Baayen's P:
Vm(Brown.spc,1)/N(Brown.spc)

## when input is a spectrum (and only then) you can specify a vector
## of m's; e.g., to obtain class sizes of first 5 spectrum elements
## you can write:
```

```
Vm(Brown.spc,1:5)

## the Brown vgc
data(Brown.emp.vgc)

## with a vgc as input, N, V and Vm return vectors of the respective
## values for each sample size listed in the vgc
Ns <- N(Brown.emp.vgc)
Vs <- V(Brown.emp.vgc)
V1s <- Vm(Brown.emp.vgc,1)

head(Ns)
head(Vs)
head(V1s)

## since the last sample size in Brown.emp.vgc
## corresponds to the full Brown, the last elements
## of the Ns, Vs and V1s vectors are the same as
## the quantities extracted from the spectrum and
## tfl:
Ns[length(Ns)]
Vs[length(Vs)]
V1s[length(V1s)]
```

`N-V-Vm.spc` *Access Methods for Frequency Spectra (zipfR)*

Description

Return the sample size (`N.spc`), vocabulary size (`V.spc`) and class sizes (`Vm.spc`) of the frequency spectrum represented by a `spc` object. For an expected spectrum with variance information, `VV.spc` returns the variance of the expected spectrum size and `VVm.spc` the variances of individual spectrum elements.

Note that these functions are not user-visible. They can be called implicitly through the generic methods `N`, `V`, `Vm`, `VV` and `VVm`, applied to an object of type `spc`.

Usage

```
## S3 method for class 'spc':
N(obj, ...)

## S3 method for class 'spc':
V(obj, ...)

## S3 method for class 'spc':
Vm(obj, m, ...)

## S3 method for class 'spc':
VV(obj, N=NA, ...)

## S3 method for class 'spc':
```

```
VVm(obj, m, N=NA, ...)
```

Arguments

`obj`	an object of class `spc`, representing an observed or expected frequency spectrum
`m`	positive integer value determining the frequency class m to be returned (or a vector of such values).
`N`	not applicable (this argument of the generic method is not used by the implementation for `spc` objects and must not be specified)
`...`	additional arguments passed on from generic method will be ignored

Details

`VV.spc` a `VVm.spc` will fail if the object `obj` is not an expected frequency spectrum with variance data.

For an incomplete frequency spectrum, `Vm.spc` (and `VVm.spc`) will return `NA` for all spectrum elements that are not listed in the object (i.e. for `m > m.max`).

Value

`N.spc` returns the sample size N, `V.spc` returns the vocabulary size V (or expected vocabulary size $E[V]$), and `Vm.spc` returns a vector of class sizes V_m (ot the expected spectrum elements $E[V_m]$).

For an expected spectrum with variances, `VV.spc` returns the variance $Var[V]$ of the expected vocabulary size, and `VVm.spc` returns variances $Var[V_m]$ of the spectrum elements.

See Also

`N`, `V`, `Vm`, `VV`, `VVm` for the generic methods and links to other implementations

`spc` for details on frequency spectrum objects and links to other relevant functions

---

`N-V-Vm.tfl` *Access Methods for Type Frequency Lists (zipfR)*

Description

Return the sample size (`N.tfl`) and vocabulary size (`V.tfl`) of the type frequency list represented by a `tfl` object, as well as class sizes (`Vm.tfl`) of the corresponding frequency spectrum.

Note that these functions are not user-visible. They can be called implicitly through the generic methods `N`, `V` and `Vm`, applied to an object of type `tfl`.

Usage

```
## S3 method for class 'tfl':
N(obj, ...)

## S3 method for class 'tfl':
V(obj, ...)

## S3 method for class 'tfl':
Vm(obj, m, ...)
```

Arguments

`obj`	an object of class `tfl`, representing an observed type frequency list
`m`	non-negative integer value determining the frequency class m to be returned
`...`	additional arguments passed on from generic method will be ignored

Details

Only a single value is allowed for m, which may also be 0. In order to obtain multiple class sizes V_m, convert the type frequency list to a frequency spectrum with `tfl2spc` first.

For an incomplete type frequency list, `Vm.tfl` will return `NA` if `m` is outside the range of listed frequencies (i.e. for `m < f.min` or `m > f.max`).

Value

`N.tfl` returns the sample size N, `V.tfl` returns the vocabulary size V (or expected vocabulary size $E[V]$), and `Vm.tfl` returns the number of types that occur exactly m times in the sample, i.e. the class size V_m.

See Also

`N`, `V`, `Vm` for the generic methods and links to other implementations

`tfl` for details on type frequency list objects and links to other relevant functions

---

`N-V-Vm.vgc` *Access Methods for Vocabulary Growth Curves (zipfR)*

Description

Return the vector of sample sizes (`N.vgc`), vocabulary sizes (`V.vgc`) or class sizes (`Vm.vgc`) from the vocabulary growth curve (VGC) represented by a `vgc` object. For an expected or interpolated VGC with variance information, `VV.vgc` returns the vector of variances of the vocabulary size and `VVm.vgc` the variance vectors for individual spectrum elements.

Note that these functions are not user-visible. They can be called implicitly through the generic methods `N`, `V`, `Vm`, `VV` and `VVm`, applied to an object of type `vgc`.

Usage

```
## S3 method for class 'vgc':
N(obj, ...)

## S3 method for class 'vgc':
V(obj, ...)

## S3 method for class 'vgc':
Vm(obj, m, ...)

## S3 method for class 'vgc':
VV(obj, N=NA, ...)

## S3 method for class 'vgc':
VVm(obj, m, N=NA, ...)
```

Arguments

`obj`	an object of class `vgc`, representing an observed, interpolated or expected VGC
`m`	positive integer value determining the frequency class m for which the vector of class sizes is returned
`N`	not applicable (this argument of the generic method is not used by the implementation for `vgc` objects and must not be specified)
`...`	additional arguments passed on from generic method will be ignored

Details

`VV.vgc` a `VVm.vgc` will fail if the object `obj` does not include variance data. `Vm.vgc` and `VVm.vgc` will fail if the selected frequency class is not included in the VGC data.

Value

`N.vgc` returns the vector of sample sizes N, `V.vgc` returns the corresponding vocabulary sizes $V(N)$ (or expected vocabulary sizes $E[V(N)]$), and `Vm.vgc` returns the vector of class sizes $V_m(N)$ (or the expected spectrum elements $E[V_m(N)]$) for the selected frequency class m.

For an expected or interpolated VGC with variance information, `VV.vgc` returns the vector of variances $Var[V(N)]$ of the expected vocabulary size, and `VVm.vgc` returns vector of variances $Var[V_m(N)]$ for the selected frequency class m.

See Also

`N`, `V`, `Vm`, `VV`, `VVm` for the generic methods and links to other implementations

`vgc` for details on vocabulary growth curve objects and links to other relevant functions

Tiger — *Tiger NP and PP expansions (zipfR)*

Description

Objects of classes tfl, spc and vgc that contain frequency data for the syntactic expansions of Noun Phrases (NP) and Prepositional Phrases (PP) in the Tiger German treebank.

Details

In this dataset, types are not words, but syntactic expansions, i.e., sequences of syntactic categories that form NPs (in TigerNP) or PPs (in TigerPP), according to the Tiger annotation scheme for German. Thus, for example, among the expansion types in the TigerNP dataset, we find ART_NN and ART_ADJA_NN, whereas among the PP expansions in TigerPP we find APPR_ART_NN and APPR_NN (APPR is the tag for prepositions in the Tiger tagset).

The Tiger treebank contains about 900,000 tokens (50,000 sentences) of German newspaper text from the Frankfurter Rundschau. The token frequencies of the expansion types are taken from this corpus.

TigerNP.tfl and TigerPP.tfl are the type frequency lists. TigerNP.spc and TigerPP.spc are frequency spectra. TigerNP.emp.vgc and TigerPP.emp.vgc are the corresponding observed vocabulary growth curves (tracking the development of V and V(1) in the original order of occurrence of the expansion tokens in the source corpus).

References

Tiger Project: http://www.ims.uni-stuttgart.de/projekte/TIGER

Examples

```
data(TigerNP.tfl)
summary(TigerNP.tfl)

data(TigerNP.spc)
summary(TigerNP.spc)

data(TigerNP.emp.vgc)
summary(TigerNP.emp.vgc)

data(TigerPP.tfl)
summary(TigerPP.tfl)

data(TigerPP.spc)
summary(TigerPP.spc)

data(TigerPP.emp.vgc)
summary(TigerPP.emp.vgc)
```

VV-Vm *Variances of the Expected Frequency Spectrum (zipfR)*

Description

VV and VVm are generic methods that can (and should) be used to compute the variance of the vocabulary size and the variances of spectrum elements according to an LNRE model (i.e. an object of class lnre). These methods are also used to access variance information stored in some objects of class spc and vgc.

Usage

```
VV(obj, N=NA, ...)
VVm(obj, m, N=NA, ...)
```

Arguments

obj	an object of class lnre (LNRE model), spc (frequency spectrum) or vgc (vocabulary growth curve).
m	positive integer value determining the frequency class m for which variances are returned (or a vector of such values).
N	sample size N for which variances are calculated (lnre objects only)
...	additional arguments passed on to the method implementation (see respective manpages for details)

Details

spc and vgc objects must represent an expected or interpolated frequency spectrum or VGC, and must include variance data.

For vgc objects, the VVm method allows only a single value m to be specified.

The argument N is only allowed for LNRE models and will trigger an error message otherwise.

Value

For a LNRE model (class lnre), VV computes the variance of the random variable $V(N)$ (vocabulary size), and VVm computes the variance of the random variables $V_m(N)$ (spectrum elements), for a sample of specified size N.

For an observed or interpolated frequency spectrum (class spc), VV returns the variance of the expected vocabulary size, and VVm returns variances of the spectrum elements. These methods are only applicable if the spc object includes variance information.

For an expected or interpolated vocabulary growth curve (class vgc), VV returns the variance vector of the expected vocabulary sizes V, and VVm the corresponding vector for V_m. These methods are only applicable if the vgc object includes variance information.

See Also

For details on the implementations of these methods, see VV.spc, VV.vgc, etc.

Expected vocabulary size and frequency spectrum for a sample of size N according to a LNRE model can be computed with the analogous methods EV and EVm. For spc and vgc objects, V and V_m are always accessed with the methods V and Vm, even if they represent expected or interpolated values.

Examples

```
## see lnre documentation for examples
```

---

beta_gamma *Incomplete Beta and Gamma Functions (zipfR)*

Description

The functions documented here compute incomplete and regularized Beta and Gamma functions as well as their logarithms and the corresponding inverse functions. These functions will be of interest to developers, not users of the toolkit.

Usage

```
  Cgamma(a, log=!missing(base), base=exp(1))
  Igamma(a, x, lower=TRUE, log=!missing(base), base=exp(1))
  Igamma.inv(a, y, lower=TRUE, log=!missing(base), base=exp(1))
  Rgamma(a, x, lower=TRUE, log=!missing(base), base=exp(1))
  Rgamma.inv(a, y, lower=TRUE, log=!missing(base), base=exp(1))

  Cbeta(a, b, log=!missing(base), base=exp(1))
  Ibeta(x, a, b, lower=TRUE, log=!missing(base), base=exp(1))
  Ibeta.inv(y, a, b, lower=TRUE, log=!missing(base), base=exp(1))
  Rbeta(x, a, b, lower=TRUE, log=!missing(base), base=exp(1))
  Rbeta.inv(y, a, b, lower=TRUE, log=!missing(base), base=exp(1))
```

Arguments

a, b	non-negative numeric vectors, the parameters of the Gamma and Beta functions (b applies only to Beta functions)
x	a non-negative numeric vector, the point at which the incomplete or regularized Gamma or Beta function is evaluated (for the Beta functions, x must be in the range $[0, 1]$
y	a non-negative numeric vector, the values of the Gamma or Beta function on linear or logarithmic scale
lower	whether to compute the lower (TRUE) or upper (FALSE) incomplete or regularized Gamma or Beta function

`log`	if `TRUE`, return values of the Gamma and Beta functions – as well as the `y` argument of the inverse functions – are on logarithmic scale
`base`	a positive number, specifying the base of the logarithmic scale for values of the Gamma and Beta functions (default: natural logarithm). Setting the `base` parameter implies `log=TRUE`.

Value

`Cgamma` returns the (complete) Gamma function evaluated at `a`, $\Gamma(a)$. `Igamma` returns the (lower or upper) incomplete Gamma function with parameter `a` evaluated at point `x`, $\gamma(a, x)$ (`lower=TRUE`) or $\Gamma(a, x)$ (`lower=FALSE`). `Rgamma` returns the corresponding regularized Gamma function, $P(a, x)$ (`lower=TRUE`) or $Q(a, x)$ (`lower=FALSE`). If `log=TRUE`, the returned values are on logarithmic scale as specified by the `base` parameter.

`Igamma.inv` and `Rgamma.inv` compute the inverse of the incomplete and regularized Gamma functions with respect to the parameter `x`. I.e., `Igamma.inv(a,y)` returns the point `x` at which the (lower or upper) incomplete Gamma function with parameter `a` evaluates to `y`, and *mutatis mutandis* for `Rgamma.inv(a,y)`. If `log=TRUE`, the parameter `y` is taken to be on a logarithmic scale as specified by `base`.

`Cbeta` returns the (complete) Beta function with arguments `a` and `b`, $B(a, b)$. `Ibeta` returns the (lower or upper) incomplete Beta function with parameters `a` and `b`, evaluated at point `x`, $B(x; a, b)$ (`lower=TRUE`) and $B^*(x; a, b)$ (`lower=FALSE`). Note that in contrast to the Gamma functions, capital B refers to the *lower* incomplete Beta function, and there is no standardized notation for the upper incomplete Beta function, so B^* is used here as an ad-hoc symbol. `Rbeta` returns the corresponding regularized Beta function, $I(x; a, b)$ (`lower=FALSE`) or $I^*(x; a, b)$ (`lower=TRUE`). If `log=TRUE`, the returned values are on logarithmic scale as specified by the `base` parameter.

`Ibeta.inv` and `Rbeta.inv` compute the inverse of the incomplete and regularized Beta functions with respect to the parameter `x`. I.e., `Ibeta.inv(y,a,b)` returns the point `x` at which the (lower or upper) incomplete Beta function with parameters `a` and `b` evaluates to `y`, and *mutatis mutandis* for `Rbeta.inv(y,a,b)`. If `log=TRUE`, the parameter `y` is taken to be on a logarithmic scale as specified by `base`.

All Gamma and Beta functions can be vectorized in the arguments `x`, `y`, `a` and `b`, with the usual R value recycling rules in the case of multiple vectorizations.

Mathematical 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

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

The complete Gamma function calculates the full Gamma integral, i.e. $\Gamma(a) = \gamma(a, 0)$. 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

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

The lower incomplete Beta function is defined by the Beta integral

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

and the upper incomplete Beta function is defined by the complementary integral

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

The complete Beta function calculates the full Beta integral, i.e. $B(a,b) = B(1;a,b) = B^*(0;a,b)$. The regularized Beta function scales the incomplete Beta function to the interval $[0,1]$, by dividing through $B(a,b)$. The lower regularized Beta function is thus given by

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

and the upper regularized Beta function is given by

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

See Also

`gamma` and `lgamma`, which are fully equivalent to `Cgamma`. `beta` and `lbeta`, which are fully equivalent to `Cbeta`

The implementations of the incomplete and regularized Gamma functions are based on the Gamma distribution (see `pgamma`), and those of the Beta functions are based on the Beta distribution (see `pbeta`).

Examples

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

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

n <- 49
k <- 6
1 / ((n+1) * Cbeta(n-k+1, k+1)) # == choose(n, k)

## P(X >= k) for binomial distribution with parameters n and p
n <- 100
p <- .1
k <- 15
Rbeta(p, k, n-k+1) # == pbinom(k-1, n, p, lower=FALSE)
```

LNRE *Type and Probability Distributions of LNRE Models (zipfR)*

Description

Type density g (`tdlnre`), type distribution G (`tplnre`), type quantiles G^{-1} (`tqlnre`), probability density f (`dlnre`), distribution function F (`plnre`), quantile function F^{-1} (`qlnre`), logarithmic type and probability densities (`ltdlnre` and `ldlnre`), and random sample generation (`rlnre`) for LNRE models.

Usage

```
tdlnre(model, x, ...)
tplnre(model, q, lower.tail=FALSE, ...)
tqlnre(model, p, lower.tail=FALSE, ...)

dlnre(model, x, ...)
plnre(model, q, lower.tail=TRUE, ...)
qlnre(model, p, lower.tail=TRUE, ...)

ltdlnre(model, x, base=10, log.x=FALSE, ...)
ldlnre(model, x, base=10, log.x=FALSE, ...)

rlnre(model, n, ...)
```

Arguments

`model`	an object belonging to a subclass of `lnre`, representing a LNRE model
`x`	vector of type probabilities pi for which the density function is evaluated
`q`	vector of type probability quantiles, i.e. threshold values ρ on the type probability axis
`p`	vector of tail probabilities
`lower.tail`	if `TRUE`, lower tail probabilities or type counts are returned / expected in the `p` argument. Note that the defaults differ for distribution function and type distribution, and see "Details" below.
`base`	positive number, the base with respect to which the log-transformation is peformed (see "Details" below)
`log.x`	if `TRUE`, the values passed in the argument `x` are assumed to be logarithmic, i.e. $\log_a \pi$
`n`	size of random sample to generate. If `length(n) > 1`, the length is taken to be the number required.
`...`	further arguments are passed through to the method implementations (currently unused)

Details

Note that the order in which arguments are specified differs from the analogous functions for common statistical distributions in the R standard library. In particular, the LNRE model `model` always has to be given as the first parameter so that R can dispatch the function call to an appropriate method implementation for the chosen LNRE model.

Some of the functions may not be available for certain types of LNRE models. In particular, no analytical solutions are known for the distribution and quantiles of GIGP models, so the functions `tplnre`, `tqlnre`, `plnre`, `qlnre` and `rlnre` (which depends on `qlnre` and `tplnre`) are not implemented for objects of class `lnre.gigp`.

The default tails differ for the distribution function (`plnre`, `qlnre`) and the type distribution (`tplnre`, `tqlnre`), in order to match the definitions of $F(\rho)$ and $G(\rho)$. While the distribution function defaults to lower tails (`lower.tail=TRUE`, corresponding to F and F^{-1}), the type distribution defaults to upper tails (`lower.tail=FALSE`, corresponding to G and G^{-1}).

Unlike for standard distriutions, logarithmic tail probabilities (`log.p=TRUE`) are not provided for the LNRE models, since here the focus is usually on the bulk of the distribution rather than on the extreme tails.

The log-transformed density functions f^* and g^* returned by `ldlnre` and `ltdlnre`, respectively, can be understood as probability and type densities for $\log_a \pi$ instead of π, and are useful for visualization of LNRE populations (with a logarithmic scale for the parameter π on the x-axis). For example,

$$G(\log_a \rho) = \int_{\log_a \rho}^{0} g^*(t)\, dt$$

Value

For `rnlre`, a factor of length `n`, representing a random sample from the population described by the specified LNRE model.

For all other functions, a vector of non-negative numbers of the same length as the second argument (`x`, `p` or `q`).

`tdlnre` returns the type density $g(\pi)$ for the values of π specified in the vector `x`. `tplnre` returns the type distribution $G(\rho)$ (default) or its complement $1 - G(\rho)$ (if `lower.tail=TRUE`), for the values of ρ specified in the vector `q`. `tqlnre` returns type quantiles, i.e. the inverse $G^{-1}(x)$ (default) or $G^{-1}(S - x)$ (if `lower.tail=TRUE`) of the type distribution, for the type counts x specified in the vector `p`.

`dlnre` returns the probability density $f(\pi)$ for the values of π specified in the vector `x`. `plnre` returns the distribution function $F(\rho)$ (default) or its complement $1-F(\rho)$ (if `lower.tail=FALSE`), for the values of ρ specified in the vector `q`. `qlnre` returns quantiles, i.e. the inverse $F^{-1}(p)$ (default) or $F^{-1}(1 - p)$ (if `lower.tail=FALSE`) of the distribution function, for the probabilities p specified in the vector `p`.

`ldlnre` and `ltdlnre` compute logarithmically transformed versions of the probability and type density functions, respectively, taking logarithms with respect to the base a specified in the `base` argument (default: $a = 10$). See "Details" above for more information.

See Also

`lnre` for more information about LNRE models and how to initialize them

random samples generated with `rnlre` can be further processed with the functions `vec2tfl`, `vec2spc` and `vec2vgc`

Examples

```
## define ZM and fZM LNRE models
ZM <- lnre("zm", alpha=.8, B=1e-3)
FZM <- lnre("fzm", alpha=.8, A=1e-5, B=.05)

## random samples from the two models
head(table(rlnre(ZM, 10000)))
head(table(rlnre(FZM, 10000)))

## plot logarithmic type density functions
x <- 10^seq(-6, 1, by=.01)  # pi = 10^(-6) .. 10^(-1)
y.zm <- ltdlnre(ZM, x)
y.fzm <- ltdlnre(FZM, x)
## Not run: plot(x, y.zm, type="l", lwd=2, col="red", log="x", ylim=c(0,14000))
## Not run: lines(x, y.fzm, lwd=2, col="blue")
## Not run: zipfR.legend(2, legend=c("ZM", "fZM"), lwd=3, col=c("red", "blue"))

## probability pi_k of k-th type according to FZM model
k <- 10
plnre(FZM, tqlnre(FZM, k-1)) - plnre(FZM, tqlnre(FZM, k))

## number of types with pi >= 1e-6
tplnre(ZM, 1e-6)

## lower tail fails for infinite population size
## Not run: tplnre(ZM, 1e-3, lower=TRUE)

## total probability mass assigned to types with pi <= 1e-6
plnre(ZM, 1e-6)
```

`estimate.model` *Estimate LNRE Model Parameters (zipfR)*

Description

Internal function: Generic method for estimation of LNRE model parameters. Based on the class of its first argument, the method dispatches to a suitable implementation of the estimation procedure.

Unless you are a developer working on the `zipfR` source code, you are probably looking for the `lnre` manpage.

Usage

```
estimate.model(model, spc, param.names,
               method, cost.function, m.max=15,
               debug=FALSE, ...)
```

Arguments

`model`	LNRE model object of the appropriate class (a subclass of `lnre`). All parameters of the LNRE model that are not listed in `param.names` must have been initialized to their prespecified values in the `model` object.
`spc`	an observed frequency spectrum, i.e. an object of class `spc`. The values of the missing parameters will be estimated from this frequency spectrum.
`param.names`	a character vector giving the names of parameters for which values have to be estimated ("missing" parameters)
`method`	name of the minimization algorithm used for parameter estimation (see `lnre` for details)
`cost.function`	cost function to be minimized (see `lnre` for details). **NB:** this is a direct reference to the function object rather than just the name of the cost function. Lookup of the appropriate cost function implementation is performed in the `lnre` constructor.
`m.max`	number of spectrum elements that will be used to compute the cost function (passed on to `cost.function`)
`debug`	if `TRUE`, some debugging and progress information will be printed during the estimation procedure
`...`	additional arguments are passed on and may be used by some implementations

Details

By default, `estimate.model` dispatches to a generic implementation of the estimation procedure that can be used with all types of LNRE models (`estimate.model.lnre`).

This generic implementation can be overridden for specific LNRE models, e.g. to calculate better init values or improve the estimation procedure in some other way. To provide a custom implementation for Zipf-Mandelbrot models (of class `lnre.zm`), for instance, it is sufficient to define the corresponding method implementation `estimate.model.lnre.zm`. If no custom implementation is provided but the user has selected the `Custom` method (which is the default), `estimate.model` falls back on `Nelder-Mead` for multi-dimensional minimization and `NLM` for one-dimensional minimization (where Nelder-Mead is considered to be unreliable).

Parmeter estimation is performed by minimization of the cost function passed in the `cost.function` argument (see `lnre` for details). Depending on the `method` argument, a range of different minimization algorithms can be used (see `lnre` for a complete listing). The minimization algorithm always operates on *transformed* parameter values, making use of the `transform` utility provided by LNRE models (see `lnre.details` for more information about utility functions). All parameters are initialized to 0 in the transformed scale, which should translate to sensible starting points.

Note that the `estimate.model` implementations *do not perform any error checking*. It is the responsibility of the caller to make sure that the arguments are sensible and complete. In particular, all model parameters that will not be estimated (i.e. are not listed in `param.names`) must have been initialized to their prespecified values in the `model` passed to the function.

Value

A modified version of `model`, where the missing parameters listed in `param.names` have been estimated from the observed frequency spectrum `spc`. In addition, goodness-of-fit information is added to the object.

See Also

The user-level function for estimating LNRE models is `lnre`. Its manpage also lists available cost functions and minimization algorithms.

The internal structure of `lnre` objects (representing LNRE models) is described on the `lnre.details` manpage, which also outlines the necessary steps for implementing a new LNRE model.

The minimization algorithms used are described in detail on the `nlm` and `optim` manpages from R's standard library.

`lnre` *LNRE Models (zipfR)*

Description

LNRE model constructor, returns an object representing a LNRE model with the specified parameters, or allows parameters to be estimated automatically from an observed frequency spectrum.

Usage

```
lnre(type=c("zm", "fzm", "gigp"),
     spc=NULL, debug=FALSE,
     cost=c("chisq", "linear", "smooth.linear", "mse", "exact"),
     m.max=15,
     method=c("Custom", "NLM", "Nelder-Mead", "SANN"),
     exact=TRUE, sampling=c("Poisson", "multinomial"),
     ...)
```

Arguments

`type`	class of LNRE model to use (see "LNRE Models" below)
`spc`	observed frequency spectrum used to estimate model parameters
`debug`	if `TRUE`, detailed debugging information will be printed during parameter estimation
`cost`	cost function for measuring the "distance" between observed and expected vocabulary size and frequency spectrum. Parameters are estimated by minimizing this cost function (see "Cost Functions" below for a listing of available cost functions).
`m.max`	number of spectrum elements considered by the cost function (see "Cost Functions" below for more information)
`method`	algorithm used for parameter estimation, by minimizing the value of the cost function (see "Parameter Estimation" below for details, and "Minimization Algorithms" for descriptions of the available algorithms)
`exact`	if `FALSE`, certain LNRE models will be allowed to use approximations when calculating expected values and variances, in order to improve performance and numerical stability. However, the computed values might be inaccurate or inconsistent in "extreme" situations: in particular, $E[V]$ might be larger than N when N is very small; $\sum_m E[V_m]$ can be larger than $E[V]$ at the same N; $\sum_m m \cdot E[V_m]$ can be larger than N

`sampling`	type of random sampling model to use. `Poisson` sampling is mathematically simpler and allows fast and robust calculations, while `multinomial` sampling is more accurate especially for very small samples. `Poisson` sampling is the default and should be unproblematic for sample sizes $N \geq 10000$. **NB:** The `multinomial` sampling option has not been implemented yet.
`...`	all further named arguments are interpreted as parameter values for the chosen LNRE model (see the respective manpages for names and descriptions of the model parameters)

Details

Currently, the following LNRE models are supported by the `zipfR` package:

The **Zipf-Mandelbrot (ZM)** LNRE model (see `lnre.zm` for details).

The **finite Zipf-Mandelbrot (fZM)** LNRE model (see `lnre.fzm` for details).

The **Generalized Inverse Gauss-Poisson (GIGP)** LNRE model (see `lnre.gigp` for details).

If explicit model parameters are specified in addition to an observed frequency spectrum `spc`, these parameters are fixed to the given values and are excluded from the estimation procedure. This feature can be useful if fully automatic parameter estimation leads to a poor or counterintuitive fit.

Value

An object of a suitable subclass of `lnre`, depending on the `type` argument (e.g. `lnre.fzm` for `type="fzm"`). This object represents a LNRE model of the selected type with the specified parameter values, or with parameter values estimated from the observed frequency spectrum `spc`.

The internal structure of `lnre` objects is described on the `lnre.details` manpage (intended for developers).

Parameter Estimation

Automatic parameter estimation for LNRE models is performed by matching the expected vocabulary size and frequency spectrum of the model against the observed data passed in the `spc` argument.

For this purpose, a *cost function* has to be defined as a measure of the "distance" between observed and expected frequency spectrum. Parameters are then estimated by applying a *minimization algorithm* in order to find those parameter values that lead to the smallest possible cost.

Parameter estimation is a crucial and often also quite critical step in the application of LNRE models. Depending on the shape of the observed frequency spectrum, the automatic estimation procedure may result in a poor and counter-intuitive fit, or may fail altogether.

Users can influence parameter estimation by choosing from a range of predefined cost functions and from several minimization algorithms, as described in the following sections. Some experimentation with the `cost`, `m.max` and `method` arguments will often help to resolve estimation failures and may result in a considerably better goodness-of-fit.

Cost Functions

The following cost functions are available and can be selected with the `cost` argument. All functions are based on the differences between observed and expected values for vocabulary size and the first elements of the frequency spectrum ($V_1, \ldots, V_m$, where m is given by the `m.max` argument):

chisq: cost function based on a simplified version of the multivariate chi-squared test for goodness-of-fit (assuming independence between the random variables V_m). This cost function usually achieves the best results in the goodness-of-fit evaluation and is used by default.

linear: linear cost function, which sums over the absolute differences between observed and expected values. This cost function puts more weight on fitting the vocabulary size and the first few elements of the frequency spectrum (where absolute differences are much larger than for higher spectrum elements).

smooth.linear: modified version of the linear cost function, which smoothes the kink of the absolute value function for a difference of 0 (since non-differentiable cost functions might be problematic for gradient-base minimization algorithms)

mse: mean squared error cost function, averaging over the squares of differences between observed and expected values. This cost function penalizes large absolute differences more heavily than linear cost (and therefore puts even greater weight on fitting vocabulary size and the first spectrum elements).

exact: this "virtual" cost function attempts to match the observed vocabulary size and first spectrum elements exactly, ignoring differences for all higher spectrum elements. This is achieved by adjusting the value of `m.max` automatically, depending on the number of free parameters that are estimated (in general, the number of constraints that can be satisfied by estimating parameters is the same as the number of free parameters). Having adjusted `m.max`, the `mse` cost function is used to determined parameter values, so that the estimation procedure will not fail even if the constraints cannot be matched exactly.

Minimization Algorithms

Several different minimization algorithms can be used for parmeter estimation and are selected with the `method` argument:

Custom: by default, a custom estimation procedure is used for each type of LNRE model, which may exploit special mathematical properties of the model in order to calculate one or more of the parameter values directly. For example, one parameter of the ZM and fZM models can easily be determined from the constraint $E[V] = V$ (but note that this additional constraint leads to a different fit than is obtained by plain minimization of the cost function!). Custom estimation might also apply special configuration settings to improve convergence of the minimization process, based on knowledge about the valid ranges and "behaviour" of model parameters. If no custom estimation procedure has been implemented for the selected LNRE model, `lnre` falls back on the `Nelder-Mead` or `NLM` algorithm.

NLM: a standard Newton-type algorithm for nonlinear minimization, implemented by the `nlm` function, which makes use of numerical derivatives of the cost function. `NLM` minimization converges quickly and obtains very precise parameter estimates (for a local minimum of the cost function), but it is not very stable and may cause parameter estimation to fail altogether.

Nelder-Mead: the Nelder-Mead algorithm, implemented by the `optim` function, performs minimization without using derivatives. Parameter estimation is therefore very robust, while almost as fast and accurate as the `NLM` method. `Nelder-Mead` is also used internally by most custom minimization algorithms.

SANN minimization by simulated annealing, also provided by the `optim` function. Like `Nelder-Mead`, this algorithm is very robust because it avoids numerical derivatives, but convergence is extremely slow. In some cases, `SANN` might produce a better fit than `Nelder-Mead` (if the latter converges to a suboptimal local minimum).

See the `nlm` and `optim` manpages for more information about the minimization algorithms used and key references.

See Also

Detailed descriptions of the different LNRE models provided by zipfR and their parameters can be found on the manpages lnre.zm, lnre.fzm and lnre.gigp.

Useful methods for trained models are lnre.spc, lnre.vgc, EV, EVm, VV, VVm. Suitable implementations of the print and summary methods are also provided (see print.lnre for details). Note that the methods N, V and Vm can be applied to LNRE models with estimated parameters and return information about the observed frequency spectrum used for parameter estimation.

The lnre.details manpage gives details about the implementation of LNRE models and the internal structure of lnre objects, while estimate.model has more information on the parameter estimation procedure (both manpages are intended for developers).

See lnre.goodness.of.fit for a complete description of the goodness-of-fit test that is automatically performed after parameter estimation (and which is reported in the summary of the LNRE model). This function can also be used to evaluate the predictions of the LNRE model on a different data set than the one used for parameter estimation.

Examples

```
## load Dickens dataset
data(Dickens.spc)

## estimate parameters of GIGP model and show summary
m <- lnre("gigp", Dickens.spc)
m


## N, V and V1 of spectrum used to compute model
## (should be the same as for Dickens.spc)
N(m)
V(m)
Vm(m,1)


## expected V and V_m and their variances for arbitrary N
EV(m,100e6)
VV(m,100e6)
EVm(m,1,100e6)
VVm(m,1,100e6)

## use only 10 instead of 15 spectrum elements to estimate model
## (note how fit improves for V and V1)
m.10 <- lnre("gigp", Dickens.spc, m.max=10)
m.10

## experiment with different cost functions
m.mse <- lnre("gigp", Dickens.spc, cost="mse")
m.mse
m.exact <- lnre("gigp", Dickens.spc, cost="exact")
m.exact


## NLM minimization algorithm is faster but less robust
m.nlm <- lnre("gigp", Dickens.spc, method="NLM")
m.nlm
```

```
## ZM and fZM LNRE models have special estimation algorithms
m.zm <- lnre("zm", Dickens.spc)
m.zm
m.fzm <- lnre("fzm", Dickens.spc)
m.fzm


## estimation is much faster if approximations are allowed
m.approx <- lnre("fzm", Dickens.spc, exact=FALSE)
m.approx


## specify parameters of LNRE models directly
m <- lnre("zm", alpha=.5, B=.01)
lnre.spc(m, N=1000, m.max=10)

m <- lnre("fzm", alpha=.5, A=1e-6, B=.01)
lnre.spc(m, N=1000, m.max=10)

m <- lnre("gigp", gamma=-.5, B=.01, C=.01)
lnre.spc(m, N=1000, m.max=10)
```

`lnre.fzm` *The finite Zipf-Mandelbrot (fZM) LNRE Model (zipfR)*

Description

The finite Zipf-Mandelbrot (fZM) LNRE model of Evert (2004).

The constructor function `lnre.fzm` is not user-visible. It is invoked implicitly when `lnre` is called with LNRE model type `"fzm"`.

Usage

```
lnre.fzm(alpha=.8, A=1e-9, B=.01, param=list())

## user call: lnre("fzm", spc=spc) or lnre("fzm", alpha=.8, A=1e-9, B=.01)
```

Arguments

`alpha`	the *shape* parameter α, a number in the range $(0, 1)$
`A`	the *lower cutoff* parameter A, a positive number. Note that a valid set of parameters must satisfy $0 < A < B$.
`B`	the *upper cutoff* parameter B, a positive number ($B > 1$ is allowed although it is inconsistent with the interpretation of B)
`param`	a list of parameters given as name-value pairs (alternative method of parameter specification)

Details

The parameters of the fZM model can either be specified as immediate arguments:

```
lnre.fzm(alpha=.5, A=5e-12, B=.1)
```

or as a list of name-value pairs:

```
lnre.fzm(param=list(alpha=.5, A=5e-12, B=.1))
```

which is usually more convenient when the constructor is invoked by another function (such as `lnre`). If both immediate arguments and the `param` list are given, the immediate arguments override conflicting values in `param`. For any parameters that are neither specified as immediate arguments nor listed in `param`, the defaults from the function prototype are inserted.

The `lnre.fzm` constructor also checks the types and ranges of parameter values and aborts with an error message if an invalid parameter is detected.

NB: parameter estimation is faster and more robust for the inexact fZM model, so you might consider passing the `exact=FALSE` option to `lnre` unless you intend to make predictions for small sample sizes N and/or high spectrum elements $E[V_m(N)]$ ($m \gg 1$) with the model.

Value

A partially initialized object of class `lnre.fzm`, which is completed and passed back to the user by the lnre function. See `lnre` for a detailed description of `lnre.fzm` objects (as a subclass of `lnre`).

Mathematical Details

Similar to ZM, the **fZM model** is a LNRE re-formulation of the **Zipf-Mandelbrot** law for a population with a finite vocabulary size S, i.e.

$$\pi_k = \frac{C}{(k+b)^a}$$

for $k = 1, \ldots, S$. The parameters of the Zipf-Mandelbrot law are $a > 1$, $b \geq 1$ and S (see also Baayen 2001, 101ff). The fZM model is given by the **type density function**

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

for $A \leq \pi \leq B$ (and $\pi = 0$ otherwise), and has three **parameters** $0 < \alpha < 1$ and $0 < A < B \leq 1$. The normalizing constant is

$$C = \frac{1-\alpha}{B^{1-\alpha} - A^{1-\alpha}}$$

and the population vocabulary size is

$$S = \frac{1-\alpha}{\alpha} \cdot \frac{A^{-\alpha} - B^{-\alpha}}{B^{1-\alpha} - A^{1-\alpha}}$$

See Evert (2004) and the `lnre.zm` manpage for further details.

References

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

Evert, Stefan (2004). A simple LNRE model for random character sequences. *Proceedings of JADT 2004*, 411-422.

See Also

`lnre` for pointers to relevant methods and functions for objects of class `lnre`, as well as a complete listing of LNRE models implemented in the `zipfR` library.

`lnre.gigp` *The Generalized Inverse Gauss-Poisson (GIGP) LNRE Model (zipfR)*

Description

The Generalized Inverse Gauss-Poisson (GIGP) LNRE model of Sichel (1971).

The constructor function `lnre.gigp` is not user-visible. It is invoked implicitly when `lnre` is called with LNRE model type `"gigp"`.

Usage

```
lnre.gigp(gamma=-.5, B=.01, C=.01, param=list())

## user call: lnre("gigp", spc=spc) or lnre("gigp", gamma=-.5, B=.01, C=.01)
```

Arguments

`gamma`	the *shape* parameter γ, a negative number in the range $(-1, 0)$. γ corresponds to $-\alpha$ in the Zipf-Mandelbrot notation.
`B`	the *low-frequency decay* parameter b, a non-negative number. This parameter determines how quickly the type density function vanishes for $\pi \to 0$, with larger values corresponding to faster decay.
`C`	the *high-frequency decay* parameter c, a non-negative number. This parameter determines how quickly the type density function vanishes for large values of π, with *smaller* values corresponding to faster decay.
`param`	a list of parameters given as name-value pairs (alternative method of parameter specification)

Details

The parameters of the GIGP model can either be specified as immediate arguments:

```
lnre.gigp(gamma=-.47, B=.001, C=.001)
```

or as a list of name-value pairs:

```
lnre.gigp(param=list(gamma=-.47, B=.001, C=.001))
```

which is usually more convenient when the constructor is invoked by another function (such as `lnre`). If both immediate arguments and the `param` list are given, the immediate arguments override conflicting values in `param`. For any parameters that are neither specified as immediate arguments nor listed in `param`, the defaults from the function prototype are inserted.

The `lnre.gigp` constructor also checks the types and ranges of parameter values and aborts with an error message if an invalid parameter is detected.

Notice that the implementation of GIGP leads to numerical problems when estimating the expected frequency of high spectrum elements (you might start worrying if you need to go above $m = 150$).

Note that the parameters b and c are normally written in lowercase (e.g. Baayen 2001). For the technical reasons, it was necessary to use uppercase letters `B` and `C` in this implementation.

Value

A partially initialized object of class `lnre.gigp`, which is completed and passed back to the user by the lnre function. See `lnre` for a detailed description of `lnre.gigp` objects (as a subclass of `lnre`).

Mathematical Details

Despite its fance name, the **Generalized Inverse Gauss-Poisson** or **GIGP model** belongs to the same class of LNRE models as ZM and fZM. This class of models is characterized by a power-law in the type density function and derives from the **Zipf-Mandelbrot law** (see `lnre.zm` for details on the relationship between power-law LNRE models and the Zipf-Mandelbrot law).

The GIGP model is given by the type density function

$$g(\pi) := C \cdot \pi^{\gamma-1} \cdot e^{-\frac{\pi}{c} - \frac{b^2 c}{4\pi}}$$

with parameters $-1 < \gamma < 0$ and $b, c \geq 0$. The normalizing constant is

$$C = \frac{(2/bc)^{\gamma+1}}{K_{\gamma+1}(b)}$$

and the population vocabulary size is

$$S = \frac{2}{bc} \cdot \frac{K_{\gamma}(b)}{K_{\gamma+1}(b)}$$

Note that the "shape" parameter γ corresponds to $-\alpha$ in the ZM and fZM models. The GIGP model was introduced by Sichel (1971). See Baayen (2001, 89-93) for further details.

References

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

Sichel, H. S. (1971). On a family of discrete distributions particularly suited to represent long-tailed frequency data. *Proceedings of the Third Symposium on Mathematical Statistics*, 51-97.

See Also

`lnre` for pointers to relevant methods and functions for objects of class `lnre`, as well as a complete listing of LNRE models implemented in the `zipfR` library.

`lnre.goodness.of.fit` *Goodness-of-fit Evaluation of LNRE Models (zipfR)*

Description

This function measures the goodness-of-fit of a LNRE model compared to an observed frequency spectrum, using a multivariate chi-squared test (Baayen 2001, p. 119ff).

Usage

```
lnre.goodness.of.fit(model, spc, n.estimated=0, m.max=15)
```

Arguments

`model`	an LNRE model object, belonging to a suitable subclass of `lnre`.
`spc`	an observed frequency spectrum, i.e. an object of class `spc`. This can either be the spectrum on which the model parameters have been estimated, or a different, independent spectrum.
`n.estimated`	number of parameters of the LNRE model that have been estimated on `spc`. This number is automatically subtracted from the degrees of freedom of the resulting chi-squared statistic. When `spc` is an independent spectrum, `n.estimated` should always be set to the default value of 0.
`m.max`	number of spectrum elements that will be used to compute the chi-squared statistic. The default value of 15 is also used by Baayen (2001). For small samples, it may be sensible to use fewer spectrum elements, e.g. by setting `m.max=10` or `m.max=5`. Depending on how many degrees of freedom have to be subtracted, `m.max` should not be chosen too low.

Details

By default, the number of spectrum elements included in the calculation of the chi-squared statistic may be reduced automatically in order to ensure that it is not dominated by the sampling error of spectrum elements with very small expected frequencies (which are scaled up due to the small variance of these random variables). As an ad-hoc rule of thumb, spectrum elements V_m with variance less than 5 are excluded, since the normal approximation to their discrete distribution is likely to be inaccurate in this case.

Automatic reduction is disabled when the parameter `m.max` is specified explicitly (use `m.max=15` to disable automatic reduction without changing the default value).

Value

A data frame with one row and the following variables:

`X2`	value of the multivariate chi-squared statistic X^2
`df`	number of degrees of freedom of X^2, corrected for the number of parameters that have been estimated on `spc`
`p`	p-value corresponding to X^2

References

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

See Also

`lnre` for more information about LNRE models

Examples

```
## load spectrum of first 100k Brown tokens
data(Brown100k.spc)

## use this spectrum to compute zm and gigp
## models
zm <- lnre("zm",Brown100k.spc)
gigp <- lnre("gigp",Brown100k.spc)

## lnre.goodness.of.fit with appropriate
## n.estimated value produces the same multivariate
## chi-squared test that is reported in a model
## summary

## compare:
zm
lnre.goodness.of.fit(zm,Brown100k.spc,n.estimated=2)

gigp
lnre.goodness.of.fit(gigp,Brown100k.spc,n.estimated=3)

## goodness of fit of the 100k models calculated on the
## whole Brown spectrum (although this is superset of
## 100k spectrum, let's pretend it is an independent
## spectrum, and set n.estimated to 0)

data(Brown.spc)

lnre.goodness.of.fit(zm,Brown.spc,n.estimated=0)
lnre.goodness.of.fit(gigp,Brown.spc,n.estimated=0)
```

`lnre.spc` *Compute Expected Frequency Spectrum of LNRE Model (zipfR)*

Description

`lnre.spc` computes the expected frequency spectrum of a LNRE model at specified sample size `N`, returning an object of class `spc`. Since almost all expected spectrum elements are non-zero, only an incomplete spectrum can be generated.

Usage

```
lnre.spc(model, N=NULL, variances=FALSE, m.max=100)
```

Arguments

`model`	an object belonging to a subclass of `lnre`, representing a LNRE model
`N`	a single positive integer, specifying the sample size N for which the expected frequency spectrum is calculated (defaults to same sample size as used for estimating the model)
`variances`	if `TRUE`, include variances for the spectrum elements in the `spc` object
`m.max`	number of spectrum elements listed in the frequency spectrum. The default of 100 is chosen to avoid numerical problems that certain LNRE models (in particular, GIGP) have for higher m. If variance data is included, the default value is automatically reduced to 50.

Details

TODO, if any

Value

An object of class `spc`, representing the incomplete expected frequency spectrum of the LNRE model `lnre` at sample size `N`. If `variances=TRUE`, the spectrum also includes variance data.

See Also

`spc` for more information about frequency spectra and links to relevant functions; `lnre` for more information about LNRE models and how to initialize them

Examples

```
## load Dickens dataset and compute lnre models
data(Dickens.spc)

zm <- lnre("zm",Dickens.spc)
fzm <- lnre("fzm",Dickens.spc, exact=FALSE)
gigp <- lnre("gigp",Dickens.spc)

## calculate the corresponding expected
## frequency spectra at the Dickens size
zm.spc <- lnre.spc(zm,N(Dickens.spc))
fzm.spc <- lnre.spc(fzm,N(Dickens.spc))
gigp.spc <- lnre.spc(gigp,N(Dickens.spc))

## comparative plot
plot(Dickens.spc,zm.spc,fzm.spc,gigp.spc,m.max=10)

## expected spectra at N=100e+8
## and comparative plot
zm.spc <- lnre.spc(zm,1e+8)
fzm.spc <- lnre.spc(fzm,1e+8)
gigp.spc <- lnre.spc(gigp,1e+8)

plot(zm.spc,fzm.spc,gigp.spc,m.max=10)

## with variances
zm.spc <- lnre.spc(zm,1e+8,variances=TRUE)
```

```
head(zm.spc)

## asking for more than 50 spectrum elements
## (increasing m.max will eventually lead
## to error, at different threshold for
## the different models)
zm.spc <- lnre.spc(zm,1e+8,m.max=1000)
fzm.spc <- lnre.spc(fzm,1e+8,m.max=1000)
gigp.spc <- lnre.spc(gigp,1e+8,m.max=100) ## gigp breaks first!
```

lnre.details *Technical Details of LNRE Model Objects (zipfR)*

Description

This manpage describes technical details of LNRE models and parameter estimation. It is intended developers who want to implement new LNRE models, improve the parameter estimation algorithms, or work directly with the internals of `lnre` objects. All information required for standard applications of LNRE models can be found on the `lnre` manpage.

Details

Most operations on LNRE models (in particular, computation of expected values and variances, distribution function and type distribution, random sampling, etc.) are realized as S3 methods, so they are automatically dispatched to appropriate implementations for the various types of LNRE models (e.g., `EV.lnre.zm`, `EV.lnre.fzm` and `EV.lnre.gigp` for the `EV` method). For some methods (e.g. estimated variances `VV` and `VVm`), a single generic implementation can be used for all model types, provided through the base class (`VV.lnre` and `VVm.lnre` for variances).

If you want to implement new LNRE models, have a look at "Implementing LNRE Models" below.

Important note: LNRE model parameters can be passed as named arguments to the `lnre` constructor function when they are not estimated automatically from an observed frequency spectrum. For this reason, parameter names must be carefully chosen so that they do not clash with other arguments of the `lnre` function. Note that because of R's argument matching rules, any parameter name that is a *prefix* of a standard argument name will lead to such a clash. In particular, single-letter parameters (such as b and c for the GIGP model) should always be written in uppercase (`B` and `C` in `lnre.gigp`).

Value

A LNRE model with estimated (or manually specified) parameter values is represented by an object belonging to a suitable subclass of `lnre`. The specific class depends on the type of LNRE model, as specified in the `type` argument to the `lnre` constructor function (e.g. `lnre.fzm` for a fZM model selected with `type="fzm"`).

All subtypes of `lnre` object share the same data format, viz. a list with the following components:

`type`	a character string specifying the class of LNRE model, e.g. `"fzm"` for a finite Zipf-Mandelbrot model
`name`	a character string specifying a human-readable name for the LNRE model, e.g. `"finite Zipf-Mandelbrot"`
`param`	list of named model parameters, e.g. `(alpha=.8, B=.01)` for a ZM model

`param2`	a list of "secondary" parameters, i.e. constants that can be determined from the model parameters but are frequently used in the formulae for expected values, variances, etc.; e.g. `(C=.5)` for the ZM model above
`S`	population size, i.e. number of types in the population described by the LNRE model (may be `Inf`, e.g. for a ZM model)
`exact`	whether approximations are allowed when calculating expectations and variances (`FALSE`) or not (`TRUE`)
`multinomial`	whether to use equations for multionmial sampling (`TRUE`) or independent Poisson sampling (`FALSE`)
`spc`	an object of class `spc`, the observed frequency spectrum from which the model parameters have been estimated (only if the LNRE model is based on empirical data)
`gof`	an object of class `lnre.gof` with goodness-of-fit information for the estimated LNRE model (only if based on empirical data, i.e. if the `spc` component is also present)
`util`	a set of utility functions, given as a list with the following components:

`update`: function with signature `(self, param, transformed=FALSE)`, which updates the parameters of the LNRE model `self` with the values in `param`, checks that their values are in the allowed range, and re-calculates "secondary" parameters and lexicon size if necessary. If `transformed=TRUE`, the specified parameters are translated back to normal scale before the update (see below). Of course, `self` should be the object from which the utility function was called. `update` returns a modified version of the object `self`.

`transform`: function with signature `(param, inverse=FALSE)`, which transform model parameters (given as a list in the argument `param`) to an unbounded range centered at 0, and back (with option `inverse=TRUE`). The transformed model parameters are used for parameter estimation, so that unconstrained minimization algorithms can be applied. The link function for the transformation depends on the LNRE model and the "distribution" of each parameter. A felicitous choice can be crucial for robust and quick parameter estimation, especially with Newton-like gradient algorithms. Note that setting all transformed parameters to 0 should provide a reasonable starting point for the parameter estimation.

`print`: partial print method for this subclass of LNRE model, which displays the name of the model, its parameters, and optionally some additional information (invoked internally by `print.lnre` and `summary.lnre`)

Implementing LNRE Models

In order to implement a new class of LNRE models, the following steps are necessary (illustrated on the example of a lognormal type density function, introducing the new LNRE class `lnre.lognormal`):

- Provide a constructor function for LNRE models of this type (here, `lnre.lognormal`), which must accept the parameters of the LNRE model as named arguments with reasonable default values (or alternatively as a list passed in the `param` argument). The constructor must return a partially initialized object of an appropriate subclass of `lnre` (`lnre.lognormal` in our example), and make sure that this object also inherits from the `lnre` class.
- Provide the `update`, `transform` and `print` utility functions for the LNRE model, which must be returned in the `util` field of the LNRE model object (see "Value" above).

- Add the new type of LNRE model to the `type` argument of the generic `lnre` constructor, and insert the new constructor function (`lnre.lognormal`) in the `switch` call in the body of `lnre`.
- As a minimum requirement, implementations of the `EV` and `EVm` methods must be provided for the new LNRE model (in our example, they will be named `EV.lnre.lognormal` and `EVm.lnre.lognormal`).
- If possible, provide equations for the type density, probability density, type distribution and distribution function of the new LNRE model, as implementations of the `tdlnre`, `dlnre`, `tplnre`/`tqlnre` and `plnre`/`qlnre` methods for the new LNRE model class. If all these functions are defined, log-scaled densities and random number generation are automatically handled by generic implementations.
- Optionally, provide a custom function for parameter estimation of the new LNRE model, as an implementation of the `estimate.model` method (here, `estimate.model.lnre.lognormal`). Custom parameter estimation can considerably improve convergence and goodness-of-fit if it is possible to obtain direct estimates for one or more of the parameters, e.g. from the condition $E[V] = V$. However, the default Nelder-Mead algorithm is robust and produces satisfactory results, as long as the LNRE model defines an appropriate parameter transformation mapping. It is thus often more profitable to optimize the `transform` utility than to spend a lot of time implementing a complicated parameter estimation function.

The best way to get started is to take a look at one of the existing implementations of LNRE models. The GIGP model represents a "minimum" implementation (without custom parameter estimation and distribution functions), whereas ZM and fZM provide good examples of custom parameter estimation functions.

See Also

User-level information about LNRE models and parameter estimation can be found on the `lnre` manpage.

Descriptions of the different LNRE models implemented in `zipfR` and their parameters are given on separate manpages `lnre.zm`, `lnre.fzm` and `lnre.gigp`. These descriptions are intended for interested end users, but are not required for standard applications of the models.

The `estimate.model` manpage explains details of the parameter estimation procedure (intended for developers).

See `lnre.goodness.of.fit` for a description of the goodness-of-fit test performed after parameter estimation of an LNRE model. This function can also be used to evaluate the predictions of the model on a different data set.

`lnre.vgc` *Expected Vocabulary Growth Curves of LNRE Model (zipfR)*

Description

`lnre.vgc` computes expected vocabulary growth curves $E[V(N)]$ according to a LNRE model, returning an object of class `vgc`. Data points are returned for the specified values of N, optionally including estimated variances and/or growth curves for the spectrum elements $E[V_m(N)]$.

Usage

```
lnre.vgc(model, N, m.max=0, variances=FALSE)
```

Arguments

`model`	an object belonging to a subclass of `lnre`, representing a LNRE model
`N`	an increasing sequence of non-negative integers, specifying the sample sizes N for which vocabulary growth data should be calculated
`m.max`	if specified, include vocabulary growth curves $E[V_m(N)]$ for spectrum elements up to `m.max`. Must be a single integer in the range $1 \ldots 9$.
`variances`	if `TRUE`, include variance estimates for the vocabulary size (and the spectrum elements, if applicable)

Details

TODO, if any

Value

An object of class `vgc`, representing the expected vocabulary growth curve $E[V(N)]$ of the LNRE model `lnre`, with data points at the sample sizes `N`.

If `m.max` is specified, expected growth curves $E[V_m(N)]$ for spectrum elements (*hapax legomena*, *dis legomena*, etc.) up to `m.max` are also computed.

If `variances=TRUE`, the `vgc` object includes variance data for all growth curves.

See Also

`vgc` for more information about vocabulary growth curves and links to relevant functions; `lnre` for more information about LNRE models and how to initialize them

Examples

```
## load Dickens dataset and estimate lnre models
data(Dickens.spc)

zm <- lnre("zm",Dickens.spc)
fzm <- lnre("fzm",Dickens.spc,exact=FALSE)
gigp <- lnre("gigp",Dickens.spc)

## compute expected V and V_1 growth up to 100 million tokens
## in 100 steps of 1 million tokens
zm.vgc <- lnre.vgc(zm,(1:100)*1e6, m.max=1)
fzm.vgc <- lnre.vgc(fzm,(1:100)*1e6, m.max=1)
gigp.vgc <- lnre.vgc(gigp,(1:100)*1e6, m.max=1)

## compare
plot(zm.vgc,fzm.vgc,gigp.vgc,add.m=1,legend=c("ZM","fZM","GIGP"))

## load Italian ultra- prefix data
data(ItaUltra.spc)
```

```
## compute zm model
zm <- lnre("zm",ItaUltra.spc)

## compute vgc up to about twice the sample size
## with variance of V
zm.vgc <- lnre.vgc(zm,(1:100)*70, variances=TRUE)

## plot with confidence intervals derived from variance in
## vgc (with larger datasets, ci will typically be almost
## invisible)
plot(zm.vgc)
```

`lnre.zm` *The Zipf-Mandelbrot (ZM) LNRE Model (zipfR)*

Description

The Zipf-Mandelbrot (ZM) LNRE model of Evert (2004).

The constructor function `lnre.zm` is not user-visible. It is invoked implicitly when `lnre` is called with LNRE model type `"zm"`.

Usage

```
lnre.zm(alpha=.8, B=.01, param=list())

## user call: lnre("zm", spc=spc) or lnre("zm", alpha=.8, B=.1)
```

Arguments

`alpha`	the *shape* parameter α, a number in the range $(0, 1)$
`B`	the *upper cutoff* parameter B, a positive number ($B > 1$ is allowed although it is inconsistent with the interpretation of B)
`param`	a list of parameters given as name-value pairs (alternative method of parameter specification)

Details

The parameters of the ZM model can either be specified as immediate arguments:

```
lnre.zm(alpha=.5, B=.1)
```

or as a list of name-value pairs:

```
lnre.zm(param=list(alpha=.5, B=.1))
```

which is usually more convenient when the constructor is invoked by another function (such as `lnre`). If both immediate arguments and the `param` list are given, the immediate arguments override conflicting values in `param`. For any parameters that are neither specified as immediate arguments nor listed in `param`, the defaults from the function prototype are inserted.

The `lnre.zm` constructor also checks the types and ranges of parameter values and aborts with an error message if an invalid parameter is detected.

Value

A partially initialized object of class `lnre.zm`, which is completed and passed back to the user by the lnre function. See `lnre` for a detailed description of `lnre.zm` objects (as a subclass of `lnre`).

Mathematical Details

The **ZM model** is a re-formulation of the **Zipf-Mandelbrot** law

$$\pi_k = \frac{C}{(k+b)^a}$$

with parameters $a > 1$ and $b \geq 1$ (see also Baayen 2001, 101ff) as a LNRE model. It is given by the **type density function**

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

for $0 \leq \pi \leq B$ (and $\pi = 0$ otherwise), with the **parameters** $0 < \alpha < 1$ and $0 < B \leq 1$. The normalizing constant is

$$C = \frac{1-\alpha}{B^{1-\alpha}}$$

and the population vocabulary size is $S = \infty$. The parameters of the ZM model are related to those of the original Zipf-Mandelbrot law by $a = 1/\alpha$ and $b = (1-\alpha)/(B \cdot \alpha)$. See Evert (2004) for further details.

References

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

Evert, Stefan (2004). A simple LNRE model for random character sequences. *Proceedings of JADT 2004*, 411-422.

See Also

`lnre` for pointers to relevant methods and functions for objects of class `lnre`, as well as a complete listing of LNRE models implemented in the `zipfR` library.

plot.spc *Plot Word Frequency Spectra (zipfR)*

Description

Plot a word frequency spectrum, or a comparison of several word frequency spectra, either as a side-by-side barplot or as points and lines on various logarithmic scales.

Usage

```
## S3 method for class 'spc':
plot(x, y, ...,
     m.max=if (log=="") 15 else 50,
     log="", conf.level=.95,
     bw=zipfR.par("bw"), points=TRUE,
     xlim=NULL, ylim=NULL,
     xlab="m", ylab="V_m", legend=NULL,
     main="Frequency Spectrum",
     barcol=NULL, pch=NULL, lty=NULL, lwd=NULL, col=NULL)
```

Arguments

`x, y, ...`	one or more objects of class `spc`, representing observed or expected frequency spectra to be plotted
`m.max`	number of frequency classes that will be shown in plot. The default is 15 on linear scale and 50 when using any type of logarithmic scale.
`log`	a character string specifying the axis or axes for which logarithmic scale is to be used (`"x"`, `"y"`, or `"xy"`), similar to the `log` argument of `plot.default`. By default, a barplot on linear scale is displayed. Use `log=""` to show non-logarithmic points-and-lines plot (also see "Details" below).
`conf.level`	confidence level for confidence intervals in logarithmic plots (see "Details" below). The default value of .95 produces 95%-confidence intervals. Set to `NA` in order to suppress confidence interval markers.
`bw`	if `TRUE`, draw plot in B/W style (default is the global `zipfR.par` setting)
`points`	if `TRUE`, spectrum plots on any type of logarithmic scale are drawn as overplotted lines and points (default). Otherwise, they are drawn as lines with different styles.
`xlim, ylim`	visible range on x- and y-axis. The default values are automatically determined to fit the selected data in the plot.
`xlab, ylab`	labels for the x-axis and y-axis. The default values nicely typeset mathematical expressions. The y-axis label also distinguishes between observed and expected frequency spectra.
`main`	a character string or expression specifying a main title for the plot
`legend`	optional vector of character strings or expressions, specifying labels for a legend box, which will be drawn in the upper right-hand corner of the screen. If `legend` is given, its length must correspond to the number of frequency spectra in the plot.

`barcol, pch, lty, lwd, col`
: style vectors that can be used to override the global styles defined by `zipfR.par`. If these vectors are specified, they must contain at least as many elements as there are frequency spectra in the plot: the values are *not* automatically recycled.

Details

By default, the frequency spectrum or spectra are represented as a barplot, with both axes using linear scale. If the `log` parameter is given, the spectra are shown either as lines in different styles (`points=FALSE`) or as overplotted points and lines (`point=TRUE`). The value of `log` specifies which axes should use logarithmic scale (specify `log=""` for a points-and-lines plot on linear scale).

In y-logarithmic plots, frequency classes with $V_m = 0$ are drawn outside the plot region (below the bottom margin) rather than skipped.

In all logarithmic plots, confidence intervals are indicated for expected frequency spectra with variance data (by vertical lines with T-shaped hooks at both ends). The size of the confidence intervals is controlled by the `conf.level` parameter (default: 95%). Set `conf.level=NA` in order to suppress the confidence interval indicators.

Line and point styles, as well as bar colours in the barplot, can be defined globally with `zipfR.par`. They can be overridden locally with the optional parameters `barcol`, `pch`, `lty`, `lwd` and `col`, but this should only be used when absolutely necessary. In most cases, it is more advisable to change the global settings temporarily for a sequence of plots.

The `bw` parameter is used to switch between B/W and colour modes. It can also be set globally with `zipfR.par`.

See Also

`spc`, `lnre`, `lnre.spc`, `plot.vgc`, `zipfR.par`, `zipfR.plotutils`

Examples

```
## load Italian ultra- prefix data
data(ItaUltra.spc)

## plot spectrum
plot(ItaUltra.spc)

## logarithmic scale for m (more elements are plotted)
plot(ItaUltra.spc,log="x")

## just lines
plot(ItaUltra.spc,log="x",points=FALSE)

## just the first five elements, then the first 100
plot(ItaUltra.spc,m.max=5)
plot(ItaUltra.spc,m.max=100,log="x")

## compute zm model and expeccted spectrum
zm <- lnre("zm",ItaUltra.spc)
zm.spc <- lnre.spc(zm,N(ItaUltra.spc))

## compare observed and expected spectra (also
## in black and white to print on papers)
plot(ItaUltra.spc,zm.spc,legend=c("observed","expected"))
plot(ItaUltra.spc,zm.spc,legend=c("observed","expected"),bw=TRUE)
plot(ItaUltra.spc,zm.spc,legend=c("observed","expected"),log="x")
plot(ItaUltra.spc,zm.spc,legend=c("observed","expected"),log="x",bw=TRUE)

## re-generate expected spectrum with variances
zm.spc <- lnre.spc(zm,N(ItaUltra.spc),variances=TRUE)

## now 95
plot(zm.spc,log="x")

## different title and labels
plot(zm.spc,log="x",main="Expected Spectrum with Confidence Interval",xlab="spectrum elem
```

`plot.vgc` *Plot Vocabulary Growth Curves (zipfR)*

Description

Plot a vocabulary growth curve (i.e., $V(N)$ or $V_m(N)$ against N), or a comparison of several vocabulary growth curves.

Usage

```
## S3 method for class 'vgc':
plot(x, y, ...,
     m=NA, add.m=NULL, N0=NULL,
     conf.level=.95, conf.style=c("ticks", "lines"),
     log=c("", "x", "y", "xy"),
     bw=zipfR.par("bw"),
     xlim=NULL, ylim=NULL,
     xlab="N", ylab="V(N)", legend=NULL,
     main="Vocabulary Growth",
     lty=NULL, lwd=NULL, col=NULL)
```

Arguments

`x, y, ...`	one or more objects of class `vgc`, representing observed or expected vocabulary growth curves to be plotted
`m`	a single integer m in the range $1 \ldots 9$. If specified, graphs will be plotted for $V_m(N)$ instead of $V(N)$ (the default). Note that all `vgc` objects to be plotted must contain the necessary data in this case.
`add.m`	a vector of integers in the range $1 \ldots 9$. If specified, graphs for $V_m(N)$ will be added as thin lines to the default $V(N)$ curve, for all specified frequency classes m. This option cannot be combined with the `m` option above. See "Details" below.
`N0`	if specified, draw a dashed vertical line at $N = N_0$, indicating the sample size where a LNRE model has been estimated (this is never done automatically)

`log`	a character string specifying the axis or axes for which logarithmic scale is to be used (`"x"`, `"y"`, or `"xy"`), similar to the `log` argument of `plot.default`. By default, both axes use linear scale (also see "Details" below).
`conf.level`	confidence level for confidence intervals around expected vocabulary growth curves (see "Details" below). The default value of .95 produces 95%-confidence intervals. Set to `NA` in order to suppress confidence interval markers.
`conf.style`	if `"ticks"`, confidence intervals are indicated by vertical lines at each data point in the `vgc` object (default). If `"lines"`, confidence intervals are indicated by thin curves above and below the VGC (which may be difficult to see when plotting multiple VGCs). Notice that confidence intervals might be so narrow as to be invisible in plots (one way to visualize them in such case might be to set an extremely conservative confidence level, such as .9999).
`bw`	if `TRUE`, draw plot in B/W style (default is the global `zipfR.par` setting)
`xlim, ylim`	visible range on x- and y-axis. The default values are automatically determined to fit the selected data in the plot.
`xlab, ylab`	labels for the x-axis and y-axis. The default values nicely typeset mathematical expressions. The y-axis label also distinguishes between observed and expected vocabulary growth curves, as well as between $V(N)$ and $V_m(N)$.
`main`	a character string or expression specifying a main title for the plot
`legend`	optional vector of character strings or expressions, specifying labels for a legend box, which will be drawn in the lower right-hand corner of the screen. If `legend` is given, its length must correspond to the number of VGCs in the plot.
`lty, lwd, col`	style vectors that can be used to override the global styles defined by `zipfR.par`. If these vectors are specified, they must contain at least as many elements as there are VGCs in the plot: the values are *not* automatically recycled.

Details

By default, standard vocabulary growth curves are plotted for all specified `vgc` objects, i.e. graphs of $V(N)$ against N. If `m` is specified, growth curves for hapax legomena or other frequency classes are shown instead, i.e. graphs of $V_m(N)$ against N. In this case, all `vgc` objects must contain the necessary data for $V_m(N)$.

Alternatively, the option `add.m` can be used to display growth curves for one or more spectrum elements *in addition* to the standard VGCs. These growth curves are plotted as thinner lines, otherwise matching the styles of the main curves. Since such plots can become fairly confusing and there is no finer control over the styles of the additional curves, it is generally not recommended to make use of the `add.m` option.

Confidence intervals are indicated for expected vocabulary growth curves with variance data, either by short vertical lines (`conf.style="ticks"`, the default) or by thin curves above and below the main growth curve (`conf.style="lines"`). The size of the confidence intervals is controlled by the `conf.level` parameter (default: 95%). Set `conf.level=NA` in order to suppress the confidence interval indicators.

In y-logarithmic plots, data points with $V(N) = 0$ or $V_m(N) = 0$ are drawn outside the plot region (below the bottom margin) rather than skipped.

Line and point styles can be defined globally with `zipfR.par`. They can be overridden locally with the optional parameters `lty`, `lwd` and `col`, but this should only be used when absolutely necessary. In most cases, it is more advisable to change the global settings temporarily for a sequence of plots.

The `bw` parameter is used to switch between B/W and color modes. It can also be set globally with `zipfR.par`.

See Also

`vgc`, `lnre`, `lnre.vgc`, `plot.spc`, `zipfR.par`, `zipfR.plotutils`

Examples

```
## load Our Mutual Friend spectrum and empirical vgc
data(DickensOurMutualFriend.emp.vgc)
data(DickensOurMutualFriend.spc)

## plot empirical V and V1 growth
plot(DickensOurMutualFriend.emp.vgc,add.m=1)

## use log scale for y-axis
plot(DickensOurMutualFriend.emp.vgc,add.m=1,log="y")

## binomially interpolated vgc at same points as
## empirical vgc
omf.bin.vgc <- vgc.interp(DickensOurMutualFriend.spc,N(DickensOurMutualFriend.emp.vgc))

## compare empirical and interpolated vgc, also with
## thinner lines, and in black and white
plot(DickensOurMutualFriend.emp.vgc,omf.bin.vgc,legend=c("observed","interpolated"))
plot(DickensOurMutualFriend.emp.vgc,omf.bin.vgc,legend=c("observed","interpolated"),lwd=c
plot(DickensOurMutualFriend.emp.vgc,omf.bin.vgc,legend=c("observed","interpolated"),bw=TR

## load Great Expectations spectrum and use it to
## compute ZM model
data(DickensGreatExpectations.spc)
ge.zm <- lnre("zm",DickensGreatExpectations.spc)

## expected V of Great Expectations at sample
## sizes of OMF's interpolated vgc
ge.zm.vgc <- lnre.vgc(ge.zm,N(omf.bin.vgc))

## compare interpolated OMF Vs and inter/extra-polated
## GE Vs, with a vertical line at sample size
## used to compute GE model
plot(omf.bin.vgc,ge.zm.vgc,N0=N(ge.zm),legend=c("OMF","GE"))

## load Italian ultra- prefix data and compute zm model
data(ItaUltra.spc)
ultra.zm <- lnre("zm",ItaUltra.spc)

## compute vgc up to about twice the sample size
## with variance of V
ultra.zm.vgc <- lnre.vgc(ultra.zm,(1:100)*70, variances=TRUE)

## plot with confidence intervals derived from variance in
## vgc (with larger datasets, ci will typically be almost
## invisible)
plot(ultra.zm.vgc)
```

```
## use more conservative confidence level, and plot
## the intervals as lines
plot(ultra.zm.vgc,conf.level=.99,conf.style="lines")

## suppress ci plotting, and insert different title and labels
plot(ultra.zm.vgc,conf.level=NA,main="ultra-",xlab="sample sizes",ylab="types")

## load Brown adjective spectrum
## (about 80k tokens)
data(BrownAdj.spc)

## binomially interpolated curve of V and V_1 to V_5
BrownAdj.bin.vgc <- vgc.interp(BrownAdj.spc,(1:100)*800,m.max=5)

## plot with V and 5 spectrum elements
plot(BrownAdj.bin.vgc,add.m=c(1:5))
```

`print.lnre` *Printing LNRE Models (zipfR)*

Description

Implementations of the `print` and `summary` methods for LNRE models (subclasses of `lnre`).

Usage

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

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

Arguments

`x, object`	an object of class `lnre` or one of its subclasses, representing a LNRE model
`...`	other arguments passed on from generic method will be ignored

Details

NB: implementation details and format of the summary are subject to change in future releases

In the current implementation, `print` and `summary` produce the same output for LNRE models.

This summary comprises the type of LNRE model, its parameter values, derived parameters such as normalization constants, and the population size S.

If the model parameters have been estimated from an observed frequency spectrum, a comparison of the observed and expected frequency spectrum is shown, including goodness-of-fit statistics.

Value

`NULL`

Unlike other implementations of the `summary` method, `summary.lnre` only prints a summary on screen and does not return a special "summary" object.

See Also

See the `lnre` manpage for more information on LNRE models.

Examples

```
# load Brown verbs dataset and estimate lnre models
data(BrownVer.spc)
zm <- lnre("zm",BrownVer.spc)
fzm <- lnre("fzm",BrownVer.spc,exact=FALSE)
gigp <- lnre("gigp",BrownVer.spc)

# look at summaries with either summary or print
summary(zm)
print(zm)

summary(fzm)
print(fzm)

summary(gigp)
print(gigp)
```

`print.spc` *Printing Frequency Spectra (zipfR)*

Description

Implementations of the `print` and `summary` methods for frequency spectrum objects (of class `spc`).

Usage

```
## S3 method for class 'spc':
print(x, all=FALSE, ...)

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

Arguments

`x, object`	an object of class `spc`, representing a frequency spectrum
`all`	if `FALSE`, only the first ten non-empty frequency classes will be shown (default)
`...`	other arguments passed on from generic method will be ignored

Details

NB: implementation details and format of the summary are subject to change in future releases

`print.spc` works similar to the standard `print` method for data frames, but provides additional information about N and V. Unless `all` is set to `TRUE`, only the first ten non-zero spectrum elements will be shown.

`summary.spc` gives a concise summary of the most important information about the frequency spectrum. In addition to N V, the first spectrum elements are shown. The summary will also indicate whether the spectrum is incomplete, an expected spectrum, or has variances (but does not show the variances).

Value

`NULL`

Unlike other implementations of the `summary` method, `summary.spc` only prints a summary on screen and does not return a special "summary" object.

See Also

See the `spc` manpage for details on `spc` objects.

Examples

```
## load Brown verbs dataset
data(BrownVer.spc)

## look at summary and print BrownVer.spc
summary(BrownVer.spc)
print(BrownVer.spc)

## print all non-zero spectrum elements
print(BrownVer.spc,all=TRUE)

## estimate zm model and construct expected spectrum with
## variances
zm <- lnre("zm",BrownVer.spc)
zm.spc <- lnre.spc(zm,N(zm),variances=TRUE)

## summary and print for the expected spectrum
summary(zm.spc)
print(zm.spc)
```

`print.tfl` *Printing Type Frequency Lists (zipfR)*

Description

Implementations of the `print` and `summary` methods for type frequency list objects (of class `tfl`).

Usage

```
## S3 method for class 'tfl':
print(x, all=FALSE, ...)

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

Arguments

`x, object`	an object of class `tfl`, representing a type frequency list
`all`	if `FALSE`, only the twenty most frequent types will be shown (default)
`...`	other arguments passed on from generic method will be ignored

Details

NB: implementation details and format of the summary are subject to change in future releases

`print.tfl` works similar to the standard `print` method for data frames, but provides additional information about N and V. Unless `all` is set to `TRUE`, only the twenty most frequent types will be shown.

`summary.tfl` gives a concise summary of the most important information about the type frequency list. In addition to showing N V, the summary also indicates whether the list is incomplete and shows examples of type representations (if present).

Value

`NULL`

Unlike other implementations of the `summary` method, `summary.tfl` only prints a summary on screen and does not return a special "summary" object.

See Also

See the `tfl` manpage for details on `tfl` objects.

Examples

```
## load Brown tfl
data(Brown.tfl)

## summary and print most frequent types
summary(Brown.tfl)
print(Brown.tfl)

## the whole type list (don't try
## this unless you have some time to
## spare)
## Not run: print(Brown.tfl,all=TRUE)
```

`print.vgc` *Printing Vocabulary Growth Curves (zipfR)*

Description

Implementations of the `print` and `summary` methods for vocabulary growth curve objects (of class `vgc`).

Usage

```
## S3 method for class 'vgc':
print(x, all=FALSE, ...)

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

Arguments

`x, object`	an object of class `vgc`, representing a vocabulary growth curve
`all`	if `FALSE`, vocabulary growth data are shown for at most 25 sample sizes (default)
`...`	other arguments passed on from generic method will be ignored

Details

NB: implementation details and format of the summary are subject to change in future releases

`print.vgc` calls the standard `print` method for data frames internally, but reduces the data set randomly to show at most 25 sample sizes (unless `all=TRUE`).

`summary.vgc` gives a concise summary of the available vocabulary growth data in the `vgc` object, including the number and range of sample sizes, whether spectrum elements are included, and whether variances are included.

Value

`NULL`

Unlike other implementations of the `summary` method, `summary.vgc` only prints a summary on screen and does not return a special "summary" object.

See Also

See the `vgc` manpage for details on `vgc` objects.

Examples

```
## load Brown "informative" prose empirical vgc
data(BrownInform.emp.vgc)

## summary, print (random subset) and print all
summary(BrownInform.emp.vgc)
print(BrownInform.emp.vgc)
print(BrownInform.emp.vgc,all=TRUE)

## load Brown informative prose spectrum
## and get estimate a fzm model
data(BrownInform.spc)
fzm <- lnre("fzm",BrownInform.spc,exact=FALSE)

## obtain expected vgc up to 2M tokens
## with spectrum elements up to V_3
## and variances
fzm.vgc <- lnre.vgc(fzm,(1:100)*2e+4,m.max=3,variances=TRUE)

## summary and print
summary(fzm.vgc)
print(fzm.vgc)
print(fzm.vgc,all=TRUE)
```

`read.multiple.objects` *Reading Multiple Objects from Files (zipfR)*

Description

`read.multiple.objects` constructs a list of `spc`, `vgc` or `tfl` objects from a set of input text files in the specified directory

NB: This function is intended for users that want to run advanced experiments (e.g., handling hundreds of spectra generated in multiple randomizations experiments). For the standard one-object-at-a-time reading functionality, look at the documentation of `read.spc`, `read.vgc` and `read.tfl`

Usage

```
read.multiple.objects(directory, prefix, class=c("spc", "vgc", "tfl"))
```

Arguments

`directory`	character string specifying the directory where the target input files reside (absolute path, or path relative to current working directory)
`prefix`	character string specifying prefix that must be shared by all target input file names
`class`	one of `spc`, `vgc` or `tfl` as character string, specifying the class of object we are importing (see the manpages of `spc`, `vgc` and `tfl` for details)

Format

`read.multiple.objects` reads in all files matching the pattern `prefix.id.class` from the specified directory, where the `prefix` and `class` strings are passed as arguments, and `id` is an arbitrary string that is used as index of the corresponding object in the output list

`read.multiple.objects` calls the `read` function corresponding to the `class` argument. Thus, the input files must respect the formatting conventions of the relevant reading functions (see documentation of `read.spc`, `read.vgc` and `read.tfl`)

Value

`read.multiple.objects` returns a list of objects of the specified class; each object is indexed with the id extracted from the corresponding file name (see section "Format")

See Also

See the `spc`, `vgc` and `tfl` manpages for details on the corresponding objects; `read.spc`, `read.vgc` and `read.tfl` for the single-file reading functions and input format details

Examples

```
## Not run:

## examples will not be run during package compilation
## since they would require accessing external files

## suppose that the current working directory contains
## 100 spc files named: rand.1.spc, rand.2.spc, ...,
## rand.100.spc

## read the files in:
spc.list <- read.multiple.objects(".","rand","spc")

## you can access each spc using the id extracted from
## the file name, e.g.:
summary(spc.list[["1"]])

## more usefully, you might want to iterate over the
## whole list, e.g., to calculate mean V:
mean(sapply(spc.list,V))

## notice that ids are arbitrary strings
## e.g., suppose that directory /home/me/animals
## contains sounds.dog.vgc and sounds.elephant.vgc

## we read the vgcs in:
vgc.list <- read.multiple.objects("/home/me/animals","sounds","vgc")

## accessing the elephant vgc:
V(vgc.list[["elephant"]])

## of course, tfl-reading works in the same way (assuming
## that the animals directory also contains some tfl files):
tfl.list <- read.multiple.objects("/home/me/animals","sounds","tfl")
```

```
## End(Not run)
```

read.spc — *Loading and Saving Frequency Spectra (zipfR)*

Description

`read.spc` loads frequency spectrum from `.spc` file

`write.spc` saves frequency spectrum object in `.spc` file

Usage

```
read.spc(file)

write.spc(spc, file)
```

Arguments

`file`	character string specifying the pathname of a disk file. See section "Format" for a description of the required file format
`spc`	a frequency spectrum, i.e. an object of class `spc`

Format

A TAB-delimited text file with column headers but no row names (suitable for reading with `read.delim`). The file must contain at least the following two columns:

`m` frequency class m

`Vm` number V_m of types in frequency class m (or expected class size $E[V_m]$)

An optional column labelled `VVm` can be used to specify variances of expected class sizes (for a frequency spectrum derived from a LNRE model or by binomial interpolation).

These columns may appear in any order in the text file. All other columns will be silently ignored.

Details

The `.spc` file format does not store the values of `N`, `V` and `VV` explicitly. Therefore, incomplete frequency spectra and expected spectra with variances cannot be fully reconstructed from disk files. Saving such frequency spectra (or loading a spectrum with variance data) will trigger corresponding warnings.

Value

`read.spc` returns an object of class `spc` (see the `spc` manpage for details)

See Also

See the `spc` manpage for details on `spc` objects. See `read.tfl` and `read.vgc` for import/export of other data structures.

Examples

```
## Not run:

## examples will not be run during package compilation
## since they would require accessing and writing to
## external files

## load Italian ultra- data
## and write corresponding spectrum to external text file
data(ItaUltra.spc)
write.spc(ItaUltra.spc,"ultra.spc")
## now ultra.spc is a text file with columns m and Vm

## we ready it back in
New.spc <- read.spc("ultra.spc")

## same spectrum as ItaUltra.spc, compare:
summary(New.spc)
summary(ItaUltra.spc)

## DON'T do the following, incomplete spectrum will not be
## restored properly!!!
zm <- lnre("zm",ItaUltra.spc) # estimate model
zm.spc <- lnre.spc(zm,N(zm)) # incomplete spectrum from model
write.spc(zm.spc,"var.spc") # warnings
bad.spc <- read.spc("/Users/baroni/Desktop/var.spc")
## latter is DIFFERENT from zm.spc!!!

## End(Not run)
```

`read.tfl` *Loading and Saving Type Frequency Lists (zipfR)*

Description

`read.tfl` loads type frequency list from `.tfl` file

`write.tfl` saves type frequency list object in `.tfl` file

Usage

```
read.tfl(file)

write.tfl(tfl, file)
```

Arguments

`file`	character string specifying the pathname of a disk file. See section "Format" for a description of the required file format
`tfl`	a type frequency list, i.e. an object of class `tfl`

Format

A TAB-delimited text file with column headers but no row names (suitable for reading with `read.delim`), containing the following columns:

`f` type frequencies f_k

`k` optional: the corresponding type IDs k. If missing, increasing non-negative integers are automatically assigned as IDs.

`type` optional: type representations (such as word forms or lemmas)

These columns may appear in any order in the text file. Only the `f` column is mandatory and all unrecognized columns will be silently ignored.

Details

The `.tfl` file format stores neither the values of `N` and `V` nor the range of type frequencies explicitly. Therefore, incomplete type frequency lists cannot be fully reconstructed from disk files (and will not even be recognized as such). An attempt to save such a list will trigger a corresponding warning.

Value

`read.tfl` returns an object of class `tfl` (see the `tfl` manpage for details)

See Also

See the `tfl` manpage for details on `tfl` objects. See `read.spc` and `read.vgc` for import/export of other data structures.

Examples

```
## Not run:

## examples will not be run during package compilation
## since they would require accessing and writing to
## external files

## load Brown tfl and write it to external file
data(Brown.tfl)
write.tfl(Brown.tfl,"brown.tfl")
## now brown.tfl is external file with fields
## k (an id), f (frequency), type (word)

## read it back in
New.tfl <- read.tfl("brown.tfl")

## same as Brown.tfl
summary(New.tfl)
summary(Brown.tfl)
print(New.tfl)
print(Brown.tfl)
head(New.tfl)
head(Brown.tfl)

## suppose you have a text file with a
## frequency list, one f per line, e.g.:
## f
## 14
## 12
## 31
## ...

## you can import this with read.tfl
MyData.tfl <- read.tfl("mylist.txt")
summary(MyData.tfl)
print(MyData.tfl) # ids in column k added by zipfR

## from this you can generate a spectrum with tfl2spc
MyData.spc <- tfl2spc(MyData.tfl)
summary(MyData.spc)

## End(Not run)
```

`read.vgc` *Loading and Saving Vocabulary Growth Curves (zipfR)*

Description

`read.vgc` loads vocabulary growth data from `.vgc` file

`write.vgc` saves vocabulary growth data in `.vgc` file

Usage

```
read.vgc(file)

write.vgc(vgc, file)
```

Arguments

`file`	character string specifying the pathname of a disk file. See section "Format" for a description of the required file format
`vgc`	a vocabulary growth curve, i.e. an object of class `vgc`

Format

A TAB-delimited text file with column headers but no row names (suitable for reading with `read.delim`). The file must contain at least the following two columns:

N increasing integer vector of sample sizes N

V corresponding observed vocabulary sizes $V(N)$ or expected vocabulary sizes $E[V(N)]$

Optionally, columns `V1`, ..., `V9` can be added to specify the number of hapaxes ($V_1(N)$), dis legomena ($V_2(N)$), and further spectrum elements up to $V_9(N)$.

It is not necessary to include all 9 columns, but for any $V_m(N)$ in the data set, all "lower" spectrum elements $V_{m'}(N)$ (for $m' < m$) must also be present. For example, it is valid to have columns `V1 V2 V3`, but not `V1 V3 V5` or `V2 V3 V4`.

Variances for expected vocabulary sizes and spectrum elements can be given in further columns `VV` (for $Var[V(N)]$), and `VV1`, ..., `VV9` (for $Var[V_m(N)]$). `VV` is mandatory in this case, and columns `VVm` must be specified for exactly the same frequency classes `m` as the `Vm` above.

These columns may appear in any order in the text file. All other columns will be silently ignored.

Value

`read.vgc` returns an object of class `vgc` (see the `vgc` manpage for details)

See Also

See the `vgc` manpage for details on `vgc` objects. See `read.tfl` and `read.spc` for import/export of other data structures.

Examples

```
## Not run:

## examples will not be run during package compilation
## since they would require accessing and writing to
## external files

## load Italian ultra- prefix vgc
## and write to external text file
data(ItaUltra.emp.vgc)
write.vgc(ItaUltra.emp.vgc,"ultra.vgc")
## now ultra.vgc is a text file with columns N, V and V1

## we ready it back in
New.vgc <- read.vgc("ultra.vgc")

## same vgc as ItaUltra.emp.vgc, compare:
summary(New.vgc)
summary(ItaUltra.emp.vgc)
head(New.vgc)
head(ItaUltra.emp.vgc)

## End(Not run)
```

sample.spc — *Incremental Samples from a Frequency Spectrum (zipfR)*

Description

Compute incremental random samples from a frequency spectrum (an object of class `spc`).

Usage

```
sample.spc(obj, N, force.list=FALSE)
```

Arguments

`obj`	an object of class `spc`, representing a frequency spectrum
`N`	a vector of non-negative integers in increasing order, the sample sizes for which incremental samples will be generated
`force.list`	if `TRUE`, the return value will always be a list of `spc` objects, even if `N` is just a single integer

Details

This function is currently implemented as a wrapper around `sample.tfl`, using `spc2tfl` and `tfl2spc` to convert between frequency spectra and type frequency lists. A direct implementation might be slightly more efficient, but would very likely not make a substantial difference.

Value

If `N` is a single integer (and the `force.list` flag is not set), a `spc` object representing the frequency spectrum of a random sample of size N from `obj`.

If `N` is a vector of length greater one, or if `force.list=TRUE`, a list of `spc` objects representing the frequency spectra of incremental random samples of the specified sizes N. *Incremental* means that each sample is a superset of the preceding sample.

See Also

`spc` for more information about frequency spectra

`sample.tfl` is an analogous function for type frequency lists (objects of class `tfl`)

`sample.spc` takes a single *concrete* random subsample from a spectrum and returns the spectrum of the subsample, unlike `spc.interp`, that computes the *expected* frequency spectrum for random subsamples of size `N` by binomial interpolation.

Examples

```
## read Brown spectrum
data(Brown.spc)
summary(Brown.spc)

## sample a spectrum of 100k tokens
```

```
MiniBrown.spc <- sample.spc(Brown.spc,1e+5)
summary(MiniBrown.spc)

## if we repat, we get a different sample
MiniBrown.spc <- sample.spc(Brown.spc,1e+5)
summary(MiniBrown.spc)
```

`sample.tfl` *Incremental Samples from a Type Frequency List (zipfR)*

Description

Compute incremental random samples from a type frequency list (an object of class `tfl`).

Usage

```
sample.tfl(obj, N, force.list=FALSE)
```

Arguments

`obj`	an object of class `tfl`, representing a type frequency list
`N`	a vector of non-negative integers in increasing order, the sample sizes for which incremental samples will be generated
`force.list`	if `TRUE`, the return value will always be a list of `tfl` objects, even if `N` is just a single integer

Details

The current implementation is reasonably efficient, but will be rather slow when applied to very large type frequency lists.

Value

If `N` is a single integer (and the `force.list` flag is not set), a `tfl` object representing a random sample of size N from the type frequency list `obj`.

If `N` is a vector of length greater one, or if `force.list=TRUE`, a list of `tfl` objects representing incremental random samples of the specified sizes N. *Incremental* means that each sample is a superset of the preceding sample.

See Also

`tfl` for more information about type frequency lists

`sample.spc` is an analogous function for frequency spectra (objects of class `spc`)

Examples

```
## load Brown tfl
data(Brown.tfl)
summary(Brown.tfl)

## sample a tfl of 100k tokens
MiniBrown.tfl <- sample.tfl(Brown.tfl,1e+5)
summary(MiniBrown.tfl)

## if we repat, we get a different sample
MiniBrown.tfl <- sample.tfl(Brown.tfl,1e+5)
summary(MiniBrown.tfl)
```

`spc` *Frequency Spectra (zipfR)*

Description

In the `zipfR` library, `spc` objects are used to represent a word frequency spectrum (either an observed spectrum or the expected spectrum of a LNRE model at a given sample size).

With the `spc` constructor function, an object can be initialized directly from the specified data vectors. It is more common to read an observed spectrum from a disk file with `read.spc` or compute an expected spectrum with `lnre.spc`, though.

`spc` objects should always be treated as read-only.

Usage

```
spc(Vm, m=1:length(Vm), VVm=NULL, N=NA, V=NA, VV=NA,
    m.max=0, expected=!missing(VVm))
```

Arguments

`m`	integer vector of frequency classes m (if omitted, `Vm` is assumed to list the first k frequency classes $V_1, \ldots, V_k$)
`Vm`	vector of corresponding class sizes V_m (may be fractional for expected frequency spectrum $E[V_m]$)
`VVm`	optional vector of estimated variances $Var[V_m]$ (for expected frequency spectrum only)
`N, V`	total sample size N and vocabulary size V of frequency spectrum. While these values are usually determined automatically from `m` and `Vm`, they are required for an incomplete frequency spectrum that does not list all non-empty frequency classes.
`VV`	variance $Var[V]$ of expected vocabulary size. If `VVm` is specified, `VV` should also be given.

`m.max`	highest frequency class m listed in incomplete spectrum. If `m.max` is set, `N` and `V` also have to be specified, and all non-zero frequency classes up to `m.max` have to be included in the input vectors. Frequency classes above `m.max` in the input will automatically be deleted.
`expected`	set to `TRUE` if the frequency spectrum represents expected values $E[V_m]$ of the class sizes according to some LNRE model (this is automatically triggered when the `VVm` argument is specified).

Details

A `spc` object is a data frame with the following variables:

`m` frequency class m, an integer vector

`Vm` class size, i.e. number V_m of types in frequency class m (either observed class size from a sample or expected class size $E[V_m]$ based on a LNRE model)

`VVm` optional: estimated variance $V[V_m]$ of expected class size (only meaningful for expected spectrum derived from LNRE model)

The following attributes are used to store additional information about the frequency spectrum:

`m.max` if non-zero, the frequency spectrum is incomplete and lists only frequency classes up to `m.max`

`N, V` sample size N and vocabulary size V of the frequency spectrum. For a complete frequency spectrum, these values could easily be determined from `m` and `Vm`, but they are essential for an incomplete spectrum.

`VV` variance of expected vocabulary size; only present if `hasVariances` is `TRUE`. Note that `VV` may have the value `NA` is the user failed to specify it.

`expected` if `TRUE`, frequency spectrum lists expected class sizes $E[V_m]$ (rather than observed sizes V_m). Note that the `VVm` variable is only allowed for an expected frequency spectrum.

`hasVariances` indicates whether or not the `VVm` variable is present

Value

An object of class `spc` representing the specified frequency spectrum. This object should be treated as read-only (although such behaviour cannot be enforced in R).

See Also

`read.spc`, `write.spc`, `spc.vector`, `sample.spc`, `spc2tfl`, `tfl2spc`, `lnre.spc`, `plot.spc`

Generic methods supported by `spc` objects are `print`, `summary`, `N`, `V`, `Vm`, `VV`, and `VVm`.

Implementation details and non-standard arguments for these methods can be found on the manpages `print.spc`, `summary.spc`, `N.spc`, `V.spc`, etc.

Examples

```
## load Brown imaginative prose spectrum and inspect it
data(BrownImag.spc)

summary(BrownImag.spc)
print(BrownImag.spc)
```

```
plot(BrownImag.spc)

N(BrownImag.spc)
V(BrownImag.spc)
Vm(BrownImag.spc,1)
Vm(BrownImag.spc,1:5)

## compute ZM model, and generate PARTIAL expected spectrum
## with variances for a sample of 10 million tokens
zm <- lnre("zm",BrownImag.spc)
zm.spc <- lnre.spc(zm,1e+7,variances=TRUE)

## inspect extrapolated spectrum
summary(zm.spc)
print(zm.spc)

plot(zm.spc,log="x")

N(zm.spc)
V(zm.spc)
VV(zm.spc)
Vm(zm.spc,1)
VVm(zm.spc,1)

## generate an artificial Zipfian-looking spectrum
## and take a look at it
zipf.spc <- spc(round(1000/(1:1000)^2))

summary(zipf.spc)
plot(zipf.spc)

## see manpages of lnre, and the various *.spc mapages
## for more examples of spc usage
```

`spc2tfl` *Convert Between Frequency Spectra and Type Frequency Lists (zipfR)*

Description

`tfl2spc` computes an observed frequency spectrum from a type frequency list, while `spc2tfl` reconstructs the type frequency list underlying a frequency spectrum (but without type representations).

Usage

```
tfl2spc(tfl)

spc2tfl(spc)
```

Arguments

`tfl`	an object of class `tfl`, representing a type frequency list
`spc`	an object of class `spc`, representing a frequency spectrum

Details

The current implementation of these functions does not support incomplete type frequency lists and frequency spectra.

`spc2tfl` can only convert frequency spectra where all class sizes are integers. For this reason, expected frequency spectra (including all spectra with variance data) are not supported.

Value

For `tfl2spc`, an object of class `spc` representing the frequency spectrum corresponding to the type frequency list `tfl`.

For `spc2tfl`, an object of class `tfl` representing type frequency list underlying the observed frequency spectrum `tfl`.

See Also

`spc` for more information about `spc` objects and links to relevant functions; `tfl` for more information about `tfl` objects and links to relevant functions

Examples

```
## Brown tfl and spc
data(Brown.tfl)
data(Brown.spc)

## a spectrum from a tfl
Brown.spc2 <- tfl2spc(Brown.tfl)

## identical to Brown.spc:
summary(Brown.spc)
summary(Brown.spc2)

tail(Brown.spc)
tail(Brown.spc2)

## a tfl from a spectrum
Brown.tfl2 <- spc2tfl(Brown.spc)

## same frequency information as Brown.tfl
## but with different ids and no type labels
summary(Brown.tfl)
summary(Brown.tfl2)

print(Brown.tfl2)
print(Brown.tfl)
```

`spc.interp` *Expected Frequency Spectrum by Binomial Interpolation (zipfR)*

Description

`spc.interp` computes the expected frequency spectrum for a random sample of specified size N, taken from a data set described by the frequency spectrum object `obj`.

Usage

```
spc.interp(obj, N, m.max=max(obj$m), allow.extrapolation=FALSE)
```

Arguments

`obj`	an object of class `spc`, representing the frequency spectrum of the data set from which samples are taken
`N`	a single non-negative integer specifying the sample size for which the expected frequency spectrum is calculated
`m.max`	number of spectrum elements listed in the expected frequency spectrum. By default, as many spectrum elements are included as the spectrum `obj` contains, since the expectations of higher spectrum elements will always be 0 in the binomial interpolation. See note in section "Details" below.
`allow.extrapolation`	if `TRUE`, the requested sample size N may be larger than the sample size of the frequency spectrum `obj`, for binomial *extrapolation*. This obtion should be used with great caution (see `EVm.spc` for details).

Details

See the `EVm.spc` manpage for more information, especially concerning binomial *extrapolation*.

For large frequency spectra, the default value of `m.max` may lead to very long computation times. It is therefore recommended to specify `m.max` explicitly and calculate only as many spectrum elements as are actually required.

Value

An object of class `spc`, representing the expected frequency spectrum for a random sample of size `N` taken from the data set that is described by `obj`.

See Also

`spc` for more information about frequency spectra and links to relevant functions

The implementation of `spc.interp` is based on the functions `EV.spc` and `EVm.spc`. See the respective manpages for technical details.

`vgc.interp` computes expected vocabulary growth curves by binomial interpolation from a frequency spectrum

`sample.spc` takes a single *concrete* random subsample from a spectrum and returns the spectrum of the subsample, unlike `spc.interp`, that computes the *expected* frequency spectrum for random subsamples of size `N` by binomial interpolation.

Examples

```
## load the Tiger NP expansion spectrum
## (sample size: about 109k tokens)
data(TigerNP.spc)

## interpolated expected frequency subspectrum of 50k tokens
TigerNP.sub.spc <- spc.interp(TigerNP.spc,5e+4)
summary(TigerNP.sub.spc)

## previous is slow since it calculates all expected  spectrum
## elements; suppose we only need the first 10 expected
## spectrum element frequencies; then we can do:
TigerNP.sub.spc <- spc.interp(TigerNP.spc,5e+4,m.max=10) # much faster!
summary(TigerNP.sub.spc)
```

`spc.vector` *Create Vector of Spectrum Elements (zipfR)*

Description

`spc.vector` returns a selected range of elements from a frequency spectrum as a plain numeric vector (which may contain entries with $V_m = 0$, unlike the `spc` object itself).

Usage

```
spc.vector(obj, m.min=1, m.max=15, all=FALSE)
```

Arguments

`obj`	an object of class `spc`, representing an observed or expected frequency spectrum
`m.min, m.max`	determine the range of frequency classes to be returned (defaulting to $1 \ldots 15$)
`all`	if `TRUE`, a vector containing the entire frequency spectrum is returned, i.e. `m.max` is set to `max(obj$m)`. Note that the value of `m.min` can still be overridden manually to return only part of the spectrum.

Details

`spc.vector(obj, a, b)` is fully equivalent to `Vm(obj, a:b)` (and is implemented in this way).

Value

A numeric vector with the selected elements of the frequency spectrum. In this vector, empty frequency classes ($V_m = 0$) are represented by 0 entries (unlike the `spc` object, which omits all empty classes).

See Also

`spc` for more information about `spc` objects and links to relevant functions

`Vm.spc` for an alternative way of extracting spectrum vectors from a `.spc` object, and `N.spc`, `V.spc`, `VV.spc`, `VVm.spc` for extracting related information

Examples

```
## Brown Noun spectrum
data(BrownNoun.spc)

## by default, extract first 15 elements
spc.vector(BrownNoun.spc)

## first five elements
spc.vector(BrownNoun.spc,1,5)

## just frequencies of spc elements 4 and 5
spc.vector(BrownNoun.spc,4,5)
## same as
Vm(BrownNoun.spc,4:5)
```

`tfl` *Type Frequency Lists (zipfR)*

Description

In the `zipfR` library, `tfl` objects are used to represent a type frequency list, which specifies the observed frequency of each type in a corpus. For mathematical reasons, expected type frequencies are rarely considered.

With the `tfl` constructor function, an object can be initialized directly from the specified data vectors. It is more common to read a type frequency list from a disk file with `read.tfl` or, in some cases, derive it from an observed frequency spectrum with `spc2tfl`.

`tfl` objects should always be treated as read-only.

Usage

```
tfl(f, k=1:length(f), type=NULL, f.min=min(f), f.max=max(f),
    incomplete=!(missing(f.min) && missing(f.max)), N=NA, V=NA,
    delete.zeros=FALSE)
```

Arguments

`k`	integer vector of type IDs k (if omitted, natural numbers $1, 2, \ldots$ are assigned automatically)
`f`	vector of corresponding type frequencies f_k
`type`	optional character vector of type representations (e.g. word forms or lemmata), used for informational and printing purposes only

`incomplete`	indicates that the type frequency list is incomplete, i.e. only contains types in a certain frequency range (typically, the lowest-frequency types may be excluded). Incomplete type frequency lists are rarely useful.
`N, V`	sample size and vocabulary size corresponding to the type frequency list have to be specified explicitly for incomplete lists
`f.min, f.max`	frequency range represented in an incomplete type frequency list (see details below)
`delete.zeros`	if `TRUE`, delete types with $f = 0$ from the type frequency list, *after* assigning type IDs. This operation does *not* make the resulting `tfl` object incomplete.

Details

If `f.min` and `f.max` are not specified, but the list is marked as incomplete (with `incomplete=TRUE`), they are automatically determined from the frequency vector `f` (making the assumption that all types in this frequency range are listed). Explicit specification of either `f.min` or `f.max` implies an incomplete list. In this case, all types outside the specified range will be deleted from the list. If `incomplete=FALSE` is explicitly given, `N` and `V` will be determined automatically from the input data (which is assumed to be complete), but the resulting type frequency list will still be incomplete.

If you just want to remove types with $f = 0$ without marking the type frequency list as incomplete, use the option `delete.zeros=TRUE`.

A `tfl` object is a data frame with the following variables:

`k` integer type ID k

`f` corresponding type frequency f_k

`type` optional: character vector with type representations used for printing

The data frame always has to be sorted with respect to the `k` column (ascending order).

The following attributes are used to store additional information about the frequency spectrum:

`N, V` sample size N and vocabulary size V corresponding to the type frequency list. For a complete list, these values could easily be determined from the `f` variable, but they are essential for an incomplete list.

`incomplete` if `TRUE`, the type frequency list is incomplete, i.e. it lists only types in the frequency range given by `f.min` and `f.max`

`f.min, f.max` range of type frequencies represented in the list (should be ignored unless the `incomplete` flag is set)

`hasTypes` indicates whether or not the `type` variable is present

Value

An object of class `tfl` representing the specified type frequency list. This object should be treated as read-only (although such behaviour cannot be enforced in R).

See Also

`read.tfl`, `write.tfl`, `sample.tfl`, `spc2tfl`, `tfl2spc`

Generic methods supported by `tfl` objects are `print`, `summary`, `N`, `V` and `Vm`.

Implementation details and non-standard arguments for these methods can be found on the manpages `print.tfl`, `summary.tfl`, `N.tfl`, `V.tfl`, etc.

Examples

```
## typically, you will read a tfl from a file
## (see examples in the read.tfl manpage)

## or you can load a ready-made tfl
data(Brown.tfl)
summary(Brown.tfl)
print(Brown.tfl)

## or create it from a spectrum (with different ids and
## no type labels)
data(Brown.spc)

Brown.tfl2 <- spc2tfl(Brown.spc)

## same frequency information as Brown.tfl
## but with different ids and no type labels
summary(Brown.tfl2)
print(Brown.tfl2)

## how to display draw a Zipf's rank/frequency plot
## by extracting frequencies from a tfl
plot(sort(Brown.tfl$f,decreasing=TRUE),log="y",xlab="rank",ylab="frequency")

## simulating a tfl
Zipfian.tfl <- tfl(1000/(1:1000))
plot(Zipfian.tfl$f,log="y")
```

`vec2xxx` *Type-Token Statistics for Samples and Empirical Data (zipfR)*

Description

Compute type-frequency list, frequency spectrum and vocabulary growth curve from a token vector representing a random sample or an observed sequence of tokens.

Usage

```
vec2tfl(x)

vec2spc(x)

vec2vgc(x, steps=200, stepsize=NA, m.max=0)
```

Arguments

`x`	a vector of length N_0, representing a random sample or other observed data set of N_0 tokens. For each token, the corresponding element of `x` specifies the *type* that the token belongs to. Usually, `x` is a character vector, but it might also specify integer IDs in some cases.

`steps`	number of steps for which vocabulary growth data $V(N)$ is calculated. The values of N will be evenly spaced (up to rounding differences) from $N = 1$ to $N = N_0$.
`stepsize`	alternative way of specifying the steps of the vocabulary growth curve. In this case, vocabulary growth data will be calculated every `stepsize` tokens. The first step is chosen such that the last step corresponds to the full sample ($N = N_0$). Only one of the parameters `steps` and `stepsize` may be specified.
`m.max`	an integer in the range $1 \dots 9$, specifying how many spectrum elements $V_m(N)$ to include in the vocabulary growth curve. By default only vocabulary size $V(N)$ is calculated, i.e. `m.max=0`.

Details

There are two main applications for the `vec2xxx` functions:

a) They can be used to calculate type-token statistics and vocabulary growth curves for random samples generated from a LNRE model (with the `rlnre` function).

b) They provide an easy way to process a user's own data without having to rely on external scripts to compute frequency spectra and vocabulary growth curves. All that is needed is a text file in one-token-per-line formt (i.e. where each token is given on a separate line). See "Examples" below for further hints.

Both applications work well for samples of up to approx. 1 million tokens. For considerably larger data sets, specialized external software should be used, such as the Perl scripts provided on the `zipfR` homepage.

Value

An object of class `tfl`, `spc` or `vgc`, representing the type frequency list, frequency spectrum or vocabulary growth curve of the token vector `x`, respectively.

See Also

`tfl`, `spc` and `vgc` for more information about type frequency lists, frequency spectra and vocabulary growth curves

`rlnre` for generating random samples (in the form of the required token vectors) from a LNRE model

`readLines` and `scan` for loading token vectors from disk files

Examples

```
## type-token statistics for random samples from a LNRE distribution

model <- lnre("fzm", alpha=.5, A=1e-6, B=.05)
x <- rlnre(model, 100000)

vec2tfl(x)
vec2spc(x)  # same as tfl2spc(vec2tfl(x))
vec2vgc(x)

sample.spc <- vec2spc(x)
exp.spc <- lnre.spc(model, 100000)
```

```
## Not run:  plot(exp.spc, sample.spc)

sample.vgc <- vec2vgc(x, m.max=1, steps=500)
exp.vgc <- lnre.vgc(model, N=N(sample.vgc), m.max=1)
## Not run:  plot(exp.vgc, sample.vgc, add.m=1)

## load token vector from a file in one-token-per-line format

## Not run:  x <- readLines(filename)
## Not run:  x <- readLines(file.choose()) # with file selection dialog

## you can also perform whitespace tokenization and filter the data

## Not run:  brown <- scan("brown.pos", what=character(0), quote="")
## Not run:  nouns <- grep("/NNS?$", brown, value=TRUE)
## Not run:  plot(vec2spc(nouns))
## Not run:  plot(vec2vgc(nouns, m.max=1), add.m=1)
```

---

vgc *Vocabulary Growth Curves (zipfR)*

Description

In the `zipfR` library, `vgc` objects are used to represent a vocabulary growth curve (VGC). This can be an observed VGC from an incremental set of sample (such as a corpus), a randomized VGC obtained by binomial interpolation, or the expected VGC according to a LNRE model.

With the `vgc` constructor function, an object can be initialized directly from the specified data vectors. It is more common to read an observed VGC from a disk file with `read.vgc`, generate a randomized VGC with `vgc.interp` or compute an expected VGC with `lnre.vgc`, though.

`vgc` objects should always be treated as read-only.

Usage

```
vgc(N, V, Vm=NULL, VV=NULL, VVm=NULL, expected=FALSE, check=TRUE)
```

Arguments

`N`	integer vector of sample sizes N for which vocabulary growth data is available
`V`	vector of corresponding vocabulary sizes $V(N)$, or expected vocabulary sizes $E[V(N)]$ for an interpolated or expected VGC.
`Vm`	optional list of growth vectors for hapaxes $V_1(N)$, dis legomena $V_2(N)$, etc. Up to 9 growth vectors are accepted (i.e. $V_m(N)$ for $m \leq 9$). For an interpolated or expected VGC, the vectors represent expected class sizes $E[V_m(N)]$.
`VV`	optional vector of variances $Var[V(N)]$ for an interpolated or expected VGC
`VVm`	optional list of variance vectors $Var[V_m(N)]$ for an expected VGC. If present, these vectors must be defined for exactly the same frequency classes m as the vectors in `Vm`.

`expected`	if `TRUE`, the object represents an interpolated or expected VGC (for informational purposes only)
`check`	by default, various sanity checks are performed on the data supplied to the `spc` constructor. Specify `check=FALSE` to skip these sanity test, e.g. when automatically processing data from external programs that may be numerically unstable.

Details

If variances (`VV` or `VVm`) are specified for an expected VGC, all relevant vectors must be given. In other words, `VV` always has to be present in this case, and `VVm` has to be present whenever `Vm` is specified, and must contain vectors for exactly the same frequency classes.

`V` and `VVm` are integer vectors for an observed VGC, but will usually be fractional for an interpolated or expected VGC.

A `vgc` object is a data frame with the following variables:

`N` sample size N

`V` corresponding vocabulary size (either observed vocabulary size $V(N)$ or expected vocabulary size $E[V(N)]$)

`V1 ...V9` optional: observed or expected spectrum elements ($V_m(N)$ or $E[V_m(N)]$). Not all of these variables have to be present, but there must not be any "gaps" in the spectrum.

`VV` optional: variance of expected vocabulary size, $Var[V(N)]$

`VV1 ...VV9` optional: variances of expected spectrum elements, $Var[V_m(N)]$. If variances are present, they must be available for exactly the same frequency classes as the corresponding expected values.

The following attributes are used to store additional information about the vocabulary growth curve:

`m.max` if non-zero, the VGC includes spectrum elements $V_m(N)$ for m up to `m.max`. For `m.max=0`, no spectrum elements are present.

`expected` if `TRUE`, the object represents an interpolated or expected VGC, with expected vocabulary size and spectrum elements. Otherwise, the object represents an observed VGC.

`hasVariances` indicates whether or not the `VV` variable is present (as well as `VV1`, `VV2`, etc., if appropriate)

Value

An object of class `vgc` representing the specified vocabulary growth curve. This object should be treated as read-only (although such behaviour cannot be enforced in R).

See Also

`read.vgc`, `write.vgc`, `plot.vgc`, `vgc.interp`, `lnre.vgc`

Generic methods supported by `vgc` objects are `print`, `summary`, `N`, `V`, `Vm`, `VV`, and `VVm`.

Implementation details and non-standard arguments for these methods can be found on the manpages `print.vgc`, `summary.vgc`, `N.vgc`, `V.vgc`, etc.

Examples

```
## load Dickens' work empirical vgc and take a look at it

data(Dickens.emp.vgc)
summary(Dickens.emp.vgc)
print(Dickens.emp.vgc)

plot(Dickens.emp.vgc,add.m=1)

## vectors of sample sizes in the vgc, and the
## corresponding V and V_1 vectors
Ns <- N(Dickens.emp.vgc)
Vs <- V(Dickens.emp.vgc)
Vm <- V(Dickens.emp.vgc,1)

## binomially interpolated V and V_1 at the same sample sizes
## as the empirical curve
data(Dickens.spc)
Dickens.bin.vgc <- vgc.interp(Dickens.spc,N(Dickens.emp.vgc),m.max=1)

## compare observed and interpolated
plot(Dickens.emp.vgc,Dickens.bin.vgc,add.m=1,legend=c("observed","interpolated"))

## load Italian ultra- prefix data
data(ItaUltra.spc)

## compute zm model
zm <- lnre("zm",ItaUltra.spc)

## compute vgc up to about twice the sample size
## with variance of V
zm.vgc <- lnre.vgc(zm,(1:100)*70, variances=TRUE)

summary(zm.vgc)
print(zm.vgc)

## plot with confidence intervals derived from variance in
## vgc (with larger datasets, ci will typically be almost
## invisible)
plot(zm.vgc)

## for more examples of vgc usages, see manpages of lnre.vgc,
## plot.vgc, print.vgc  and vgc.interp
```

`vgc.interp` *Expected Vocabulary Growth by Binomial Interpolation (zipfR)*

Description

`vgc.interp` computes the expected vocabulary growth curve for random sample taken from a data set described by the frequency spectrum object `obj`.

Usage

```
vgc.interp(obj, N, m.max=0, allow.extrapolation=FALSE)
```

Arguments

`obj`	an object of class `spc`, representing the frequency spectrum of the data set from which samples are taken
`N`	a vector of increasing non-negative integers specifying the sample sizes for the expected vocabulary size is calculated (as well as expected spectrum elements if requested)
`m.max`	an integer in the range $1 \dots 9$, specifying the number of spectrum elements to be included in the vocabulary growth curve (default: none)
`allow.extrapolation`	if `TRUE`, the requested sample sizes N may be larger than the sample size of the frequency spectrum `obj`, so that binomial *extrapolation* is performed. This obtion should be used with great caution (see `EV.spc` for details).

Details

See the `EV.spc` manpage for more information, especially concerning binomial *extrapolation*.

Note that the *result* of `vgc.interp` is an object of class `vgc` (a vocabulary growth curve), but its *input* is an object of class `spc` (a frequency spectrum).

Value

An object of class `vgc`, representing the expected vocabulary growth curves for random samples taken from the data set described by `obj`. Data points will be generated for the specified sample sizes `N`.

See Also

`vgc` for more information about vocabulary growth curves and links to relevant functions; `spc` for more information about frequency spectra

The implementation of `vgc.interp` is based on the functions `EV.spc` and `EVm.spc`. See the respective manpages for technical details.

`spc.interp` computes the expected frequency spectrum for a random sample by binomial interpolation.

Examples

```
## load the Tiger PP expansion spectrum
## (sample size: about 91k tokens)
data(TigerPP.spc)

## binomially interpolated curve
TigerPP.bin.vgc <- vgc.interp(TigerPP.spc,(1:100)*910)
summary(TigerPP.bin.vgc)

## let's also add growth of V_1 to V_5 and plot
```

```
TigerPP.bin.vgc <- vgc.interp(TigerPP.spc,(1:100)*910,m.max=5)
plot(TigerPP.bin.vgc,add.m=c(1:5))
```

`zipfR.legend` *Draw a Legend Box in one of the Corners (zipfR)*

Description

`zipfR.legend` is a thin wrapper around the standard `legend` function, helping to place the legend box in one of the corners of the screen without knowing the precise coordinate ranges in use.

Fine control over the placement of the legend box is possible with the optional arguments `margin.x` and `margin.y`. `zipfR.legend` is used internally by the high-level plotting functions `plot.spc` and `plot.vgc`.

Usage

```
zipfR.legend(corner, margin.x=.05, margin.y=margin.x,
             legend=NULL, bg="white", ...)
```

Arguments

`corner`	an integer specifying the corner in which to place the legend box (1 = top left, 2 = top right, 3 = bottom right, and 4 = bottom left)
`margin.x`	distance of the legend box from the border of the plotting region along the x-axis, as a fraction of the width of the plotting region (default: 5%)
`margin.y`	distance of the legend box from the border of the plotting region along the y-axis, as a fraction of the height of the plotting region. Defaults to the same value as `margin.x`, so in most cases it is sufficient to specify `margin.x` in order to move the legend box towards a corner or away from the corner.
`legend`	a character or expression vector specifying the legend text (passed on to `legend`
`bg`	background colour of the legend box. Defaults to `"white"`, so parts of the plot behind the box are hidden.
`...`	further arguments are passed on to the `legend` function and can be used to specify line styles, plot symbols and/or fill styles for the legend entries

See Also

`legend` for details on how to specify legend entries. Most of the options documented there can also be used with `zipfR.legend`.

`plot.spc`, `plot.vgc` and `zipfR.plotutils` for more information about graphics functions in `zipfR`

Examples

```
## zipfR.legend() can be used with all standard plots as well

## Not run: plot(sin, 0, 2*pi)
## Not run: zipfR.legend(2, margin.x=.1, legend="Some silly legend")
```

`zipfR-package` *zipfR: lexical statistics in R*

Description

The zipfR package performs Large-Number-of-Rare-Events (LNRE) modeling of (linguistic) type frequency distributions (Baayen 2001) and provides utilities to run various forms of lexical statistics analysis in R.

Details

The best way to get started with zipfR is to read the tutorial, which you can find via the HTML documentation (follow the Overview link); you can also download it from `http://purl.org/stefan.evert/zipfR/`

zipfR is released under the GNU General Public License (`http://www.gnu.org/copyleft/gpl.html`)

Author(s)

Stefan Evert <⟨stefan.evert@uos.de⟩> and Marco Baroni <⟨marco.baroni@unitn.it⟩>

Maintainer: Stefan Evert <⟨stefan.evert@uos.de⟩>

References

zipfR Website: `http://purl.org/stefan.evert/zipfR/`

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

Baroni, Marco (to appear). Distributions in text. To appear in: A. Lüdeling and M. Kytö (eds.), *Corpus Linguistics. An International Handbook*, chapter 39. Mouton de Gruyter, Berlin.

Evert, Stefan (2004). *The Statistics of Word Cooccurrences: Word Pairs and Collocations.* PhD Thesis, IMS, University of Stuttgart. URN urn:nbn:de:bsz:93-opus-23714 `http://elib.uni-stuttgart.de/opus/volltexte/2005/2371/`

Evert, Stefan (2004). A simple LNRE model for random character sequences. *Proceedings of JADT 2004*, 411-422.

Evert, Stefan and Baroni, Marco (2006). Testing the extrapolation quality of word frequency models. *Proceedings of Corpus Linguistics 2005.*

Evert, Stefan and Baroni, Marco (2006). The zipfR library: Words and other rare events in R. *useR! 2006: The second R user conference.*

See Also

The zipfR tutorial: available from http://purl.org/stefan.evert/zipfR/ and via the HTML documentation (by following the Overview link)

Some good entry points into the zipfR documentation are be `spc`, `vgc`, `tfl`, `read.spc`, `read.tfl`, `read.vgc`, `lnre`, `lnre.vgc`, `plot.spc`, `plot.vgc`

The same authors also develop the `corpora` library (available on CRAN) supporting simple inferential statistics for corpus analysis

Harald Baayen's LEXSTATS tools: http://www.mpi.nl/world/persons/private/baayen/software.html

Stefan Evert's UCS tools: http://collocations.de/

Examples

```
## load Oliver Twist and Great Expectations frequency spectra
data(DickensOliverTwist.spc)
data(DickensGreatExpectations.spc)

## check sample size and vocabulary and hapax counts
N(DickensOliverTwist.spc)
V(DickensOliverTwist.spc)
Vm(DickensOliverTwist.spc,1)
N(DickensGreatExpectations.spc)
V(DickensGreatExpectations.spc)
Vm(DickensGreatExpectations.spc,1)

## compute binomially interpolated growth curves
ot.vgc <- vgc.interp(DickensOliverTwist.spc,(1:100)*1570)
ge.vgc <- vgc.interp(DickensGreatExpectations.spc,(1:100)*1865)

## plot them
plot(ot.vgc,ge.vgc,legend=c("Oliver Twist","Great Expectations"))

## load Dickens' works frequency spectrum
data(Dickens.spc)

## compute Zipf-Mandelbrot model from Dickens data
## and look at model summary
zm <- lnre("zm",Dickens.spc)
zm

## plot observed and expected spectrum
zm.spc <- lnre.spc(zm,N(Dickens.spc))
plot(Dickens.spc,zm.spc)

## obtain expected V and V1 values at arbitrary sample sizes
EV(zm,1e+8)
EVm(zm,1,1e+8)

## generate expected V and V1 growth curves up to a sample size
## of 10 million tokens and plot them, with vertical line at
## estimation size
ext.vgc <- lnre.vgc(zm,(1:100)*1e+5,m.max=1)
plot(ext.vgc,N0=N(zm),add.m=1)
```

`zipfR.par` *Set or Query Graphics Parameters (zipfR)*

Description

Set default graphics parameters for `zipfR` high-level plots and plot utilities, 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.

NB: This is an advanced function to fine-tune zipfR plots. For basic plotting options (that are likely to be sufficient for most purposes) see `plot.spc` and `plot.vgc` instead.

Usage

```
zipfR.par(..., bw.mode=FALSE)
```

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. A listing of valid parameter names is given below.
`bw.mode`	if `TRUE` and parameter values are queried, then return the corresponding parameters for B/W mode if possible (e.g., `zipfR.par("col",bw.mode=TRUE)` returns the value of the `col.bw` parameter). Note that `bw.mode` cannot be abbreviated in the function call!

Details

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 `zipfR.par`, which is typically used to restore previous parameter values (that have been saved in a list variable).

Most of the default values can be manually overridden in the high-level plots.

`zipfR.par()` shows all parameters with their current values, and `names(zipfR.par())` produces a listing of valid parameter names.

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 `zipfR.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).

zipfR Graphics Parameters

`col` a character or integer vector specifying up to 10 line colours (see the `par` manpage for details). Values of shorter vectors are recycled as necessary.

`lty` a character or integer vector specifying up to 10 line styles (see the `par` manpage for details). Values of shorter vectors are recycled as necessary.

`lwd` a numeric vector specifying up to 10 line widths (see the `par` manpage for details). Values of shorter vectors are recycled as necessary.

`pch` a character or integer vector specifying up to 10 plot symbols. Values of shorter vectors are recycled as necessary.

`barcol` a character or integer vector specifying up to 10 colours for the bars in non-logarithmic spectrum plots. Values of shorter vectors are recycled as necessary.

`col.bw` the line colours used in B/W mode (`bw=TRUE`)

`lty.bw` the line styles used in B/W mode (`bw=TRUE`)

`lwd.bw` the line widths used in B/W mode (`bw=TRUE`)

`pch.bw` the plot symbols used in B/W mode (`bw=TRUE`)

`barcol.bw` the bar colours used in B/W mode (`bw=TRUE`)

`bw` if `TRUE`, plots are drawn in B/W mode unless specified otherwise (default: `FALSE`, i.e. colour mode

`device` plot device used by the `zipfR` plotutils (see `zipfR.begin.plot` for details). Currently supported devices are `x11` (the default), `eps`, `pdf`, as well as `png` and `quartz` where available.

`init.par` list of named graphics parameters passed to the `par` function whenever a new viewport is created with `zipfR.begin.plot`

`width, height` default width and height of the plotting window opened by `zipfR.begin.plot`

See Also

`plot.spc`, `plot.vgc`, `zipfR.begin.plot`, `zipfR.end.plot`

Examples

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

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

## temporary changes to graphics paramters:
par.save <- zipfR.par(bw=TRUE, lwd.bw=2)
## plots use the modified parameters here
zipfR.par(par.save)                # restore previous values
```

zipfR.plotutils *Plotting Utilities (zipfR)*

Description

Conveniently create plots with different layout and in different output formats (both on-screen and various graphics file formats).

Each plot is wrapped in a pair of `zipfR.begin.plot` and `zipfR.end.plot` commands, which make sure that a suitable plotting window / image file is opened and closed as required. Format and dimensions of the plots are controlled by global settings made with `zipfR.par`, but can be overridden in the `zipfR.begin.plot` call.

`zipfR.pick.device` automatically selects a default device by scanning the specified vector for strings of the form `--pdf`, `--eps`, etc.

NB: These are advanced functions to fine-tune plotting output. For basic plotting functionalities (that should be sufficient in most cases) see `plot.spc` and `plot.vgc` instead.

Usage

```
zipfR.pick.device(args=commandArgs())

zipfR.begin.plot(device=zipfR.par("device"), filename="",
                 width=zipfR.par("width"), height=zipfR.par("height"),
                 bg=zipfR.par("bg"), pointsize=zipfR.par("pointsize"))

## plotting commands go here

zipfR.end.plot()
```

Arguments

`args`	a character vector, which will be scanned for strings of the form `--pdf`, `--eps`, etc. If `args` is not specified, the command-line arguments supplied to R will be examined.
`device`	name of plotting device to be used (see "Devices" below)
`filename`	for graphics file devices, *basename* of the output file. A suitable extension for the selected file format will be added automatically to `filename`. This parameter is ignored for screen devices.
`width, height`	width and height of the plotting window or image, in inches
`bg`	background colour of the plotting window or image (use `"transparent"` for images with transparent background)
`pointsize`	default point size for text in the plot

Details

`zipfR.begin.plot` opens a new plotting window or image file of the specified dimensions (`width`, `height`), using the selected graphics device (`device`). Background colour (`bg`) and default point size (`pointsize`) are set as requested. Then, any global graphics parameter settings (defined with the `init.par` option of `zipfR.par`) are applied. See the `zipfR.par` manpage for the "factory default" settings of these options.

`zipfR.end.plot` finalizes the current plot. For image file devices, the device will be closed, writing the generated file to disk. For screen devices, the plotting window remains visible until a new plot is started (which will close and re-open the plotting window).

The main purpose of the `zipfR` plotting utilities is to make it easier to draw plots that are both shown on screen (for interactive work) and saved to image files in various formats. If an R script specifies `filenames` in all `zipfR.begin.plot` commands, a single global parameter setting at the start of the script is sufficient to switch from screen graphics to EPS files, or any other supported file format.

The factory-default graphics device is `x11`, which is available on all major platforms (sometimes as an alias for a native device). On Mac OS X, the Aqua GUI version of R defaults to the `quartz` device, which produces higher-quality images.

The `png` bitmap device may not be available on all platforms, and may also require access to an X server. Since the width and height of a PNG device have to be specified in pixels rather than inches, `zipfR.begin.plot` translates the `width` and `height` settings, assuming a resolution of 150 dpi. Use of the `png` device is strongly discouraged. A better way of producing high-quality bitmaps is to generate EPS image (with the `eps` device) and convert them to PNG or JPEG format with the external `pstoimg` program (part of the `latex2html` distribution).

`zipfR.pick.device` will issue a warning if multiple flags matching supported graphics devices are found. However, it is not an error to find no matching flag, and all unrecognized strings are silently ignored.

Devices

Currently, the following devices are supported (and can be used in the `device` argument).

Screen devices:

`x11` X11 graphics device, available on all major platforms (may be mapped to native device, e.g. in Windows)

`quartz` high-quality graphics device with anti-aliasing, available on Mac OS X only (Aqua GUI version)

Graphics file devices:

`eps` Encapsulated PostScript (EPS) output (using `postscript` device with appropriate settings)

`pdf` PDF output

`png` PNG bitmap file (may not be available on all platforms)

See Also

`zipfR.par`, `par`

`x11`, `quartz`, `postscript`, `pdf` and `png` for more information about the supported graphics devices

`zipfR`-specific plotting commands are `plot.spc` and `plot.vgc`

Examples

```
## these graphics parameters will be set for every new plot

## Not run: zipfR.par(init.par=list(bg="lightblue", cex=1.3))
## Not run: zipfR.par(width=12, height=9)

## will be shown on screen or saved to specified file, depending on
## selected device (eps -> "myplot.eps", pdf -> "myplot.pdf", etc.)

## Not run: zipfR.begin.plot(filename="myplot")
## Not run: plot.spc(Brown100k.spc)
## Not run: zipfR.end.plot()

## By starting an R script "myplots.R" with this command, you can
## display plots on screen when stepping through the script in an
## interactive session, or save them to disk files in various
## graphics formats with "R --no-save --args --pdf < myplots.R" etc.

## Not run: zipfR.pick.device()
```