Package 'bbmle'

May 11, 2022

Title Tools for General Maximum Likelihood Estimation

Description Methods and functions for fitting maximum likelihood models in R. This package modifies and extends the 'mle' classes in the 'stats4' package.

Version 1.0.25

Depends R (>= 3.0.0), stats4

Imports stats, numDeriv, lattice, MASS, methods, bdsmatrix, Matrix, mvtnorm

Suggests emdbook, rms, ggplot2, RUnit, MuMIn, AICcmodavg, Hmisc, optimx (>= 2013.8.6), knitr, testthat

VignetteBuilder knitr

BuildVignettes yes

License GPL

URL https://github.com/bbolker/bbmle

Collate 'mle2-class.R' 'mle2-methods.R' 'mle.R' 'confint.R' 'predict.R' 'profile.R' 'update.R' 'dists.R' 'IC.R' 'slice.R' 'impsamp.R' 'TMB.R'

RoxygenNote 7.1.0

Encoding UTF-8

NeedsCompilation no

Author Ben Bolker [aut, cre], R Development Core Team [aut], Iago Giné-Vázquez [ctb]

Maintainer Ben Bolker <bolker@mcmaster.ca>

Repository CRAN

Date/Publication 2022-05-11 08:20:02 UTC

R topics documented:

`as.data.frame.profile.mle2` *convert profile to data frame*

Description

converts a profile of a fitted mle2 object to a data frame

Usage

```
## S3 method for class 'profile.mle2'
as.data.frame(x, row.names=NULL,
optional=FALSE, ...)
```

Arguments

`x`	a profile object
`row.names`	row names (unused)
`optional`	unused
`...`	unused

Value

a data frame with columns

`param`	name of parameter being profiled
`z`	signed square root of the deviance difference from the minimum
`parameter values`	named par.vals.parname
`focal`	value of focal parameter: redundant, but included for plotting convenience

Author(s)

Ben Bolker

Examples

```
## use as.data.frame and lattice to plot profiles
x <- 0:10
y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
library(bbmle)
LL <- function(ymax=15, xhalf=6) {
      -sum(stats::dpois(y, lambda=ymax/(1+x/xhalf), log=TRUE))
  }
## uses default parameters of LL
fit1 <- mle2(LL)
p1 <- profile(fit1)
d1 <- as.data.frame(p1)
library(lattice)
xyplot(abs(z)~focal|param,data=d1,
  subset=abs(z)<3,
  type="b",
  xlab="",
  ylab=expression(paste(abs(z),
  " (square root of ",Delta," deviance)")),
  scale=list(x=list(relation="free")))
```

`BIC-methods` *Log likelihoods and model selection for mle2 objects*

Description

Various functions for likelihood-based and information-theoretic model selection of likelihood models

Usage

```
## S4 method for signature 'ANY,mle2,logLik'
AICc(object,...,nobs,k=2)
## S4 method for signature 'ANY,mle2,logLik'
qAIC(object,...,k=2)
## S4 method for signature 'ANY,mle2,logLik'
qAICc(object,...,nobs,k=2)
```

Arguments

object	A logLik or mle2 object
...	An optional list of additional logLik or mle2 objects (fitted to the same data set).
nobs	Number of observations (sometimes obtainable as an attribute of the fit or of the log-likelihood)
k	penalty parameter (nearly always left at its default value of 2)

Details

Further arguments to BIC can be specified in the ... list: delta (logical) specifies whether to include a column for delta-BIC in the output.

Value

A table of the BIC values, degrees of freedom, and possibly delta-BIC values relative to the minimum-BIC model

Methods

logLik signature(object = "mle2"): Extract maximized log-likelihood.

AIC signature(object = "mle2"): Calculate Akaike Information Criterion

AICc signature(object = "mle2"): Calculate small-sample corrected Akaike Information Criterion

anova signature(object="mle2"): Likelihood Ratio Test comparision of different models

Note

This is implemented in an ugly way and could probably be improved!

Examples

```
d <- data.frame(x=0:10,y=c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8))
(fit <- mle2(y~dpois(lambda=ymax/(1+x/xhalf)),
    start=list(ymax=25,xhalf=3),data=d))
(fit2 <- mle2(y~dpois(lambda=(x+1)*slope),
    start=list(slope=1),data=d))
BIC(fit)
```

```
BIC(fit,fit2)
```

`call.to.char` *Convert calls to character*

Description

Utility function (hack) to convert calls such as y~x to their character equivalent

Usage

```
call.to.char(x)
```

Arguments

x	a formula (call)

Details

It would be nice if `as.character(y~x)` gave "y~x", but it doesn't, so this hack achieves the same goal

Value

a character vector of length 1

Author(s)

Ben Bolker

Examples

```
as.character(y~x)
call.to.char(y~x)
```

dnorm_n *Normal distribution with profiled-out standard deviation*

Description

Returns the Normal probability densities for a distribution with the given mean values and the standard deviation equal to the root mean-squared deviation between x and mu

Usage

```
dnorm_n(x, mean, log = FALSE)
```

Arguments

x	numeric vector of data
mean	numeric vector or mean values
log	logical: return the log-density?

Details

This is a convenience function, designed for the case where you're trying to compute a MLE for the mean but don't want to bother estimating the MLE for the standard deviation at the same time

Value

Numeric vector of probability densities

Examples

```
set.seed(101)
x <- rnorm(5,mean=3,sd=2)
dnorm_n(x,mean=3,log=TRUE)
```

get.mnames *extract model names*

Description

given a list of models, extract the names (or "model n")

Usage

```
get.mnames(Call)
```

Arguments

Call	a function call (usually a list of models)

Value

a vector of model names

Author(s)

Ben Bolker

ICtab *Compute table of information criteria and auxiliary info*

Description

Computes information criteria for a series of models, optionally giving information about weights, differences between ICs, etc.

Usage

```
ICtab(..., type=c("AIC","BIC","AICc","qAIC","qAICc"),
    weights = FALSE, delta = TRUE, base = FALSE,
logLik=FALSE, sort = TRUE,
nobs=NULL, dispersion = 1, mnames, k = 2)
AICtab(...,mnames)
BICtab(...,mnames)
AICctab(...,mnames)
## S3 method for class 'ICtab'
print(x,...,min.weight)
```

Arguments

...	a list of (logLik or?) mle objects; in the case of AICtab etc., could also include other arguments to ICtab
type	specify information criterion to use
base	(logical) include base IC (and log-likelihood) values?
weights	(logical) compute IC weights?
logLik	(logical) include log-likelihoods in the table?
delta	(logical) compute differences among ICs (and log-likelihoods)?
sort	(logical) sort ICs in increasing order?
nobs	(integer) number of observations: required for type="BIC" or type="AICc" unless objects have a nobs method
dispersion	overdispersion estimate, for computing qAIC: required for type="qAIC" or type="qAICc" unless objects have a "dispersion" attribute

mnames	names for table rows: defaults to names of objects passed
k	penalty term (largely unused: left at default of 2)
x	an ICtab object
min.weight	minimum weight for exact reporting (smaller values will be reported as "<[min.weight]")

Value

A data frame containing:

IC	information criterion
df	degrees of freedom/number of parameters
dIC	difference in IC from minimum-IC model
weights	exp(-dIC/2)/sum(exp(-dIC/2))

Note

(1) The print method uses sensible defaults; all ICs are rounded to the nearest 0.1, and IC weights are printed using `format.pval` to print an inequality for values <0.001. (2) The computation of degrees of freedom/number of parameters (e.g., whether variance parameters are included in the total) varies enormously between packages. As long as the df computations for a given set of models is consistent, differences don't matter, but one needs to be careful with log likelihoods and models taken from different packages. If necessary one can change the degrees of freedom manually by saying `attr(obj,"df") <- df.new`, where `df.new` is the desired number of parameters. (3) Defaults have changed to `sort=TRUE`, `base=FALSE`, `delta=TRUE`, to match my conviction that it rarely makes sense to report the overall values of information criteria

Author(s)

Ben Bolker

References

Burnham and Anderson 2002

Examples

```
set.seed(101)
d <- data.frame(x=1:20,y=rpois(20,lambda=2))
m0 <- glm(y~1,data=d)
m1 <- update(m0,.~x)
m2 <- update(m0,.~poly(x,2))
AICtab(m0,m1,m2,mnames=LETTERS[1:3])
AICtab(m0,m1,m2,base=TRUE,logLik=TRUE)
AICtab(m0,m1,m2,logLik=TRUE)
AICctab(m0,m1,m2,weights=TRUE)
print(AICctab(m0,m1,m2,weights=TRUE),min.weight=0.1)
```

mle2 *Maximum Likelihood Estimation*

Description

Estimate parameters by the method of maximum likelihood.

Usage

```
mle2(minuslogl, start, method, optimizer,
    fixed = NULL, data=NULL,
    subset=NULL,
default.start=TRUE, eval.only = FALSE, vecpar=FALSE,
parameters=NULL,
parnames=NULL,
skip.hessian=FALSE,
hessian.opts=NULL,
use.ginv=TRUE,
trace=FALSE,
browse_obj=FALSE,
gr=NULL,
optimfun,
namedrop_args=TRUE,
...)
calc_mle2_function(formula,parameters, links, start,
   parnames, use.deriv=FALSE, data=NULL,trace=FALSE)
```

Arguments

minuslogl	Function to calculate negative log-likelihood, or a formula
start	Named list. Initial values for optimizer
method	Optimization method to use. See optim.
optimizer	Optimization function to use. Currently available choices are "optim" (the default), "nlm", "nlminb", "constrOptim", "optimx", and "optimize". If "optimx" is used, (1) the optimx package must be explicitly loaded with load or require(*Warning:* Options other than the default may be poorly tested, use with caution.)
fixed	Named list. Parameter values to keep fixed during optimization.
data	list of data to pass to negative log-likelihood function: must be specified if minuslogl is specified as a formula
subset	logical vector for subsetting data (STUB)
default.start	Logical: allow default values of minuslogl as starting values?
eval.only	Logical: return value of minuslogl(start) rather than optimizing
vecpar	Logical: is first argument a vector of all parameters? (For compatibility with optim.) If vecpar is TRUE, then you should use parnames to define the parameter names for the negative log-likelihood function.

parameters	List of linear models for parameters. *MUST BE SPECIFIED IN THE SAME ORDER as the start vector (this is a bug/restriction that I hope to fix soon, but in the meantime beware)*
links	(unimplemented) specify transformations of parameters
parnames	List (or vector?) of parameter names
gr	gradient function
...	Further arguments to pass to optimizer
formula	a formula for the likelihood (see Details)
trace	Logical: print parameter values tested?
browse_obj	Logical: drop into browser() within the objective function?
skip.hessian	Bypass Hessian calculation?
hessian.opts	Options for Hessian calculation, passed through to the hessian function
use.ginv	Use generalized inverse (ginv) to compute approximate variance-covariance
optimfun	user-supplied optimization function. Must take exactly the same arguments and return exactly the same structure as optim.
use.deriv	(experimental, not yet implemented): construct symbolic derivatives based on formula?
namedrop_args	hack: drop names in sub-lists occurring in data?

Details

The optim optimizer is used to find the minimum of the negative log-likelihood. An approximate covariance matrix for the parameters is obtained by inverting the Hessian matrix at the optimum.

The minuslogl argument can also specify a formula, rather than an objective function, of the form x~ddistn(param1,...,paramn). In this case ddistn is taken to be a probability or density function, which must have (literally) x as its first argument (although this argument may be interpreted as a matrix of multivariate responses) and which must have a log argument that can be used to specify the log-probability or log-probability-density is required. If a formula is specified, then parameters can contain a list of linear models for the parameters.

If a formula is given and non-trivial linear models are given in parameters for some of the variables, then model matrices will be generated using model.matrix. start can be given:

- as a list containing lists, with each list corresponding to the starting values for a particular parameter;
- just for the higher-level parameters, in which case all of the additional parameters generated by model.matrix will be given starting values of zero (unless a no-intercept formula with -1 is specified, in which case all the starting values for that parameter will be set equal)
- (to be implemented!) as an exhaustive (flat) list of starting values (in the order given by model.matrix)

The trace argument applies only when a formula is specified. If you specify a function, you can build in your own print() or cat() statement to trace its progress. (You can also specify a value for trace as part of a control list for optim(): see optim.)

The skip.hessian argument is useful if the function is crashing with a "non-finite finite difference value" error when trying to evaluate the Hessian, but will preclude many subsequent confidence interval calculations. (You will know the Hessian is failing if you use method="Nelder-Mead" and still get a finite-difference error.)

If convergence fails, see the manual page of the relevant optimizer (optim by default, but possibly nlm, nlminb, optimx, or constrOptim if you have set the value of optimizer) for the meanings of the error codes/messages.

Value

An object of class "mle2".

Warning

Do not use a higher-level variable named .i in parameters – this is reserved for internal use.

Note

Note that the minuslogl function should return the negative log-likelihood, -log L (not the log-likelihood, log L, nor the deviance, -2 log L). It is the user's responsibility to ensure that the likelihood is correct, and that asymptotic likelihood inference is valid (e.g. that there are "enough" data and that the estimated parameter values do not lie on the boundary of the feasible parameter space).

If lower, upper, control$parscale, or control$ndeps are specified for optim fits, they must be named vectors.

The requirement that data be specified when using the formula interface is relatively new: it saves many headaches on the programming side when evaluating the likelihood function later on (e.g. for profiling or constructing predictions). Since data.frame uses the names of its arguments as column names by default, it is probably the easiest way to package objects that are lying around in the global workspace for use in mle2 (provided they are all of the same length).

When optimizer is set to "optimx" and multiple optimization methods are used (i.e. the methods argument has more than one element, or all.methods=TRUE is set in the control options), the best (minimum negative log-likelihood) solution will be saved, regardless of reported convergence status (and future operations such as profiling on the fit will only use the method that found the best result).

See Also

mle2-class

Examples

```
x <- 0:10
y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
d <- data.frame(x,y)

## in general it is best practice to use the `data' argument,
##  but variables can also be drawn from the global environment
LL <- function(ymax=15, xhalf=6)
    -sum(stats::dpois(y, lambda=ymax/(1+x/xhalf), log=TRUE))
## uses default parameters of LL
```

```
(fit <- mle2(LL))
fit1F <- mle2(LL, fixed=list(xhalf=6))
coef(fit1F)
coef(fit1F,exclude.fixed=TRUE)

(fit0 <- mle2(y~dpois(lambda=ymean),start=list(ymean=mean(y)),data=d))
anova(fit0,fit)
summary(fit)
logLik(fit)
vcov(fit)
p1 <- profile(fit)
plot(p1, absVal=FALSE)
confint(fit)

## use bounded optimization
## the lower bounds are really > 0, but we use >=0 to stress-test
## profiling; note lower must be named
(fit1 <- mle2(LL, method="L-BFGS-B", lower=c(ymax=0, xhalf=0)))
p1 <- profile(fit1)

plot(p1, absVal=FALSE)
## a better parameterization:
LL2 <- function(lymax=log(15), lxhalf=log(6))
    -sum(stats::dpois(y, lambda=exp(lymax)/(1+x/exp(lxhalf)), log=TRUE))
(fit2 <- mle2(LL2))
plot(profile(fit2), absVal=FALSE)
exp(confint(fit2))
vcov(fit2)
cov2cor(vcov(fit2))

mle2(y~dpois(lambda=exp(lymax)/(1+x/exp(lhalf))),
   start=list(lymax=0,lhalf=0),
   data=d,
   parameters=list(lymax~1,lhalf~1))

## Not run:
## try bounded optimization with nlminb and constrOptim
(fit1B <- mle2(LL, optimizer="nlminb", lower=c(lymax=1e-7, lhalf=1e-7)))
p1B <- profile(fit1B)
confint(p1B)
(fit1C <- mle2(LL, optimizer="constrOptim", ui = c(lymax=1,lhalf=1), ci=2,
   method="Nelder-Mead"))

set.seed(1001)
lymax <- c(0,2)
lhalf <- 0
x <- sort(runif(200))
g <- factor(sample(c("a","b"),200,replace=TRUE))
y <- rnbinom(200,mu=exp(lymax[g])/(1+x/exp(lhalf)),size=2)
d2 <- data.frame(x,g,y)

fit3 <- mle2(y~dnbinom(mu=exp(lymax)/(1+x/exp(lhalf)),size=exp(logk)),
    parameters=list(lymax~g),data=d2,
```

```
    start=list(lymax=0,lhalf=0,logk=0))

## End(Not run)
```

`mle2-class` *Class "mle2". Result of Maximum Likelihood Estimation.*

Description

This class encapsulates results of a generic maximum likelihood procedure.

Objects from the Class

Objects can be created by calls of the form `new("mle2", ...)`, but most often as the result of a call to `mle2`.

Slots

`call`: (language) The call to `mle2`.

`call.orig`: (language) The call to `mle2`, saved in its original form (i.e. without data arguments evaluated).

`coef`: (numeric) Vector of estimated parameters.

`data`: (data frame or list) Data with which to evaluate the negative log-likelihood function

`fullcoef`: (numeric) Fixed and estimated parameters.

`vcov`: (numeric matrix) Approximate variance-covariance matrix, based on the second derivative matrix at the MLE.

`min`: (numeric) Minimum value of objective function = minimum negative log-likelihood.

`details`: (list) Return value from `optim`.

`minuslogl`: (function) The negative log-likelihood function.

`optimizer`: (character) The optimizing function used.

`method`: (character) The optimization method used.

`formula`: (character) If a formula was specified, a character vector giving the formula and parameter specifications.

Methods

coef `signature(object = "mle2")`: Extract coefficients. If `exclude.fixed=TRUE` (it is `FALSE` by default), only the non-fixed parameter values are returned.

confint `signature(object = "mle2")`: Confidence intervals from likelihood profiles, or quadratic approximations, or root-finding.

show `signature(object = "mle2")`: Display object briefly.

show `signature(object = "summary.mle2")`: Display object briefly.

summary `signature(object = "mle2")`: Generate object summary.

update `signature(object = "mle2")`: Update fit.

vcov `signature(object = "mle2")`: Extract variance-covariance matrix.

formula `signature(object="mle2")`: Extract formula

plot `signature(object="profile.mle2,missing")`: Plot profile.

Details on the confint method

When the parameters in the original fit are constrained using `lower` or `upper`, or when `prof.lower` or `prof.upper` are set, and the confidence intervals lie outside the constraint region, `confint` will return `NA`. This may be too conservative – in some cases, the appropriate answer would be to set the confidence limit to the lower/upper bound as appropriate – but it is the most general answer.

(If you have a strong opinion about the need for a new option to `confint` that sets the bounds to the limits automatically, please contact the package maintainer.)

Examples

```
x <- 0:10
y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
lowerbound <- c(a=2,b=-0.2)
d <- data.frame(x,y)
fit1 <- mle2(y~dpois(lambda=exp(a+b*x)),start=list(a=0,b=2),data=d,
method="L-BFGS-B",lower=c(a=2,b=-0.2))
(cc <- confint(fit1,quietly=TRUE))
## to set the lower bounds to the limit
na_lower <- is.na(cc[,1])
cc[na_lower,1] <- lowerbound[na_lower]
cc
```

`mle2.options` *Options for maximum likelihood estimation*

Description

Query or set MLE parameters

Usage

```
mle2.options(...)
```

Arguments

`...`	names of arguments to query, or a list of values to set

Details

optim.method name of optimization method (see `optim` for choices)

confint name of confidence interval method: choices are "spline", "uniroot", "hessian" corresponding to spline inversion, attempt to find best answer via uniroot, information-matrix approximation

optimizer optimization function to use by default (choices: "optim", "nlm", "nlminb", "constrOptim")

Value

Values of queried parameters, or (invisibly) the full list of parameters

See Also

`mle2-class`

---

namedrop *drop unneeded names from list elements*

Description

goes through a list (containing a combination of single- and multiple-element vectors) and removes redundant names that will make trouble for mle

Usage

```
namedrop(x)
```

Arguments

`x`	a list of named or unnamed, typically numeric, vectors

Details

examines each element of `x`. If the element has length one and is a named vector, the name is removed; if `length(x)` is greater than 1, but all the names are the same, the vector is renamed

Value

the original list, with names removed/added

Author(s)

Ben Bolker

Examples

```
x = list(a=c(a=1),b=c(d=1,d=2),c=c(a=1,b=2,c=3))
names(unlist(namedrop(x)))
names(unlist(namedrop(x)))
```

---

parnames *get and set parameter names*

Description

Gets and sets the "parnames" attribute on a negative log-likelihood function

Usage

```
parnames(obj)
parnames(obj) <- value
```

Arguments

obj	a negative log-likelihood function
value	a character vector of parameter names

Details

The parnames attribute is used by mle2() when the negative log-likelihood function takes a parameter vector, rather than a list of parameters; this allows users to use the same objective function for optim() and mle2()

Value

Returns the parnames attribute (a character vector of parameter names) or sets it.

Author(s)

Ben Bolker

Examples

```
x <- 1:5
set.seed(1001)
y <- rbinom(5,prob=x/(1+x),size=10)
mfun <- function(p) {
  a <- p[1]
  b <- p[2]
  -sum(dbinom(y,prob=a*x/(b+x),size=10,log=TRUE))
}
optim(fn=mfun,par=c(1,1))
parnames(mfun) <- c("a","b")
mle2(minuslogl=mfun,start=c(a=1,b=1),method="Nelder-Mead")
```

pop_pred_samp *generate population prediction sample from parameters*

Description

This [EXPERIMENTAL] function combines several sampling tricks to compute a version of an importance sample (based on flat priors) for the parameters.

Usage

```
pop_pred_samp(
  object,
  n = 1000,
  n_imp = n * 10,
  return_wts = FALSE,
  impsamp = FALSE,
  PDify = FALSE,
  PDmethod = NULL,
  Sigma = vcov(object),
  tol = 1e-06,
  return_all = FALSE,
  rmvnorm_method = c("mvtnorm", "MASS"),
  fix_params = NULL,
  ...
)
```

Arguments

object	a fitted mle2 object
n	number of samples to return
n_imp	number of total samples from which to draw, if doing importance sampling
return_wts	return a column giving the weights of the samples, for use in weighted summaries?
impsamp	subsample values (with replacement) based on their weights?
PDify	use Gill and King generalized-inverse procedure to correct non-positive-definite variance-covariance matrix if necessary?
PDmethod	method for fixing non-positive-definite covariance matrices
tol	tolerance for detecting small eigenvalues
return_all	return a matrix including all values, and weights (rather than taking a sample)
rmvnorm_method	package to use for generating MVN samples
fix_params	parameters to fix (in addition to parameters that were fixed during estimation)
Sigma	covariance matrix for sampling
...	additional parameters to pass to the negative log-likelihood function

References

Gill, Jeff, and Gary King. "What to Do When Your Hessian Is Not Invertible: Alternatives to Model Respecification in Nonlinear Estimation." Sociological Methods & Research 33, no. 1 (2004): 54-87. Lande, Russ and Steinar Engen and Bernt-Erik Saether, Stochastic Population Dynamics in Ecology and Conservation. Oxford University Press, 2003.

`predict-methods` *Predicted values from an mle2 fit*

Description

Given an `mle2` fit and an optional list of new data, return predictions (more generally, summary statistics of the predicted distribution)

Usage

```
    ## S4 method for signature 'mle2'
predict(object, newdata=NULL,
                        location="mean", newparams=NULL, ...)
    ## S4 method for signature 'mle2'
simulate(object, nsim,
                        seed, newdata=NULL, newparams=NULL, ...)
    ## S4 method for signature 'mle2'
residuals(object,type=c("pearson","response"),
                   location="mean",...)
```

Arguments

`object`	an mle2 object
`newdata`	optional list of new data
`newparams`	optional vector of new parameters
`location`	name of the summary statistic to return
`nsim`	number of simulations
`seed`	random number seed
`type`	residuals type
`...`	additional arguments (for generic compatibility)

Methods

x = "mle2" an `mle2` fit

Note

For some models (e.g. constant models), `predict` may return a single value rather than a vector of the appropriate length.

Examples

```
set.seed(1002)
lymax <- c(0,2)
lhalf <- 0
x <- runif(200)
g <- factor(rep(c("a","b"),each=100))
y <- rnbinom(200,mu=exp(lymax[g])/(1+x/exp(lhalf)),size=2)
dat <- data.frame(y,g,x)

fit3 <- mle2(y~dnbinom(mu=exp(lymax)/(1+x/exp(lhalf)),size=exp(logk)),
    parameters=list(lymax~g),
    start=list(lymax=0,lhalf=0,logk=0),
data=dat)

plot(y~x,col=g)
## true curves
curve(exp(0)/(1+x/exp(0)),add=TRUE)
curve(exp(2)/(1+x/exp(0)),col=2,add=TRUE)
## model predictions
xvec = seq(0,1,length=100)
lines(xvec,predict(fit3,newdata=list(g=factor(rep("a",100),levels=c("a","b")),
                                x = xvec)),col=1,lty=2)
lines(xvec,predict(fit3,newdata=list(g=factor(rep("b",100),levels=c("a","b")),
                                x = xvec)),col=2,lty=2)


## comparing automatic and manual predictions
p1 = predict(fit3)
p2A =
with(as.list(coef(fit3)),exp(`lymax.(Intercept)`)/(1+x[1:100]/exp(lhalf)))
p2B =
with(as.list(coef(fit3)),exp(`lymax.(Intercept)`+lymax.gb)/(1+x[101:200]/exp(lhalf)))
all(p1==c(p2A,p2B))
##
simulate(fit3)
```

`profile-methods` *Likelihood profiles*

Description

Compute likelihood profiles for a fitted model

Usage

```
proffun(fitted, which = 1:p, maxsteps = 100,
                    alpha = 0.01, zmax = sqrt(qchisq(1 - alpha/2, p)),
                    del = zmax/5, trace = FALSE, skiperrs=TRUE,
                    std.err,
```

```
                    tol.newmin = 0.001, debug=FALSE,
                    prof.lower, prof.upper,
                    skip.hessian = TRUE,
                    continuation = c("none","naive","linear"),
                    try_harder=FALSE, ...)
## S4 method for signature 'mle2'
profile(fitted, ...)
```

Arguments

fitted	A fitted maximum likelihood model of class "mle2"
which	a numeric or character vector describing which parameters to profile (default is to profile all parameters)
maxsteps	maximum number of steps to take looking for an upper value of the negative log-likelihood
alpha	maximum (two-sided) likelihood ratio test confidence level to find
zmax	maximum value of signed square root of deviance difference to find (default value corresponds to a 2-tailed chi-squared test at level alpha)
del	step size for profiling
trace	(logical) produce tracing output?
skiperrs	(logical) ignore errors produced during profiling?
std.err	Optional numeric vector of standard errors, for cases when the Hessian is badly behaved. Will be replicated if necessary, and NA values will be replaced by the corresponding values from the fit summary
tol.newmin	tolerance for diagnosing a new minimum below the minimum deviance estimated in initial fit is found
debug	(logical) debugging output?
prof.lower	optional vector of lower bounds for profiles
prof.upper	optional vector of upper bounds for profiles
continuation	use continuation method to set starting values? "none" sets starting values to best fit; "naive" sets starting values to those of previous profiling fit; "linear" (not yet implemented) would use linear extrapolation from the previous two profiling fits
skip.hessian	skip hessian (defunct?)
try_harder	(logical) ignore NA and flat spots in the profile, try to continue anyway?
...	additional arguments (not used)

Details

proffun is the guts of the profile method, exposed so that other packages can use it directly.

See the vignette (vignette("mle2",package="bbmle")) for more technical details of how profiling is done.

See Also

profile.mle-class

profile.mle2-class *Methods for likelihood profiles*

Description

Definition of the mle2 likelihood profile class, and applicable methods

Usage

```
## S4 method for signature 'profile.mle2'
plot(x,
   levels, which=1:p, conf = c(99, 95, 90, 80, 50)/100,
   plot.confstr = TRUE,
   confstr = NULL, absVal = TRUE, add = FALSE,
   col.minval="green", lty.minval=2,
   col.conf="magenta", lty.conf=2,
   col.prof="blue", lty.prof=1,
   xlabs=nm, ylab="z",
   onepage=TRUE,
   ask=((prod(par("mfcol")) < length(which)) && dev.interactive() &&
               !onepage),
   show.points=FALSE,
   main, xlim, ylim, ...)
## S4 method for signature 'mle2'
confint(object, parm, level = 0.95, method,
          trace=FALSE,quietly=!interactive(),
          tol.newmin=0.001,...)
## S4 method for signature 'profile.mle2'
confint(object, parm, level = 0.95, trace=FALSE, ...)
```

Arguments

x	An object of class profile.mle2
object	An object of class mle2 or profile.mle2 (as appropriate)
levels	levels at which to plot likelihood cutoffs (set by conf by default)
level	level at which to compute confidence interval
which	(numeric or character) which parameter profiles to plot
parm	(numeric or character) which parameter(s) to find confidence intervals for
method	(character) "spline", "uniroot", or "quad", for spline-extrapolation-based (default), root-finding, or quadratic confidence intervals. By default it uses the value of mle2.options("confint") – the factory setting is "spline".
trace	trace progress of confidence interval calculation when using 'uniroot' method?
conf	(1-alpha) levels at which to plot likelihood cutoffs/confidence intervals
quietly	(logical) suppress "Profiling ..." message when computing profile to get confidence interval?

tol.newmin	see profile-methods
plot.confstr	(logical) plot labels showing confidence levels?
confstr	(character) labels for confidence levels (by default, constructed from conf levels)
absVal	(logical) plot absolute values of signed square root deviance difference ("V" plot rather than straight-line plot)?
add	(logical) add profile to existing graph?
col.minval	color for minimum line
lty.minval	line type for minimum line
col.conf	color for confidence intervals
lty.conf	line type for confidence intervals
col.prof	color for profile
lty.prof	line type for profile
xlabs	x labels
ylab	y label
onepage	(logical) plot all profiles on one page, adjusting par(mfcol) as necessary?
ask	(logical) pause for user input between plots?
show.points	(logical) show computed profile points as well as interpolated spline?
main	(logical) main title
xlim	x limits
ylim	y limits
...	other arguments

Details

The default confidence interval calculation computes a likelihood profile and uses the points therein, or uses the computed points in an existing `profile.mle2` object, to construct an interpolation spline (which by default has three times as many points as were in the original set of profile points). It then uses linear interpolation between these interpolated points (!)

Objects from the Class

Objects can be created by calls of the form `new("profile.mle2", ...)`, but most often by invoking `profile` on an "mle2" object.

Slots

`profile`**:** Object of class `"list"`. List of profiles, one for each requested parameter. Each profile is a data frame with the first column called `z` being the signed square root of the deviance, and the others being the parameters with names prefixed by `par.vals`.

`summary`**:** Object of class `"summary.mle2"`. Summary of object being profiled.

Methods

confint `signature(object = "profile.mle2")`: Use profile to generate approximate confidence intervals for parameters.

plot `signature(x = "profile.mle2", y = "missing")`: Plot profiles for each parameter.

summary `signature(x = "profile.mle2")`: Plot profiles for each parameter.

show `signature(object = "profile.mle2")`: Show object.

See Also

`mle2`, `mle2-class`, `summary.mle2-class`

Examples

```
x <- 0:10
y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
d <- data.frame(x,y)
## we have a choice here: (1) don't impose boundaries on the parameters,
##  put up with warning messages about NaN values:
fit1 <- mle2(y~dpois(lambda=ymax/(1+x/xhalf)),
     start=list(ymax=1,xhalf=1),
     data=d)
p1 <- suppressWarnings(profile(fit1))
plot(p1,main=c("first","second"),
     xlab=c(~y[max],~x[1/2]),ylab="Signed square root deviance",
     show.points=TRUE)
suppressWarnings(confint(fit1)) ## recomputes profile
confint(p1)  ## operates on existing profile
suppressWarnings(confint(fit1,method="uniroot"))
## alternatively, we can use box constraints to keep ourselves
##  to positive parameter values ...
fit2 <- update(fit1,method="L-BFGS-B",lower=c(ymax=0.001,xhalf=0.001))
## Not run:
p2 <- profile(fit2)
plot(p2,show.points=TRUE)
## but the fit for ymax is just bad enough that the spline gets wonky
confint(p2)  ## now we get a warning
confint(fit2,method="uniroot")
## bobyqa is a better-behaved bounded optimizer ...
##  BUT recent (development, 2012.5.24) versions of
##    optimx no longer allow single-parameter fits!
if (require(optimx)) {
  fit3 <- update(fit1,
      optimizer="optimx",
      method="bobyqa",lower=c(ymax=0.001,xhalf=0.001))
   p3 <- profile(fit3)
   plot(p3,show.points=TRUE)
  confint(p3)
}

## End(Not run)
```

relist2 *reconstruct the structure of a list*

Description

reshapes a vector according to a list template

Usage

```
relist2(v, l)
```

Arguments

v	vector, probably numeric, of values to reshape
l	template list giving structure

Details

attempts to coerce v into a list with the same structure and names as l

Value

a list with values corresponding to v and structure corresponding to l

Author(s)

Ben Bolker

Examples

```
l = list(b=1,c=2:5,d=matrix(1:4,nrow=2))
relist2(1:9,l)
```

sbinom *Abstract definitions of distributions*

Description

Functions returning values for summary statistics (mean, median, etc.) of distributions

Usage

```
sbeta(shape1, shape2)
sbetabinom(size, prob, theta)
sbinom(size, prob)
snbinom(size, prob, mu)
snorm(mean, sd)
spois(lambda)
slnorm(meanlog, sdlog)
```

Arguments

prob	probability as defined for dbinom, dnbinom, or beta-binomial distribution (dbetabinom in the emdbook package)
size	size parameter as defined for dbinom or dbetabinom in the emdbook package, or size/overdispersion parameter as in dnbinom
mean	mean parameter as defined for dnorm
mu	mean parameter as defined for dnbinom
sd	standard deviation parameter as defined for dnorm
shape1	shape parameter for dbeta
shape2	shape parameter for dbeta
lambda	rate parameter as defined for dpois
theta	overdispersion parameter for beta-binomial (see dbetabinom in the emdbook package)
meanlog	as defined for dlnorm
sdlog	as defined for dlnorm

Value

title	name of the distribution
[parameters]	input parameters for the distribution
mean	theoretical mean of the distribution
median	theoretical median of the distribution
mode	theoretical mode of the distribution
variance	theoretical variance of the distribution
sd	theoretical standard deviation of the distribution

Note

these definitions are tentative, subject to change as I figure this out better. Perhaps construct functions that return functions? Strip down results? Do more automatically?

Author(s)

Ben Bolker

See Also

dbinom, dpois, dnorm, dnbinom

Examples

```
sbinom(prob=0.2,size=10)
snbinom(mu=2,size=1.2)
```

---

slice *Calculate likelihood "slices"*

Description

Computes cross-section(s) of a multi-dimensional likelihood surface

Usage

```
slice(x, dim=1, ...)
sliceOld(fitted, which = 1:p, maxsteps = 100,
                       alpha = 0.01, zmax = sqrt(qchisq(1 - alpha/2, p)),
                       del = zmax/5, trace = FALSE,
                       tol.newmin=0.001, ...)
slice1D(params,fun,nt=101,lower=-Inf,
                     upper=Inf,verbose=TRUE, tranges=NULL,
                     fun_args = NULL,
                     ...)
slice2D(params,fun,nt=31,lower=-Inf,
                     upper=Inf,
                     cutoff=10,verbose=TRUE,
                     tranges=NULL,
                     ...)
slicetrans(params, params2, fun, extend=0.1, nt=401,
                       lower=-Inf, upper=Inf)
```

Arguments

x	a fitted model object of some sort
dim	dimensionality of slices (1 or 2)
params	a named vector of baseline parameter values
params2	a vector of parameter values
fun	an objective function
fun_args	additional arguments to pass to fun
nt	(integer) number of slice-steps to take
lower	lower bound(s) (stub?)
upper	upper bound(s) (stub?)

cutoff	maximum increase in objective function to allow when computing ranges
extend	(numeric) fraction by which to extend range beyond specified points
verbose	print verbose output?
fitted	A fitted maximum likelihood model of class "mle2"
which	a numeric or character vector describing which parameters to profile (default is to profile all parameters)
maxsteps	maximum number of steps to take looking for an upper value of the negative log-likelihood
alpha	maximum (two-sided) likelihood ratio test confidence level to find
zmax	maximum value of signed square root of deviance difference to find (default value corresponds to a 2-tailed chi-squared test at level alpha)
del	step size for profiling
trace	(logical) produce tracing output?
tol.newmin	tolerance for diagnosing a new minimum below the minimum deviance estimated in initial fit is found
tranges	a two-column matrix giving lower and upper bounds for each parameter
...	additional arguments (not used)

Details

Slices provide a lighter-weight way to explore likelihood surfaces than profiles, since they vary a single parameter rather than optimizing over all but one or two parameters.

slice is a generic method

slice1D creates one-dimensional slices, by default of all parameters of a model

slice2D creates two-dimensional slices, by default of all pairs of parameters in a model. In each panel the closed point represents the parameters given (typically the MLEs), while the open point represents the observed minimum value within the 2D slice. If everything has gone according to plan, these points should coincide (at least up to grid precision).

slicetrans creates a slice along a transect between two specified points in parameter space (see `calcslice` in the `emdbook` package)

Value

An object of class `slice` with

slices a list of individual parameter (or parameter-pair) slices, each of which is a data frame with elements

- **var1** name of the first variable
- **var2** (for 2D slices) name of the second variable
- **x** parameter values
- **y** (for 2D slices) parameter values
- **z** slice values

ranges a list (?) of the ranges for each parameter

params vector of baseline parameter values

dim 1 or 2

`sliceOld` returns instead a list with elements `profile` and `summary` (see `profile.mle2`)

Author(s)

Ben Bolker

See Also

`profile`

Examples

```
x <- 0:10
y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
d <- data.frame(x,y)
fit1 <- mle2(y~dpois(lambda=exp(lymax)/(1+x/exp(lhalf))),
   start=list(lymax=0,lhalf=0),
   data=d)
s1 <- bbmle::slice(fit1,verbose=FALSE)
s2 <- bbmle::slice(fit1,dim=2,verbose=FALSE)
require(lattice)
plot(s1)
plot(s2)
## 'transect' slice, from best-fit values to another point
st <- bbmle::slice(fit1,params2=c(5,0.5))
plot(st)
```

---

slice.mle2-class *likelihood-surface slices*

Description

evaluations of log-likelihood along transects in parameter space

Objects from the Class

Objects can be created by calls of the form `new("slice.mle2", ...)`. The objects are similar to likelihood profiles, but don't involve any optimization with respect to the other parameters.

Slots

`profile`**:** Object of class `"list"`. List of slices, one for each requested parameter. Each slice is a data frame with the first column called `z` being the signed square root of the -2 log likelihood ratio, and the others being the parameters with names prefixed by `par.vals`.

`summary`**:** Object of class `"summary.mle2"`. Summary of object being profiled.

Methods

plot `signature(x = "profile.mle2", y = "missing")`: Plot profiles for each parameter.

See Also

`profile.mle2-class`

`strwrapx` *Wrap strings at white space and + symbols*

Description

Extended (hacked) version of strwrap: wraps a string at whitespace and plus symbols

Usage

```
strwrapx(x, width = 0.9 * getOption("width"), indent = 0,
exdent = 0, prefix = "", simplify = TRUE,
parsplit = "\n[ \t\n]*\n", wordsplit = "[ \t\n]")
```

Arguments

`x`	a character vector, or an object which can be converted to a character vector by `as.character`.
`width`	a positive integer giving the target column for wrapping lines in the output.
`indent`	a non-negative integer giving the indentation of the first line in a paragraph.
`exdent`	a non-negative integer specifying the indentation of subsequent lines in paragraphs.
`prefix`	a character string to be used as prefix for each line.
`simplify`	a logical. If `TRUE`, the result is a single character vector of line text; otherwise, it is a list of the same length as `x` the elements of which are character vectors of line text obtained from the corresponding element of `x`. (Hence, the result in the former case is obtained by unlisting that of the latter.)
`parsplit`	Regular expression describing how to split paragraphs
`wordsplit`	Regular expression decribing how to split words

Details

Whitespace in the input is destroyed. Double spaces after periods (thought as representing sentence ends) are preserved. Currently, possible sentence ends at line breaks are not considered specially.

Indentation is relative to the number of characters in the prefix string.

Examples

```
## Read in file 'THANKS'.
x <- paste(readLines(file.path(R.home("doc"), "THANKS")), collapse = "\n")
## Split into paragraphs and remove the first three ones
x <- unlist(strsplit(x, "\n[ \t\n]*\n"))[-(1:3)]
## Join the rest
x <- paste(x, collapse = "\n\n")
## Now for some fun:
writeLines(strwrap(x, width = 60))
writeLines(strwrap(x, width = 60, indent = 5))
writeLines(strwrap(x, width = 60, exdent = 5))
writeLines(strwrap(x, prefix = "THANKS> "))

## Note that messages are wrapped AT the target column indicated by
## 'width' (and not beyond it).
## From an R-devel posting by J. Hosking <jh910@juno.com>.
x <- paste(sapply(sample(10, 100, rep=TRUE),
           function(x) substring("aaaaaaaaaa", 1, x)), collapse = " ")
sapply(10:40,
       function(m)
       c(target = m, actual = max(nchar(strwrap(x, m)))))
```

---

summary.mle2-class *Class "summary.mle2", summary of "mle2" objects*

Description

Extract of "mle2" object

Objects from the Class

Objects can be created by calls of the form `new("summary.mle2", ...)`, but most often by invoking `summary` on an "mle2" object. They contain values meant for printing by `show`.

Slots

`call`: Object of class `"language"` The call that generated the "mle2" object.

`coef`: Object of class `"matrix"`. Estimated coefficients and standard errors

`m2logL`: Object of class `"numeric"`. Minus twice the log likelihood.

Methods

show `signature(object = "summary.mle2")`: Pretty-prints `object`

coef `signature(object = "summary.mle2")`: Extracts the contents of the `coef` slot

See Also

`summary`, `mle2`, `mle2-class`

Index