

Package ‘AICcmodavg’

May 6, 2026

Type Package

Title Model Selection and Multimodel Inference Based on (Q)AIC(c)

Version 2.3-4

Date 2025-03-03

Maintainer Marc J. Mazerolle <marc.mazerolle@sbf.ulaval.ca>

Depends R (>= 3.2.0)

Imports methods, stats, graphics, lattice, MASS, Matrix, nlme, stats4, survival, unmarked, VGAM, xtable

Suggests betareg, coxme, fitdistrplus, glmmTMB, lavaan, lme4, maxlike, nnet, ordinal, pscl, R2jags, R2OpenBUGS, R2WinBUGS, jagsUI, lmerTest

Description

Functions to implement model selection and multimodel inference based on Akaike's information criterion (AIC) and the second-order AIC (AICc), as well as their quasi-likelihood counterparts (QAIC, QAICc) from various model object classes. The package implements classic model averaging for a given parameter of interest or predicted values, as well as a shrinkage version of model averaging parameter estimates or effect sizes. The package includes diagnostics and goodness-of-fit statistics for certain model types including those of 'unmarked-Fit' classes estimating demographic parameters after accounting for imperfect detection probabilities. Some functions also allow the creation of model selection tables for Bayesian models of the 'bugs', 'rjags', and 'jagsUI' classes. Functions also implement model selection using BIC. Objects following model selection and multimodel inference can be formatted to LaTeX using 'xtable' methods included in the package.

License GPL (>= 2)

LazyLoad yes

Repository CRAN

NeedsCompilation no

Author Marc J. Mazerolle [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-0486-0310>>)

Date/Publication 2025-03-06 14:30:06 UTC

Contents

AICmodavg-package	3
AICc	11
AICcCustom	16
AICmodavg-defunct	18
aictab	23
aictabCustom	32
anovaOD	35
beetle	40
bictab	41
bictabCustom	49
boot.wt	52
bullfrog	56
calcium	57
cement	58
checkConv	59
checkParms	61
confset	65
countDist	69
countHist	73
covDiag	77
c_hat	79
detHist	82
detTime	86
DIC	89
dictab	91
dry.frog	93
evidence	95
extractCN	97
extractLL	99
extractSE	101
extractX	102
fam.link.mer	108
fat	110
gpa	112
ictab	113
importance	115
iron	123
lizards	124
mb.gof.test	127
min.trap	132
modavg	133
modavg.utility	146
modavgCustom	148
modavgEffect	151
modavgIC	165
modavgPred	168

modavgShrink	179
multComp	189
newt	196
Nmix.gof.test	197
pine	201
predictSE	202
salamander	205
summaryOD	206
tortoise	211
turkey	212
useBIC	213
useBICCustom	217
xtable	219

Index	227
--------------	------------

AICcmodavg-package *Model Selection and Multimodel Inference Based on (Q)AIC(c)*

Description

Description: This package includes functions to create model selection tables based on Akaike's information criterion (AIC) and the second-order AIC (AICc), as well as their quasi-likelihood counterparts (QAIC, QAICc). The package also features functions to conduct classic model averaging (multimodel inference) for a given parameter of interest or predicted values, as well as a shrinkage version of model averaging parameter estimates. Other handy functions enable the computation of relative variable importance, evidence ratios, and confidence sets for the best model. The present version supports Cox proportional hazards models and conditional logistic regression (coxph and coxme classes), linear models (lm class), generalized linear models (glm, glm.nb, vglm, hurdle, and zeroinfl classes), linear models fit by generalized least squares (gls class), linear mixed models (lme class), generalized linear mixed models (mer, merMod, and glmmTMB classes), multinomial and ordinal logistic regressions (multinom, polr, clm, and clmm classes), robust regression models (rlm class), beta regression models (betareg class), parametric survival models (survreg class), nonlinear models (nls and gnls classes), nonlinear mixed models (nlme and nlmerMod classes), univariate models (fitdist and fitdistr classes), and certain types of latent variable models (lavaan class). The package also supports various models of unmarkedFit and maxLikeFit classes estimating demographic parameters after accounting for imperfect detection probabilities. Some functions also allow the creation of model selection tables for Bayesian models of the bugs and rjags classes. Objects following model selection and multimodel inference can be formatted to LaTeX using xtable methods included in the package.

Details

Package:	AICcmodavg
Type:	Package
Version:	2.3-4
Date:	2025-03-03

License: GPL (>=2)
 LazyLoad: yes

Many functions of the package require a list of models as the input to conduct model selection and multimodel inference. Thus, you should start by organizing the output of the models in a list (See 'Examples' below).

This package contains several useful functions for model selection and multimodel inference for several model classes:

- AICc** Computes AIC, AICc, and their quasi-likelihood counterparts (QAIC, QAICc).
- aictab** Constructs model selection tables with number of parameters, AIC, delta AIC, Akaike weights or variants based on AICc, QAIC, and QAICc for a set of candidate models.
- bictab** Constructs model selection tables with number of parameters, BIC, delta BIC, BIC weights for a set of candidate models.
- boot.wt** Computes summary statistics from detection histories.
- confset** Determines the confidence set for the best model based on one of three criteria.
- DIC** Extracts DIC.
- dictab** Constructs model selection tables with number of parameters, DIC, delta DIC, DIC weights for a set of candidate models.
- evidence** Computes the evidence ratio between the highest-ranked model based on the information criteria selected and a lower-ranked model.
- importance** Computes importance values (w+) for the support of a given parameter among set of candidate models.
- modavg** Computes model-averaged estimate, unconditional standard error, and unconditional confidence interval of a parameter of interest among a set of candidate models.
- modavgEffect** Computes model-averaged effect sizes between groups based on the entire candidate model set.
- modavgShrink** Computes shrinkage version of model-averaged estimate, unconditional standard error, and unconditional confidence interval of a parameter of interest among entire set of candidate models.
- modavgPred** Computes model-average predictions, unconditional SE's, and confidence intervals among entire set of candidate models.
- multComp** Performs multiple comparisons across levels of a factor in a model selection framework.
- useBIC** Computes BIC or a quasi-likelihood counterparts (QBIC).

For models not yet supported by the functions above, the following can be useful for model selection and multimodel inference conducted from input values supplied by the user:

- AICcCustom** Computes AIC, AICc, QAIC, and QAICc from user-supplied input values of log-likelihood and number of parameters.
- aictabCustom** Creates model selection tables based on (Q)AIC(c) from user-supplied input values of log-likelihood and number of parameters.

- `bictabCustom` Creates model selection tables based on (Q)BIC from user-supplied input values of log-likelihood and number of parameters.
- `ictab` Creates model selection tables from user-supplied values of an information criterion.
- `modavgCustom` Computes model-averaged parameter estimate based on (Q)AIC(c) from user-supplied input values of log-likelihood, number of parameters, parameter estimates, and standard errors.
- `modavgIC` Computes model-averaged parameter estimate from user-supplied values of information criterion, parameter estimates, and standard errors.
- `useBICCustom` Computes BIC and QBIC from user-supplied input values of log-likelihoods and number of parameters.

A number of functions for model diagnostics are available:

- `c_hat` Estimates variance inflation factor for binomial or Poisson GLM's based on various estimators.
- `checkConv` Checks the convergence information of the algorithm for the model.
- `checkParms` Checks the occurrence of parameter estimates with high standard errors in a model.
- `countDist` Computes summary statistics from distance sampling data.
- `countHist` Computes summary statistics from count history data.
- `covDiag` Computes covariance diagnostics for lambda in N -mixture models.
- `detHist` Computes summary statistics from detection histories.
- `detTime` Computes summary statistics from time-to-detection data.
- `extractCN` Extracts condition number from models of certain classes.
- `mb.gof.test` Computes the MacKenzie and Bailey goodness-of-fit test for single season and dynamic occupancy models using the Pearson chi-square statistic.
- `Nmix.gof.test` Computes goodness-of-fit test for N -mixture models based on the Pearson chi-square statistic.

Other utility functions include:

- `anovaOD` Computes likelihood-ratio test statistic corrected for overdispersion between two models.
- `extractLL` Extracts log-likelihood from models of certain classes.
- `extractSE` Extracts standard errors from models of certain classes and adds the labels.
- `extractX` Extracts the predictors and associated information on variables from a list of candidate models.
- `fam.link.mer` Extracts the distribution family and link function from a generalized linear mixed model of classes `mer` and `merMod`.
- `predictSE` Computes predictions and associated standard errors models of certain classes.
- `summaryOD` Displays summary of model output adjusted for overdispersion.
- `xtable` Extends methods from `xtable` to format various objects resulting from model selection and multimodel inference to LaTeX or HTML tables.

Author(s)

Marc J. Mazerolle <marc.mazerolle@uqat.ca>.

References

- Anderson, D. R. (2008) *Model-based inference in the life sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., and Anderson, D. R. (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.

Examples

```
##Example 1: Poisson GLM with offset
##anuran larvae example from Mazerolle (2006)
data(min.trap)
##assign "UPLAND" as the reference level as in Mazerolle (2006)
min.trap$type <- relevel(min.trap$type, ref = "UPLAND")

##set up candidate models in a list
Cand.mod <- list()
##global model
Cand.mod[[1]] <- glm(Num_anura ~ Type + log.Perimeter + Num_ranatra,
                    family = poisson, offset = log(Effort),
                    data = min.trap)
Cand.mod[[2]] <- glm(Num_anura ~ Type + log.Perimeter, family = poisson,
                    offset = log(Effort), data = min.trap)
Cand.mod[[3]] <- glm(Num_anura ~ Type + Num_ranatra, family = poisson,
                    offset = log(Effort), data = min.trap)
Cand.mod[[4]] <- glm(Num_anura ~ Type, family = poisson,
                    offset = log(Effort), data = min.trap)
Cand.mod[[5]] <- glm(Num_anura ~ log.Perimeter + Num_ranatra,
                    family = poisson, offset = log(Effort),
                    data = min.trap)
Cand.mod[[6]] <- glm(Num_anura ~ log.Perimeter, family = poisson,
                    offset = log(Effort), data = min.trap)
Cand.mod[[7]] <- glm(Num_anura ~ Num_ranatra, family = poisson,
                    offset = log(Effort), data = min.trap)
Cand.mod[[8]] <- glm(Num_anura ~ 1, family = poisson,
                    offset = log(Effort), data = min.trap)

##check c-hat for global model
c_hat(Cand.mod[[1]], method = "pearson") #uses Pearson's chi-square/df
##note the very low overdispersion: in this case, the analysis could be
##conducted without correcting for c-hat as its value is reasonably close
##to 1

##output of model corrected for overdispersion
summaryOD(Cand.mod[[1]], c.hat = 1.04)

##assign names to each model
```



```

## observation covariates are in site-major, observation-minor order
obsCovs(pferUMF) <- data.frame(obsvar1 = rnorm(numSites(pferUMF) *
                                     obsNum(pferUMF)))

##check detection history data from data object
detHist(pferUMF)

##set up candidate model set
fm1 <- occu(~ obsvar1 ~ sitevar1, pferUMF)
##check detection history data from model object
detHist(fm1)

fm2 <- occu(~ 1 ~ sitevar1, pferUMF)
fm3 <- occu(~ obsvar1 ~ sitevar2, pferUMF)
fm4 <- occu(~ 1 ~ sitevar2, pferUMF)
Cand.models <- list(fm1, fm2, fm3, fm4)

##assign names to elements in list
##alternative to using 'modnames' argument
names(Cand.models) <- c("fm1", "fm2", "fm3", "fm4")

##check GOF of global model and estimate c-hat
mb.gof.test(fm4, nsim = 100) #nsim should be > 1000

##check for high SE's in models
lapply(Cand.models, checkParms, simplify = FALSE)

##compute table
print(aictab(cand.set = Cand.models,
             second.ord = TRUE), digits = 4)

##export as LaTeX table
if(require(xtable)) {
  xtable(aictab(cand.set = Cand.models,
               second.ord = TRUE))
}

##compute evidence ratio
evidence(aictab(cand.set = Cand.models))
##evidence ratio between top model vs lowest-ranked model
evidence(aictab(cand.set = Cand.models), model.high = "fm2", model.low = "fm3")

##compute confidence set based on 'raw' method
confset(cand.set = Cand.models, second.ord = TRUE,
        method = "raw")

##compute importance value for "sitevar1" on occupancy
##same number of models with vs without variable
importance(cand.set = Cand.models, parm = "sitevar1",
           parm.type = "psi")

##compute model-averaged estimate of "sitevar1" on occupancy
##(natural average)

```

```

modavg(cand.set = Cand.models, parm = "sitevar1",
       parm.type = "psi")

##compute model-averaged estimate of "sitevar1"
##(shrinkage estimator)
##same number of models with vs without variable
modavgShrink(cand.set = Cand.models,
             parm = "sitevar1", parm.type = "psi")

##compute model-average predictions
##check explanatory variables appearing in models
extractX(Cand.models, parm.type = "psi")

##create a data set for predictions
dat.pred <- data.frame(sitevar1 = seq(from = min(siteCovs(pferUMF)$sitevar1),
                                     to = max(siteCovs(pferUMF)$sitevar1), by = 0.5),
                      sitevar2 = mean(siteCovs(pferUMF)$sitevar2))

##model-averaged predictions of psi across range of values
##of sitevar1 and entire model set
modavgPred(cand.set = Cand.models, newdata = dat.pred,
           parm.type = "psi")
detach(package:unmarked)

## End(Not run)

## Not run:
##Example 3: example with user-supplied values of log-likelihoods and
##number of parameters

##vector with model LL's
LL <- c(-38.8876, -35.1783, -64.8970)

##vector with number of parameters
Ks <- c(7, 9, 4)

##create a vector of names to trace back models in set
Modnames <- c("Cm1", "Cm2", "Cm3")

##generate AICc table
aictabCustom(logL = LL, K = Ks, modnames = Modnames, nobs = 121,
            sort = TRUE)
##generate AIC table
aictabCustom(logL = LL, K = Ks, modnames = Modnames,
            second.ord = FALSE, nobs = 121, sort = TRUE)

##model averaging parameter estimate
##vector of beta estimates for a parameter of interest
model.ests <- c(0.0478, 0.0480, 0.0478)

##vector of SE's of beta estimates for a parameter of interest
model.se.ests <- c(0.0028, 0.0028, 0.0034)

```

```

##compute model-averaged estimate and unconditional SE based on AICc
modavgCustom(logL = LL, K = Ks, modnames = Modnames,
             estimate = model.ests, se = model.se.ests, nobs = 121)
##compute model-averaged estimate and unconditional SE based on BIC
modavgCustom(logL = LL, K = Ks, modnames = Modnames,
             estimate = model.ests, se = model.se.ests, nobs = 121,
             useBIC = TRUE)

## End(Not run)

## Not run:
##Example 4: example with user-supplied values of information criterion
##model selection based on WAIC

##WAIC values
waic <- c(105.74, 107.36, 108.24, 100.57)
##number of effective parameters
effK <- c(7.45, 5.61, 6.14, 6.05)

##create a vector of names to trace back models in set
Modnames <- c("global model", "interactive model",
             "additive model", "invertpred model")

##generate WAIC model selection table
ictab(ic = waic, K = effK, modnames = Modnames,
      sort = TRUE, ic.name = "WAIC")

##compute model-averaged estimate
##vector of predictions
Preds <- c(0.106, 0.137, 0.067, 0.050)
##vector of SE's for prediction
Ses <- c(0.128, 0.159, 0.054, 0.039)

##compute model-averaged estimate and unconditional SE based on WAIC
modavgIC(ic = waic, K = effK, modnames = Modnames,
        estimate = Preds, se = Ses,
        ic.name = "WAIC")

##export as LaTeX table
if(require(xtable)) {
  ##model-averaged estimate and confidence interval
  xtable(modavgIC(ic = waic, K = effK, modnames = Modnames,
                estimate = Preds, se = Ses,
                ic.name = "WAIC"))
  ##model selection table with estimate and SE's from each model
  xtable(modavgIC(ic = waic, K = effK, modnames = Modnames,
                estimate = Preds, se = Ses,
                ic.name = "WAIC"), print.table = TRUE)
}

## End(Not run)

```

Description

Functions to compute Akaike's information criterion (AIC), the second-order AIC (AICc), as well as their quasi-likelihood counterparts (QAIC, QAICc).

Usage

```
AICc(mod, return.K = FALSE, second.ord = TRUE, nobs = NULL, ...)
```

```
## S3 method for class 'aov'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'betareg'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'clm'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'clmm'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'coxme'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'coxph'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'fitdist'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'fitdistr'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```
## S3 method for class 'glm'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, c.hat = 1, ...)
```

```
## S3 method for class 'glmmTMB'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, c.hat = 1, ...)  
  
## S3 method for class 'glms'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'gnls'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'hurdle'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'lavaan'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'lm'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'lme'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'lmekin'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'maxlikeFit'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, c.hat = 1, ...)  
  
## S3 method for class 'mer'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'merMod'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)  
  
## S3 method for class 'lmerModLmerTest'  
AICc(mod, return.K = FALSE, second.ord = TRUE,  
      nobs = NULL, ...)
```

```

## S3 method for class 'multinom'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, c.hat = 1, ...)

## S3 method for class 'negbin'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, ...)

## S3 method for class 'nlme'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, ...)

## S3 method for class 'nls'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, ...)

## S3 method for class 'polr'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, ...)

## S3 method for class 'rlm'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, ...)

## S3 method for class 'survreg'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, ...)

## S3 method for class 'unmarkedFit'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, c.hat = 1, ...)

## S3 method for class 'vglm'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, c.hat = 1, ...)

## S3 method for class 'zeroinfl'
AICc(mod, return.K = FALSE, second.ord = TRUE,
      nobs = NULL, ...)

```

Arguments

mod	an object of class aov, betareg, clm, clmm, clogit, coxme, coxph, fitdist, fitdistr, glm, glmmTMB, gls, gnls, hurdle, lavaan, lm, lme, lmekin, maxlikeFit, mer, merMod, lmerModLmerTest, multinom, negbin, nlme, nls, polr, rlm, survreg, vglm, zeroinfl, and various unmarkedFit classes containing the output of a model.
-----	---

<code>return.K</code>	logical. If FALSE, the function returns the information criterion specified. If TRUE, the function returns <code>K</code> (number of estimated parameters) for a given model.
<code>second.ord</code>	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., <code>nobs</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c.hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If <code>c.hat</code> > 1, AICc will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>). This option is not supported for generalized linear mixed models of the <code>mer</code> or <code>merMod</code> classes.
<code>...</code>	additional arguments passed to the function.

Details

AICc computes one of the following four information criteria:

Akaike's information criterion (AIC, Akaike 1973),

$$-2 * \log - likelihood + 2 * K,$$

where the log-likelihood is the maximum log-likelihood of the model and K corresponds to the number of estimated parameters.

Second-order or small sample AIC (AICc, Sugiura 1978, Hurvich and Tsai 1989, 1991),

$$-2 * \log - likelihood + 2 * K * (n / (n - K - 1)),$$

where n is the sample size of the data set.

Quasi-likelihood AIC (QAIC, Burnham and Anderson 2002),

$$QAIC = \frac{-2 * \log - likelihood}{c - hat} + 2 * K,$$

where *c-hat* is the overdispersion parameter specified by the user with the argument `c.hat`.

Quasi-likelihood AICc (QAICc, Burnham and Anderson 2002),

$$QAICc = \frac{-2 * \log - likelihood}{c - hat} + 2 * K * (n / (n - K - 1))$$

.

Note that AIC and AICc values are meaningful to select among `gls` or `lme` models fit by maximum likelihood. AIC and AICc based on REML are valid to select among different models that only differ in their random effects (Pinheiro and Bates 2000).

Value

AICc returns the AIC, AICc, QAIC, or QAICc, or the number of estimated parameters, depending on the values of the arguments.

Note

The actual (Q)AIC(c) values are not really interesting in themselves, as they depend directly on the data, parameters estimated, and likelihood function. Furthermore, a single value does not tell much about model fit. Information criteria become relevant when compared to one another for a given data set and set of candidate models.

Author(s)

Marc J. Mazerolle

References

- Akaike, H. (1973) Information theory as an extension of the maximum likelihood principle. In: *Second International Symposium on Information Theory*, pp. 267–281. Petrov, B.N., Csaki, F., Eds, Akademiai Kiado, Budapest.
- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Hurvich, C. M., Tsai, C.-L. (1989) Regression and time series model selection in small samples. *Biometrika* **76**, 297–307.
- Hurvich, C. M., Tsai, C.-L. (1991) Bias of the corrected AIC criterion for underfitted regression and time series models. *Biometrika* **78**, 499–509.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Pinheiro, J. C., Bates, D. M. (2000) *Mixed-effect models in S and S-PLUS*. Springer Verlag: New York.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.
- Sugiura, N. (1978) Further analysis of the data by Akaike's information criterion and the finite corrections. *Communications in Statistics: Theory and Methods* **A7**, 13–26.

See Also

[AICcCustom](#), [aictab](#), [confset](#), [importance](#), [evidence](#), [c_hat](#), [modavg](#), [modavgShrink](#), [modavgPred](#), [useBIC](#),

Examples

```
##cement data from Burnham and Anderson (2002, p. 101)
data(cement)
##run multiple regression - the global model in Table 3.2
glob.mod <- lm(y ~ x1 + x2 + x3 + x4, data = cement)

##compute AICc with full likelihood
AICc(glob.mod, return.K = FALSE)

##compute AIC with full likelihood
AICc(glob.mod, return.K = FALSE, second.ord = FALSE)
##note that Burnham and Anderson (2002) did not use full likelihood
##in Table 3.2 and that the MLE estimate of the variance was
##rounded to 2 digits after decimal point

##compute AICc for mixed model on Orthodont data set in Pinheiro and
##Bates (2000)
## Not run:
require(nlme)
m1 <- lme(distance ~ age, random = ~1 | Subject, data = Orthodont,
          method= "ML")
AICc(m1, return.K = FALSE)

## End(Not run)
```

AICcCustom

Compute AIC, AICc, QAIC, and QAICc from User-supplied Input

Description

This function computes Akaike's information criterion (AIC), the second-order AIC (AICc), as well as their quasi-likelihood counterparts (QAIC, QAICc) from user-supplied input instead of extracting the values automatically from a model object. This function is particularly useful for output imported from other software or for model classes that are not currently supported by AICc.

Usage

```
AICcCustom(logL, K, return.K = FALSE, second.ord = TRUE, nobs = NULL,
           c.hat = 1)
```

Arguments

logL	the value of the model log-likelihood.
K	the number of estimated parameters in the model.
return.K	logical. If FALSE, the function returns the information criterion specified. If TRUE, the function returns K (number of estimated parameters) for a given model.
second.ord	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
nobs	the sample size required to compute the AICc or QAICc.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c.hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season or dynamic occupancy models (MacKenzie et al. 2002, 2003), <i>N</i> -mixture models (Royle 2004, Dail and Madson 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If <code>c.hat</code> > 1, AICcCustom will return the quasi-likelihood analogue of the information criterion requested.

Details

AICcCustom computes one of the following four information criteria:

Akaike's information criterion (AIC, Akaike 1973), the second-order or small sample AIC (AICc, Sugiura 1978, Hurvich and Tsai 1989, 1991), the quasi-likelihood AIC (QAIC, Burnham and Anderson 2002), and the quasi-likelihood AICc (QAICc, Burnham and Anderson 2002).

Value

AICcCustom returns the AIC, AICc, QAIC, or QAICc, or the number of estimated parameters, depending on the values of the arguments.

Note

The actual (Q)AIC(c) values are not really interesting in themselves, as they depend directly on the data, parameters estimated, and likelihood function. Furthermore, a single value does not tell much about model fit. Information criteria become relevant when compared to one another for a given data set and set of candidate models.

Author(s)

Marc J. Mazerolle

References

- Akaike, H. (1973) Information theory as an extension of the maximum likelihood principle. In: *Second International Symposium on Information Theory*, pp. 267–281. Petrov, B.N., Csaki, F., Eds, Akademiai Kiado, Budapest.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Hurvich, C. M., Tsai, C.-L. (1989) Regression and time series model selection in small samples. *Biometrika* **76**, 297–307.
- Hurvich, C. M., Tsai, C.-L. (1991) Bias of the corrected AIC criterion for underfitted regression and time series models. *Biometrika* **78**, 499–509.
- Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67–118.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.
- Sugiura, N. (1978) Further analysis of the data by Akaike's information criterion and the finite corrections. *Communications in Statistics: Theory and Methods* **A7**, 13–26.

See Also

[AICc](#), [aictabCustom](#), [confset](#), [evidence](#), [c_hat](#), [modavgCustom](#)

Examples

```
##cement data from Burnham and Anderson (2002, p. 101)
data(cement)
##run multiple regression - the global model in Table 3.2
glob.mod <- lm(y ~ x1 + x2 + x3 + x4, data = cement)

##extract log-likelihood
LL <- logLik(glob.mod)[1]

##extract number of parameters
K.mod <- coef(glob.mod) + 1

##compute AICc with full likelihood
AICcCustom(LL, K.mod, nobs = nrow(cement))
```

AICcmodavg-defunct *Defunct Functions in AICcmodavg Package*

Description

The functions listed below have been removed from the AICcmodavg package.

Usage

```
AICc.mult(...)  
AICc.unmarked(...)  
extract.LL(...)  
extract.LL.coxph(...)  
extract.LL.unmarked(...)  
aictab.clm(...)  
aictab.clmm(...)  
aictab.coxph(...)  
aictab.glm(...)  
aictab.gls(...)  
aictab.lm(...)  
aictab.lme(...)  
aictab.mer(...)  
aictab.merMod(...)  
aictab.mult(...)  
aictab.nlme(...)  
aictab.nls(...)  
aictab.polr(...)  
aictab.rlm(...)  
aictab.unmarked(...)  
dictab.bugs(...)  
dictab.rjags(...)  
modavg.clm(...)  
modavg.clmm(...)  
modavg.coxph(...)  
modavg.glm(...)  
modavg.gls(...)  
modavg.lme(...)  
modavg.mer(...)  
modavg.merMod(...)  
modavg.mult(...)  
modavg.polr(...)  
modavg.rlm(...)  
modavg.unmarked(...)  
modavg.effect(...)  
modavg.effect.glm(...)  
modavg.effect.gls(...)  
modavg.effect.lme(...)  
modavg.effect.mer(...)  
modavg.effect.merMod(...)  
modavg.effect.rlm(...)  
modavg.effect.unmarked(...)  
modavg.shrink(...)  
modavg.shrink.clm(...)  
modavg.shrink.clmm(...)  
modavg.shrink.coxph(...)  
modavg.shrink.glm(...)
```

```

modavg.shrink.gls(...)
modavg.shrink.lme(...)
modavg.shrink.mer(...)
modavg.shrink.merMod(...)
modavg.shrink.mult(...)
modavg.shrink.polr(...)
modavg.shrink.rlm(...)
modavg.shrink.unmarked(...)
modavgpred(...)
modavgpred.glm(...)
modavgpred.gls(...)
modavgpred.lme(...)
modavgpred.mer(...)
modavgpred.merMod(...)
modavgpred.rlm(...)
modavgpred.unmarked(...)
mult.comp(...)
predictSE.zip(...)

```

Arguments

... arguments passed to the function.

Details

AICc.mult has been replaced by AICc.multinom.

AICc.unmarked has been replaced by AICc.unmarkedFit.

extract.LL has been replaced by extractLL.

extract.LL.coxph has been replaced by extractLL.coxph.

extract.LL.unmarked has been replaced by extractLL.unmarkedFit.

aictab.clm has been replaced by aictab.AICsc1m.clm.

aictab.clmm has been replaced by aictab.AICclmm.

aictab.coxph has been replaced by aictab.AICcoxph.

aictab.glm has been replaced by aictab.AICglm.lm.

aictab.gls has been replaced by aictab.AICglsl.

aictab.lm has been replaced by aictab.AIClm.

aictab.lme has been replaced by aictab.AIClme.

aictab.mer has been replaced by aictab.AICmer.

aictab.merMod has been replaced by aictab.AIClmerMod, aictab.AICglmerMod, or aictab.AICnlmerMod, depending on the class of the objects.

aictab.mult has been replaced by aictab.AICmultinom.nnet.

aictab.nlme has been replaced by aictab.AICnlme.

aictab.nls has been replaced by aictab.AICnls.

aictab.polr has been replaced by aictab.AICpolr.

`aictab.rlm` has been replaced by `aictab.AICrlm.lm`.

`aictab.unmarked` has been replaced by `aictab.AICunmarkedFitOccu`, `aictab.AICunmarkedFitColExt`, `aictab.AICunmarkedFitOccuRN`, `aictab.AICunmarkedFitPCount`, `aictab.AICunmarkedFitPCO`, `aictab.AICunmarkedFitDS`, `aictab.AICunmarkedFitGDS`, `aictab.AICunmarkedFitOccuFP`, `aictab.AICunmarkedFitM`, `aictab.AICunmarkedFitGMM`, or `aictab.AICunmarkedFitGPC`, depending on the class of the objects.

`dictab.bugs` has been replaced by `dictab.AICbugs`.

`dictab.jags` has been replaced by `dictab.AICjags`.

`modavg.clm` has been replaced by `modavg.AICsc1m.clm`.

`modavg.c1mm` has been replaced by `modavg.AICsc1m.c1m`.

`modavg.coxph` has been replaced by `modavg.AICcoxph`.

`modavg.glm` has been replaced by `modavg.AIClm` or `modavg.AICglm.lm`, depending on the class of the objects.

`modavg.gls` has been replaced by `modavg.AICglS`.

`modavg.lme` has been replaced by `modavg.AIClme`.

`modavg.mer` has been replaced by `modavg.AICmer`.

`modavg.merMod` has been replaced by `modavg.AIClmerMod` or `modavg.AICglmerMod`, depending on the class of the objects.

`modavg.mult` has been replaced by `modavg.AICmultinom.nnet`.

`modavg.polr` has been replaced by `modavg.AICpolr`.

`modavg.rlm` has been replaced by `modavg.AICrlm.lm`.

`modavg.unmarked` has been replaced by `modavg.AICunmarkedFitOccu`, `modavg.AICunmarkedFitColExt`, `modavg.AICunmarkedFitOccuRN`, `modavg.AICunmarkedFitPCount`, `modavg.AICunmarkedFitPCO`, `modavg.AICunmarkedFitDS`, `modavg.AICunmarkedFitGDS`, `modavg.AICunmarkedFitOccuFP`, `modavg.AICunmarkedFitM`, `modavg.AICunmarkedFitGMM`, or `modavg.AICunmarkedFitGPC`, depending on the class of the objects.

`modavg.effect` has been replaced by `modavgEffect`.

`modavg.effect.glm` has been replaced by `modavgEffect.AICglm.lm` or `modavgEffect.AIClm`, depending on the class of the objects.

`modavg.effect.gls` has been replaced by `modavgEffect.AICglS`.

`modavg.effect.lme` has been replaced by `modavgEffect.AIClme`.

`modavg.effect.mer` has been replaced by `modavgEffect.AICmer`.

`modavg.effect.merMod` has been replaced by `modavgEffect.AICglmerMod` or `modavgEffect.AIClmerMod`, depending on the class of the objects.

`modavg.effect.rlm` has been replaced by `modavgEffect.AICrlm.lm`.

`modavg.effect.unmarked` has been replaced by `modavgEffect.AICunmarkedFitOccu`, `modavgEffect.AICunmarkedFitColExt`, `modavgEffect.AICunmarkedFitOccuRN`, `modavgEffect.AICunmarkedFitPCount`, `modavgEffect.AICunmarkedFitPCO`, `modavgEffect.AICunmarkedFitDS`, `modavgEffect.AICunmarkedFitGDS`, `modavgEffect.AICunmarkedFitOccuFP`, `modavgEffect.AICunmarkedFitMPois`, `modavgEffect.AICunmarkedFitGMM`, or `modavgEffect.AICunmarkedFitGPC`, depending on the class of the objects.

`modavg.shrink` has been replaced by `modavgShrink`.

`modavg.shrink.clm` has been replaced by `modavgShrink.AICsc1m.clm`.

`modavg.shrink.clmm` has been replaced by `modavgShrink.AICclmm`.

`modavg.shrink.coxph` has been replaced by `modavgShrink.AICcoxph`.

`modavg.shrink.glm` has been replaced by `modavgShrink.AICglm.lm` or `modavgShrink.AICglm.lm`, depending on the class of the objects.

`modavg.shrink.gls` has been replaced by `modavgShrink.AICgl`.

`modavg.shrink.lme` has been replaced by `modavgShrink.AIClme`.

`modavg.shrink.mer` has been replaced by `modavgShrink.AICmer`.

`modavg.shrink.merMod` has been replaced by `modavgShrink.AICglmerMod` or `modavgShrink.AIClmerMod`, depending on the class of the objects.

`modavg.shrink.mult` has been replaced by `modavgShrink.AICmultinom.nnet`.

`modavg.shrink.polr` has been replaced by `modavgShrink.AICpolr`.

`modavg.shrink.rlm` has been replaced by `modavgShrink.AICrlm.lm`

`modavg.shrink.unmarked` has been replaced by `modavgShrink.AICunmarkedFitOccu`, `modavgShrink.AICunmarkedFitOccuRN`, `modavgShrink.AICunmarkedFitPCCount`, `modavgShrink.AICunmarkedFitPCO`, `modavgShrink.AICunmarkedFitDS`, `modavgShrink.AICunmarkedFitGDS`, `modavgShrink.AICunmarkedFitOccuFP`, `modavgShrink.AICunmarkedFitMPois`, `modavgShrink.AICunmarkedFitGMM`, or `modavgShrink.AICunmarkedFitGPC`, depending on the class of the objects.

`modavgpred` has been replaced by `modavgPred`.

`modavgpred.glm` has been replaced by `modavgpred.AICglm.lm` or `modavgPred.AIClm`, depending on the class of the objects.

`modavgpred.gls` has been replaced by `modavgPred.AICgl`.

`modavgpred.lme` has been replaced by `modavgPred.AIClme`.

`modavgpred.mer` has been replaced by `modavgPred.AICmer`.

`modavgpred.merMod` has been replaced by `modavgpred.AICglmerMod` or `modavgPred.AIClmerMod`, depending on the class of the objects.

`modavgpred.rlm` has been replaced by `modavgPred.AICrlm.lm`.

`modavgpred.unmarked` has been replaced by `modavgPred.AICunmarkedFitOccu`, `modavgPred.AICunmarkedFitOccuRN`, `modavgPred.AICunmarkedFitPCCount`, `modavgPred.AICunmarkedFitPCO`, `modavgPred.AICunmarkedFitDS`, `modavgPred.AICunmarkedFitGDS`, `modavgPred.AICunmarkedFitOccuFP`, `modavgPred.AICunmarkedFitMPois`, `modavgPred.AICunmarkedFitGMM`, or `modavgPred.AICunmarkedFitGPC`, depending on the class of the objects.

`mult.comp` has been replaced by `multComp`.

`predictSE.zip` has been replaced by `predictSE`.

Author(s)

Marc J. Mazerolle

See Also

[aictab](#), [confset](#), [dictab](#), [importance](#), [evidence](#), [extractLL](#), [c_hat](#), [modavg](#), [modavgEffect](#), [modavgShrink](#), [modavgPred](#), [multComp](#), [predictSE](#)

aictab *Create Model Selection Tables*

Description

This function creates a model selection table based on one of the following information criteria: AIC, AICc, QAIC, QAICc. The table ranks the models based on the selected information criteria and also provides delta AIC and Akaike weights. `aictab` selects the appropriate function to create the model selection table based on the object class. The current version works with lists containing objects of `aov`, `betareg`, `clm`, `clmm`, `clogit`, `coxme`, `coxph`, `fitdist`, `fitdistr`, `glm`, `glmmTMB`, `gls`, `gnls`, `hurdle`, `lavaan`, `lm`, `lme`, `lmekin`, `maxlikeFit`, `mer`, `merMod`, `lmerModLmerTest`, `multinom`, `negbin`, `nlme`, `nls`, `polr`, `rlm`, `survreg`, `vglm`, and `zeroinfl` classes as well as various models of `unmarkedFit` classes but does not yet allow mixing of different classes.

Usage

```
aictab(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
       sort = TRUE, ...)
```

```
## S3 method for class 'AICaov.lm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICbetareg'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICsc1m.clm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICclmm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICclm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICcoxme'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICcoxph'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICfitdist'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICfitdistr'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICglm.lm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICglmmTMB'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICgls'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICgnls.gls'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICHurdle'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClavaan'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClme'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClmekin'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICmaxlikeFit.list'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)
```

```
## S3 method for class 'AICmer'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClmerMod'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClmerModLmerTest'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICglmerMod'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICnlmerMod'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICmultinom.nnet'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICnegbin.glm.lm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICnlme.lme'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICnlsls'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICpolr'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICr1m.lm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICsurvreg'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICunmarkedFitOccu'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitColExt'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitOccuRN'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitPCCount'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitPCO'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitDS'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitGDS'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitOccuFP'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitMPois'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitGMM'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitGPC'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitOccuMulti'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)
```

```

## S3 method for class 'AICunmarkedFitOccuMS'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitOccuTTD'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitMMO'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitDSO'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitGOccu'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitOccuComm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICvglm'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICzeroinfl'
aictab(cand.set, modnames = NULL,
       second.ord = TRUE, nobs = NULL, sort = TRUE, ...)

```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table. If <code>NULL</code> , the function uses the names in the <code>cand.set</code> list of candidate models (i.e., a named list). If no names appear in the list and no character vector is provided, generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as in the list of candidate models.
<code>second.ord</code>	logical. If <code>TRUE</code> , the function returns the second-order Akaike information criterion (i.e., <code>AICc</code>).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the <code>AICc</code> (i.e., <code>nobs</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .

sort	logical. If TRUE, the model selection table is ranked according to the (Q)AIC(c) values.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If <code>c.hat</code> > 1, <code>aictab</code> will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>). This option is not supported for generalized linear mixed models of the <code>mer</code> or <code>merMod</code> classes.
...	additional arguments passed to the function.

Details

`aictab` internally creates a new class for the `cand.set` list of candidate models, according to the contents of the list. The current function is implemented for `clogit`, `coxme`, `coxph`, `fitdist`, `fitdistr`, `glm`, `glmmTMB`, `gls`, `gnls`, `hurdle`, `lavaan`, `lm`, `lme`, `lmekin`, `maxlikeFit`, `mer`, `merMod`, `lmerModLmerTest`, `multinom`, `negbin`, `nlme`, `nls`, `polr`, `rlm`, `survreg`, `vglm`, and `zeroinfl` classes as well as various `unmarkedFit` classes.

The function constructs a model selection table based on one of the four information criteria: AIC, AICc, QAIC, and QAICc.

Ten guidelines for model selection:

- 1) Carefully construct your candidate model set. Each model should represent a specific (interesting) hypothesis to test.
- 2) Keep your candidate model set short. It is ill-advised to consider as many models as there are data.
- 3) Check model fit. Use your global model (most complex model) or subglobal models to determine if the assumptions are valid. If none of your models fit the data well, information criteria will only indicate the most parsimonious of the poor models.
- 4) Avoid data dredging (i.e., looking for patterns after an initial round of analysis).
- 5) Avoid overfitting models. You should not estimate too many parameters for the number of observations available in the sample.
- 6) Be careful of missing values. Remember that values that are missing only for certain variables change the data set and sample size, depending on which variable is included in any given model. I suggest to remove missing cases before starting model selection.
- 7) Use the same response variable for all models of the candidate model set. It is inappropriate to run some models with a transformed response variable and others with the untransformed variable. A workaround is to use a different link function for some models (e.g., identity vs log link).
- 8) When dealing with models with overdispersion, use the same value of `c-hat` for all models in the candidate model set. For binomial models with trials > 1 (i.e., success/trial or `cbind(success, failure)` syntax) or with Poisson GLM's, you should estimate the `c-hat` from the most complex model (global model). If `c-hat` > 1, you should use the same value for each model of the candidate model set (where appropriate) and include it in the count of parameters (*K*). Similarly, for negative

binomial models, you should estimate the dispersion parameter from the global model and use the same value across all models.

9) Burnham and Anderson (2002) recommend to avoid mixing the information-theoretic approach and notions of significance (i.e., P values). It is best to provide estimates and a measure of their precision (standard error, confidence intervals).

10) Determining the ranking of the models is just the first step. Akaike weights sum to 1 for the entire model set and can be interpreted as the weight of evidence in favor of a given model being the best one given the candidate model set considered and the data at hand. Models with large Akaike weights have strong support. Evidence ratios, importance values, and confidence sets for the best model are all measures that assist in interpretation. In cases where the top ranking model has an Akaike weight > 0.9 , one can base inference on this single most parsimonious model. When many models rank highly (i.e., $\Delta(Q)AIC(c) < 4$), one should model-average effect sizes for the parameters with most support across the entire set of models. Model averaging consists in making inference based on the whole set of candidate models, instead of basing conclusions on a single 'best' model. It is an elegant way of making inference based on the information contained in the entire model set.

Value

aictab creates an object of class aictab with the following components:

Modname	the name of each model of the candidate model set.
K	the number of estimated parameters for each model.
(Q)AIC(c)	the information criterion requested for each model (AIC, AICc, QAIC, QAICc).
Delta_(Q)AIC(c)	the appropriate delta AIC component depending on the information criteria selected.
Modellik	the relative likelihood of the model given the data ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
(Q)AIC(c)Wt	the Akaike weights, also termed "model probabilities" sensu Burnham and Anderson (2002) and Anderson (2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
Cum.Wt	the cumulative Akaike weights. These are only meaningful if results in table are sorted in decreasing order of Akaike weights (i.e., <code>sort = TRUE</code>).
c.hat	if c.hat was specified as an argument, it is included in the table.
LL	if c.hat = 1 and parameters estimated by maximum likelihood, the log-likelihood of each model.
Quasi.LL	if c.hat > 1, the quasi log-likelihood of each model.
Res.LL	if parameters are estimated by restricted maximum-likelihood (REML), the restricted log-likelihood of each model.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICc](#), [aictabCustom](#), [bictab](#), [confset](#), [c_hat](#), [evidence](#), [importance](#), [modavg](#), [modavgEffect](#), [modavgShrink](#), [modavgPred](#)

Examples

```
##Mazerolle (2006) frog water loss example
data(dry.frog)

##setup a subset of models of Table 1
Cand.models <- list( )
Cand.models[[1]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2,
  data = dry.frog)
Cand.models[[2]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2 +
  Shade:Substrate, data = dry.frog)
Cand.models[[3]] <- lm(log_Mass_lost ~ cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[4]] <- lm(log_Mass_lost ~ Shade + cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[5]] <- lm(log_Mass_lost ~ Substrate + cent_Initial_mass +
  Initial_mass2, data = dry.frog)

##create a vector of names to trace back models in set
Modnames <- paste("mod", 1:length(Cand.models), sep = " ")
```

```

##generate AICc table
aictab(cand.set = Cand.models, modnames = Modnames, sort = TRUE)
##round to 4 digits after decimal point and give log-likelihood
print(aictab(cand.set = Cand.models, modnames = Modnames, sort = TRUE),
      digits = 4, LL = TRUE)

## Not run:
##Burnham and Anderson (2002) flour beetle data
data(beetle)
##models as suggested by Burnham and Anderson p. 198
Cand.set <- list( )
Cand.set[[1]] <- glm(Mortality_rate ~ Dose, family =
  binomial(link = "logit"), weights = Number_tested,
  data = beetle)
Cand.set[[2]] <- glm(Mortality_rate ~ Dose, family =
  binomial(link = "probit"), weights = Number_tested,
  data = beetle)
Cand.set[[3]] <- glm(Mortality_rate ~ Dose, family =
  binomial(link = "cloglog"), weights = Number_tested,
  data = beetle)

##check c-hat
c_hat(Cand.set[[1]])
c_hat(Cand.set[[2]])
c_hat(Cand.set[[3]])
##lowest value of c-hat < 1 for these non-nested models, thus use
##c.hat = 1

##set up named list
names(Cand.set) <- c("logit", "probit", "cloglog")

##compare models
##model names will be taken from the list if modnames is not specified
res.table <- aictab(cand.set = Cand.set, second.ord = FALSE)
##note that delta AIC and Akaike weights are identical to Table 4.7
print(res.table, digits = 2, LL = TRUE) #print table with 2 digits and
##print log-likelihood in table
print(res.table, digits = 4, LL = FALSE) #print table with 4 digits and
##do not print log-likelihood

## End(Not run)

##two-way ANOVA with interaction
data(iron)
##full model
m1 <- lm(iron ~ Pot + Food + Pot:Food, data = iron)
##additive model
m2 <- lm(iron ~ Pot + Food, data = iron)
##null model
m3 <- lm(iron ~ 1, data = iron)

```

```

##candidate models
Cand.aov <- list(m1, m2, m3)
Cand.names <- c("full", "additive", "null")
aictab(Cand.aov, Cand.names)

##single-season occupancy model example modified from ?occu
## Not run:
require(unmarked)
##single season example modified from ?occu
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
##add fake covariates
siteCovs(pferUMF) <- data.frame(sitevar1 = rnorm(numSites(pferUMF)),
                               sitevar2 = runif(numSites(pferUMF)))

##observation covariates
obsCovs(pferUMF) <- data.frame(obsvar1 = rnorm(numSites(pferUMF) *
                                              obsNum(pferUMF)))

##set up candidate model set
fm1 <- occu(~ obsvar1 ~ sitevar1, pferUMF)
fm2 <- occu(~ 1 ~ sitevar1, pferUMF)
fm3 <- occu(~ obsvar1 ~ sitevar2, pferUMF)
fm4 <- occu(~ 1 ~ sitevar2, pferUMF)

##assemble models in named list (alternative to using 'modnames' argument)
Cand.mods <- list("fm1" = fm1, "fm2" = fm2, "fm3" = fm3, "fm4" = fm4)

##compute table
aictab(cand.set = Cand.mods, second.ord = TRUE)

detach(package:unmarked)

## End(Not run)

```

aictabCustom

Create Model Selection Tables from User-supplied Input Based on (Q)AIC(c)

Description

This function creates a model selection table from model input (log-likelihood, number of estimated parameters) supplied by the user instead of extracting the values automatically from a list of candidate models. The models are ranked based on one of the following information criteria: AIC, AICc, QAIC, QAICc. The table ranks the models based on the selected information criteria and also provides delta AIC and Akaike weights.

Usage

```
aictabCustom(logL, K, modnames = NULL, second.ord = TRUE, nobs = NULL,
             sort = TRUE, c.hat = 1)
```

Arguments

logL	a vector of log-likelihood values for the models in the candidate model set.
K	a vector containing the number of estimated parameters for each model in the candidate model set.
modnames	a character vector of model names to facilitate the identification of each model in the model selection table. If NULL, the function uses the names in the cand.set list of candidate models (i.e., a named list). If no names appear in the list and no character vector is provided, generic names (e.g., Mod1, Mod2) are supplied in the table in the same order as in the list of candidate models.
second.ord	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
nobs	the sample size required to compute the AICc or QAICc.
sort	logical. If TRUE, the model selection table is ranked according to the (Q)AIC(c) values.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from c_hat. Note that values of c.hat different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or cbind(success, failure) syntax), with Poisson GLM's, single-season or dynamic occupancy models (MacKenzie et al. 2002, 2003), <i>N</i> -mixture models (Royle 2004, Dail and Madson 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If c.hat > 1, aictabCustom will return the quasi-likelihood analogue of the information criterion requested.

Details

aictabCustom constructs a model selection table based on one of the four information criteria: AIC, AICc, QAIC, and QAICc. This function is most useful when model input is imported into R from other software (e.g., Program MARK, PRESENCE) or for model classes that are not yet supported by aictab.

Value

aictabCustom creates an object of class aictab with the following components:

Modname	the name of each model of the candidate model set.
K	the number of estimated parameters for each model.
(Q)AIC(c)	the information criteria requested for each model (AICc, AICc, QAIC, QAICc).
Delta_(Q)AIC(c)	the appropriate delta AIC component depending on the information criteria selected.

ModelLik	the relative likelihood of the model given the data ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
(Q)AIC(c)Wt	the Akaike weights, also termed "model probabilities" sensu Burnham and Anderson (2002) and Anderson (2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
Cum.Wt	the cumulative Akaike weights. These are only meaningful if results in table are sorted in decreasing order of Akaike weights (i.e., sort = TRUE).
c.hat	if c.hat was specified as an argument, it is included in the table.
LL	if c.hat = 1 and parameters estimated by maximum likelihood, the log-likelihood of each model.
Quasi.LL	if c.hat > 1, the quasi log-likelihood of each model.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67–118.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICcCustom](#), [bictabCustom](#), [confset](#), [c_hat](#), [evidence](#), [ictab](#), [modavgCustom](#)

Examples

```
##vector with model LL's
LL <- c(-38.8876, -35.1783, -64.8970)

##vector with number of parameters
Ks <- c(7, 9, 4)

##create a vector of names to trace back models in set
Modnames <- c("Cm1", "Cm2", "Cm3")

##generate AICc table
aictabCustom(logL = LL, K = Ks, modnames = Modnames, nobs = 121,
             sort = TRUE)
```

anovaOD

Likelihood-Ratio Test Corrected for Overdispersion

Description

Compute likelihood-ratio test between a given model and a simpler model.

Usage

```
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'glm'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'unmarkedFitOccu'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'unmarkedFitColExt'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'unmarkedFitOccuRN'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'unmarkedFitPCount'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'unmarkedFitPCO'
```

```
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitDS'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitGDS'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitOccuFP'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitMPois'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitGMM'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitGPC'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitOccuMS'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitOccuTTD'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitMMO'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitDSO'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitGOccu'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)
```

```
## S3 method for class 'unmarkedFitOccuComm'
```

```

anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'glmerMod'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'maxlikeFit'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'multinom'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

## S3 method for class 'vglm'
anovaOD(mod.simple, mod.complex, c.hat = 1,
        nobs = NULL, ...)

```

Arguments

<code>mod.simple</code>	an object of class <code>glm</code> , <code>glmmTMB</code> , <code>maxlikeFit</code> , <code>mer</code> , <code>merMod</code> , <code>multinom</code> , <code>vglm</code> , and various <code>unmarkedFit</code> classes containing the output of a model. This model should be a simpler version of <code>mod.complex</code> resulting from a deletion of certain terms (i.e., nested model).
<code>mod.complex</code>	an object of the same class as <code>mod.simple</code> .
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> , <code>mb.gof.test</code> , or <code>Nmix.gof.test</code> . Typically, this value should be computed for the most complex model and applied to simpler models.
<code>nobs</code>	the number of observations used in the analysis. If <code>nobs = NULL</code> , the total number of rows are used as the sample size to compute the residual degrees of freedom as $nobs - K$, where K is the number of estimated parameters. This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .
<code>...</code>	additional arguments passed to the function.

Details

This function applies a correction for overdispersion on the likelihood-ratio test between a model and its simpler counterpart. The simpler model must be nested within the more complex model, typically as the result of deleting terms. You should supply the `c.hat` value of the most complex of the two models you are comparing.

When $1 < c.hat < 4$, the likelihood-ratio test is computed as:

$$LR = \frac{-2 * (LL.simple - LL.complex)}{(K.complex - K.simple) * c.hat}$$

where $LL.simple$ and $LL.complex$ are the log-likelihoods of the simple and complex models, respectively, and where $K.complex$ and $K.simple$ are the number of estimated parameters in each model. The test statistic is approximately distributed as $F_{K.complex-K.simple, n-K.complex}$, where n is the number of observations (i.e., nobs) used in the analysis (Venables and Ripley 2002).

When nobs = NULL, the number of observations is based on the number of rows of the data frame used in the analysis. For mixed models or various models of unmarkedFit, sample size is less straightforward, and nobs could be based on the total number of observations or on the number of independent clusters (e.g., sites), among other choices.

When $c.hat = 1$, the likelihood-ratio test simplifies to:

$$LR = -2 * (LL.simple - LL.complex)$$

where in this case the test statistic is distributed as a $\chi^2_{K.complex-K.simple}$ (McCullagh and Nelder 1989).

The function supports different model types such as Poisson GLM's and GLMM's, single-season and dynamic occupancy models (MacKenzie et al. 2002, 2003), and various N -mixture models (Royle 2004, Dail and Madsen 2011).

Value

anovaOD returns an object of class anovaOD as a list with the following components:

form.simple	a character string of the parameters estimated in mod.simple.
form.complex	a character string of the parameters estimated in mod.complex.
c.hat	the c.hat estimate used to adjust the likelihood-ratio test.
devMat	a matrix storing as columns the number of parameters estimated (K), the log-likelihood of each model logLik, the difference in estimated parameters between the two models (Kdiff), minus twice the difference in log-likelihoods between the models (-2LL), the test statistic, and the associated P-value.

Author(s)

Marc J. Mazerolle

References

- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.

McCullagh, P., Nelder, J. A. (1989) *Generalized Linear Models*. Second edition. Chapman and Hall: New York.

Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

Venables, W. N., Ripley, B. D. (2002) *Modern Applied Statistics with S*. Second edition. Springer-Verlag: New York.

See Also

[c_hat](#), [mb.gof.test](#), [Nmix.gof.test](#), [summaryOD](#)

Examples

```
##anuran larvae example from Mazerolle (2006)
data(min.trap)
##assign "UPLAND" as the reference level as in Mazerolle (2006)
min.trap$type <- relevel(min.trap$type, ref = "UPLAND")

##run model
m1 <- glm(Num_anura ~ Type + log.Perimeter + Num_ranatra,
          family = poisson, offset = log(Effort),
          data = min.trap)
##null model
m0 <- glm(Num_anura ~ 1,
          family = poisson, offset = log(Effort),
          data = min.trap)

##check c-hat for global model
c_hat(m1) #uses Pearson's chi-square/df

##likelihood ratio test corrected for overdispersion
anovaOD(mod.simple = m0, mod.complex = m1, c.hat = c_hat(m1))
##compare without overdispersion correction
anovaOD(mod.simple = m0, mod.complex = m1)

##example with occupancy model
## Not run:
##load unmarked package
if(require(unmarked)){

  data(bullfrog)

  ##detection data
  detections <- bullfrog[, 3:9]

  ##assemble in unmarkedFrameOccu
  bfrog <- unmarkedFrameOccu(y = detections)

  ##run model
  fm <- occu(~ 1 ~ Reed.presence, data = bfrog)
  ##null model
```

```

fm0 <- occu(~ 1 ~ 1, data = bfrog)

##check GOF
##GOF <- mb.gof.test(fm, nsim = 1000)
##estimate of c-hat: 1.89

##display results after overdispersion adjustment
anovaOD(fm0, fm, c.hat = 1.89)

detach(package:unmarked)
}

## End(Not run)

```

beetle

Flour Beetle Data

Description

This data set illustrates the acute mortality of flour beetles (*Tribolium confusum*) following 5 hour exposure to carbon disulfide gas.

Usage

```
data(beetle)
```

Format

A data frame with 8 rows and 4 variables.

Dose dose of carbon disulfide in mg/L.

Number_tested number of beetles exposed to given dose of carbon disulfide.

Number_killed number of beetles dead after 5 hour exposure to given dose of carbon disulfide.

Mortality_rate proportion of total beetles found dead after 5 hour exposure.

Details

Burnham and Anderson (2002, p. 195) use this data set originally from Young and Young (1998) to show model selection for binomial models with different link functions (logit, probit, cloglog).

Source

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

Young, L. J., Young, J. H. (1998) *Statistical Ecology*. Kluwer Academic Publishers: London.

Examples

```

data(beetle)
## maybe str(beetle) ; plot(beetle) ...

```

Description

This function creates a model selection table based on the Bayesian information criterion (Schwarz 1978, Burnham and Anderson 2002). The table ranks the models based on the BIC and also provides delta BIC and BIC model weights. The function adjusts for overdispersion in model selection by using the QBIC when $c.hat > 1$. `bictab` selects the appropriate function to create the model selection table based on the object class. The current version works with lists containing objects of `aov`, `betareg`, `clm`, `clmm`, `clogit`, `coxme`, `coxph`, `fitdist`, `fitdistr`, `glm`, `glmmTMB`, `gls`, `gnls`, `hurdle`, `lavaan`, `lm`, `lme`, `lmekin`, `maxlikeFit`, `mer`, `merMod`, `lmerModLmerTest`, `multinom`, `nlme`, `nls`, `polr`, `rlm`, `survreg`, `vglm`, and `zeroinfl` classes as well as various models of `unmarkedFit` classes but does not yet allow mixing of different classes.

Usage

```
bictab(cand.set, modnames = NULL, nobs = NULL,  
      sort = TRUE, ...)
```

```
## S3 method for class 'AICaov.lm'  
bictab(cand.set, modnames = NULL,  
      nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICbetareg'  
bictab(cand.set, modnames = NULL,  
      nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICsc1m.clm'  
bictab(cand.set, modnames = NULL,  
      nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICclmm'  
bictab(cand.set, modnames = NULL,  
      nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICclm'  
bictab(cand.set, modnames = NULL,  
      nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICcoxme'  
bictab(cand.set, modnames = NULL,  
      nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICcoxph'  
bictab(cand.set, modnames = NULL,  
      nobs = NULL, sort = TRUE, ...)
```

```
## S3 method for class 'AICfitdist'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICfitdistr'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICglm.lm'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICglmmTMB'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICgls'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICgnls.gls'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICHurdle'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClavaan'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClm'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClme'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AIClmekin'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

## S3 method for class 'AICmaxlikeFit.list'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)
```

```
## S3 method for class 'AICmer'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AIClmerMod'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AIClmerModLmerTest'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICglmerMod'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICnlmerMod'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICmultinom.nnet'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICnlme.lme'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICnlsls'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICpolr'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICr1m.lm'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICsurvreg'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, ...)  
  
## S3 method for class 'AICunmarkedFitOccu'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)
```

```
## S3 method for class 'AICunmarkedFitColExt'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitOccuRN'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitPCount'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitPCO'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitDS'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitGDS'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitOccuFP'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitMPois'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitGMM'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitGPC'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitOccuMulti'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)  
  
## S3 method for class 'AICunmarkedFitOccuMS'  
bictab(cand.set, modnames = NULL,  
       nobs = NULL, sort = TRUE, c.hat = 1, ...)
```

```

## S3 method for class 'AICunmarkedFitOccuTTD'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitMMO'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitDSO'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitGOccu'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICunmarkedFitOccuComm'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICvglm'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, c.hat = 1, ...)

## S3 method for class 'AICzeroinfl'
bictab(cand.set, modnames = NULL,
       nobs = NULL, sort = TRUE, ...)

```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table. If <code>NULL</code> , the function uses the names in the <code>cand.set</code> list of candidate models (i.e., a named list). If no names appear in the list and no character vector is provided, generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as in the list of candidate models.
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the BIC (i.e., <code>nobs</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .
<code>sort</code>	logical. If <code>TRUE</code> , the model selection table is ranked according to the BIC values.
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or <i>N</i> -

mixture models (Royle 2004, Dail and Madsen 2011). If $c.hat > 1$, `bictab` will return the quasi-likelihood analogue of the BIC (QBIC) and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by $\sqrt{c.hat}$). This option is not supported for generalized linear mixed models of the `mer` or `merMod` classes.

... additional arguments passed to the function.

Details

BIC tends to favor simpler models than AIC whenever $n > 8$ (Schwarz 1978, Link and Barker 2006, Anderson 2008). BIC assigns uniform prior probabilities across all models (i.e., equal $1/R$), whereas in AIC and AICc, prior probabilities increase with sample size (Burnham and Anderson 2004, Link and Barker 2010). Some authors argue that BIC requires the true model to be included in the model set, whereas AIC or AICc does not (Burnham and Anderson 2002). However, Link and Barker (2006, 2010) consider both as assuming that a model in the model set approximates truth.

`bictab` internally creates a new class for the `cand.set` list of candidate models, according to the contents of the list. The current function is implemented for `clogit`, `coxme`, `coxph`, `fitdist`, `fitdistr`, `glm`, `glmmTMB`, `gls`, `gnls`, `hurdle`, `lavaan`, `lm`, `lme`, `lmekin`, `maxlikeFit`, `mer`, `merMod`, `lmerModLmerTest`, `multinom`, `nlme`, `nls`, `polr`, `rlm`, `survreg`, `vglm`, and `zeroinfl` classes as well as various `unmarkedFit` classes. The function constructs a model selection table based on BIC.

Value

`bictab` creates an object of class `bictab` with the following components:

<code>Modname</code>	the name of each model of the candidate model set.
<code>K</code>	the number of estimated parameters for each model.
<code>(Q)BIC</code>	the Bayesian information criterion for each model.
<code>Delta_(Q)BIC</code>	the delta BIC component.
<code>Modellik</code>	the relative likelihood of the model given the data ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
<code>(Q)BICwt</code>	the BIC model weights, also termed "model probabilities" (Burnham and Anderson 2002, Link and Barker 2006, Anderson 2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
<code>Cum.Wt</code>	the cumulative BIC weights. These are only meaningful if results in table are sorted in decreasing order of BIC weights (i.e., <code>sort = TRUE</code>).
<code>c.hat</code>	if <code>c.hat</code> was specified as an argument, it is included in the table.
<code>LL</code>	the log-likelihood of each model.
<code>Res.LL</code>	if parameters are estimated by restricted maximum-likelihood (REML), the restricted log-likelihood of each model.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Link, W. A., Barker, R. J. (2006) Model weights and the foundations of multimodel inference. *Ecology* **87**, 2626–2635.
- Link, W. A., Barker, R. J. (2010) *Bayesian Inference with Ecological Applications*. Academic Press: Boston.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.
- Schwarz, G. (1978) Estimating the dimension of a model. *Annals of Statistics* **6**, 461–464.

See Also

[aictab](#), [bictabCustom](#), [confset](#), [evidence](#), [importance](#), [useBIC](#),

Examples

```
##Mazerolle (2006) frog water loss example
data(dry.frog)

##setup a subset of models of Table 1
Cand.models <- list( )
Cand.models[[1]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2,
  data = dry.frog)
Cand.models[[2]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2 +
  Shade:Substrate, data = dry.frog)
Cand.models[[3]] <- lm(log_Mass_lost ~ cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[4]] <- lm(log_Mass_lost ~ Shade + cent_Initial_mass +
```

```

        Initial_mass2, data = dry.frog)
Cand.models[[5]] <- lm(log_Mass_lost ~ Substrate + cent_Initial_mass +
        Initial_mass2, data = dry.frog)

##create a vector of names to trace back models in set
Modnames <- paste("mod", 1:length(Cand.models), sep = " ")

##generate BIC table
bictab(cand.set = Cand.models, modnames = Modnames, sort = TRUE)
##round to 4 digits after decimal point and give log-likelihood
print(bictab(cand.set = Cand.models, modnames = Modnames, sort = TRUE),
      digits = 4, LL = TRUE)

## Not run:
##Burnham and Anderson (2002) flour beetle data
data(beetle)
##models as suggested by Burnham and Anderson p. 198
Cand.set <- list( )
Cand.set[[1]] <- glm(Mortality_rate ~ Dose, family =
        binomial(link = "logit"), weights = Number_tested,
        data = beetle)
Cand.set[[2]] <- glm(Mortality_rate ~ Dose, family =
        binomial(link = "probit"), weights = Number_tested,
        data = beetle)
Cand.set[[3]] <- glm(Mortality_rate ~ Dose, family =
        binomial(link = "cloglog"), weights = Number_tested,
        data = beetle)

##set up named list
names(Cand.set) <- c("logit", "probit", "cloglog")

##compare models
##model names will be taken from the list if modnames is not specified
bictab(cand.set = Cand.set)

## End(Not run)

##two-way ANOVA with interaction
data(iron)
##full model
m1 <- lm(iron ~ Pot + Food + Pot:Food, data = iron)
##additive model
m2 <- lm(iron ~ Pot + Food, data = iron)
##null model
m3 <- lm(iron ~ 1, data = iron)

##candidate models
Cand.aov <- list(m1, m2, m3)
Cand.names <- c("full", "additive", "null")
bictab(Cand.aov, Cand.names)

```

```

##single-season occupancy model example modified from ?occu
## Not run:
require(unmarked)
##single season example modified from ?occu
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
##add fake covariates
siteCovs(pferUMF) <- data.frame(sitevar1 = rnorm(numSites(pferUMF)),
                                sitevar2 = runif(numSites(pferUMF)))

##observation covariates
obsCovs(pferUMF) <- data.frame(obsvar1 = rnorm(numSites(pferUMF) *
                                                obsNum(pferUMF)))

##set up candidate model set
fm1 <- occu(~ obsvar1 ~ sitevar1, pferUMF)
fm2 <- occu(~ 1 ~ sitevar1, pferUMF)
fm3 <- occu(~ obsvar1 ~ sitevar2, pferUMF)
fm4 <- occu(~ 1 ~ sitevar2, pferUMF)

##assemble models in named list (alternative to using 'modnames' argument)
Cand.mods <- list("fm1" = fm1, "fm2" = fm2, "fm3" = fm3, "fm4" = fm4)

##compute table based on QBIC that accounts for c.hat
bictab(cand.set = Cand.mods, c.hat = 3.9)

detach(package:unmarked)

## End(Not run)

```

bictabCustom	<i>Create Model Selection Tables from User-supplied Input Based on (Q)BIC</i>
--------------	---

Description

This function creates a model selection table from model input (log-likelihood, number of estimated parameters) supplied by the user instead of extracting the values automatically from a list of candidate models. The models are ranked based on the BIC (Schwarz 1978) or on a quasi-likelihood analogue (QBIC) corrected for overdispersion. The table ranks the models based on the selected information criteria and also provides delta BIC and BIC weights.

Usage

```

bictabCustom(logL, K, modnames = NULL, nobs = NULL, sort = TRUE,
             c.hat = 1)

```

Arguments

logL	a vector of log-likelihood values for the models in the candidate model set.
K	a vector containing the number of estimated parameters for each model in the candidate model set.
modnames	a character vector of model names to facilitate the identification of each model in the model selection table. If NULL, the function uses the names in the cand.set list of candidate models (i.e., a named list). If no names appear in the list and no character vector is provided, generic names (e.g., Mod1, Mod2) are supplied in the table in the same order as in the list of candidate models.
nobs	the sample size required to compute the AICc or QAICc.
sort	logical. If TRUE, the model selection table is ranked according to the (Q)BIC values.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from c_hat. Note that values of c.hat different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or cbind(success, failure) syntax), with Poisson GLM's, single-season or dynamic occupancy models (MacKenzie et al. 2002, 2003), <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If c.hat > 1, bictabCustom will return the quasi-likelihood analogue of the information criterion requested.

Details

bictabCustom constructs a model selection table based on BIC or QBIC. This function is most useful when model input is imported into R from other software (e.g., Program MARK, PRESENCE) or for model classes that are not yet supported by bictab.

Value

bictabCustom creates an object of class bictab with the following components:

Modname	the name of each model of the candidate model set.
K	the number of estimated parameters for each model.
(Q)BIC	the information criteria requested for each model (BIC, QBIC).
Delta_(Q)BIC	the appropriate delta BIC component depending on the information criteria selected.
Modellik	the relative likelihood of the model given the data ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
(Q)BICwt	the BIC weights, also termed "model probabilities" sensu Burnham and Anderson (2002) and Anderson (2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
Cum.Wt	the cumulative BIC weights. These are only meaningful if results in table are sorted in decreasing order of BIC weights (i.e., sort = TRUE).

c.hat	if c.hat was specified as an argument, it is included in the table.
LL	if c.hat = 1 and parameters estimated by maximum likelihood, the log-likelihood of each model.
Quasi.LL	if c.hat > 1, the quasi log-likelihood of each model.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67–118.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.
- Schwarz, G. (1978) Estimating the dimension of a model. *Annals of Statistics* **6**, 461–464.

See Also

[AICcCustom](#), [aictabCustom](#), [confset](#), [c_hat](#), [evidence](#), [ictab](#), [modavgCustom](#)

Examples

```
##vector with model LL's
LL <- c(-38.8876, -35.1783, -64.8970)

##vector with number of parameters
Ks <- c(7, 9, 4)

##create a vector of names to trace back models in set
Modnames <- c("Cm1", "Cm2", "Cm3")
```

```
##generate BIC table
bictabCustom(logL = LL, K = Ks, modnames = Modnames, nobs = 121,
             sort = TRUE)
```

boot.wt

Compute Model Selection Relative Frequencies

Description

This function computes the model selection relative frequencies based on the nonparametric bootstrap (Burnham and Anderson 2002). Models are ranked based on the AIC, AICc, QAIC, or QAICc. The function currently supports objects of `aov`, `betareg`, `clm`, `glm`, `hurdle`, `lm`, `multinom`, `polr`, `r1m`, `survreg`, `vglm`, and `zeroinfl` classes.

Usage

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)
```

```
## S3 method for class 'AICaov.lm'
```

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)
```

```
## S3 method for class 'AICsurvreg'
```

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)
```

```
## S3 method for class 'AICsc1m.clm'
```

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)
```

```
## S3 method for class 'AICglm.lm'
```

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, c.hat = 1, ...)
```

```
## S3 method for class 'AICHurdle'
```

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)
```

```
## S3 method for class 'AIClm'
```

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)
```

```
## S3 method for class 'AICmultinom.nnet'
```

```
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, c.hat = 1, ...)
```

```

## S3 method for class 'AICpolr'
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)

## S3 method for class 'AICr1m.lm'
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)

## S3 method for class 'AICsurvreg'
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)

## S3 method for class 'AICvglm'
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, c.hat = 1, ...)

## S3 method for class 'AICzeroinfl'
boot.wt(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
        sort = TRUE, nsim = 100, ...)

```

Arguments

cand.set	a list storing each of the models in the candidate model set.
modnames	a character vector of model names to facilitate the identification of each model in the model selection table. If NULL, the function uses the names in the cand.set list of candidate models. If no names appear in the list, generic names (e.g., Mod1, Mod2) are supplied in the table in the same order as in the list of candidate models.
second.ord	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
nobs	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., nobs defaults to total number of observations). This is relevant only for certain types of models such as mixed models where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of nobs.
sort	logical. If TRUE, the model selection table is ranked according to the (Q)AIC(c) values.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from c_hat. Note that values of c.hat different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or cbind(success, failure) syntax) or with Poisson GLM's. If c.hat > 1, boot.wt will return the quasi-likelihood analogue of the information criterion requested.
nsim	the number of bootstrap iterations. Burnham and Anderson (2002) recommend at least 1000 and up to 10 000 iterations for certain problems.
...	additional arguments passed to the function.

Details

boot.wt is implemented for aov, betareg, glm, hurdle, lm, multinom, polr, rlm, survreg, vglm, and zeroinfl classes. During each bootstrap iteration, the data are resampled with replacement, all the models specified in cand.set are updated with the new data set, and the top-ranked model is saved. When all iterations are completed, the relative frequency of selection is computed for each model appearing in the candidate model set.

Relative frequencies of the models are often similar to Akaike weights, and the latter are often preferred due to their link with a Bayesian perspective (Burnham and Anderson 2002). boot.wt is most useful for teaching purposes of sampling-theory based relative frequencies of model selection. The current implementation is only appropriate with completely randomized designs. For more complex data structures (e.g., blocks or random effects), the bootstrap should be modified accordingly.

Value

boot.wt creates an object of class boot.wt with the following components:

Modname	the names of each model of the candidate model set.
K	the number of estimated parameters for each model.
(Q)AIC(c)	the information criteria requested for each model (AICc, AICc, QAIC, QAICc).
Delta_(Q)AIC(c)	the appropriate delta AIC component depending on the information criteria selected.
ModelLik	the relative likelihood of the model given the data ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
(Q)AIC(c)Wt	the Akaike weights, also termed "model probabilities" sensu Burnham and Anderson (2002) and Anderson (2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
PiWt	the relative frequencies of model selection from the bootstrap.
c.hat	if c.hat was specified as an argument, it is included in the table.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.

Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.

See Also

[AICc](#), [confset](#), [c_hat](#), [evidence](#), [importance](#), [modavg](#), [modavgShrink](#), [modavgPred](#)

Examples

```
##Mazerolle (2006) frog water loss example
data(dry.frog)

##setup a subset of models of Table 1
Cand.models <- list( )
Cand.models[[1]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2,
  data = dry.frog)
Cand.models[[2]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2 +
  Shade:Substrate, data = dry.frog)
Cand.models[[3]] <- lm(log_Mass_lost ~ cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[4]] <- lm(log_Mass_lost ~ Shade + cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[5]] <- lm(log_Mass_lost ~ Substrate + cent_Initial_mass +
  Initial_mass2, data = dry.frog)

##create a vector of names to trace back models in set
Modnames <- paste("mod", 1:length(Cand.models), sep = " ")

##generate AICc table with bootstrapped relative
##frequencies of model selection
boot.wt(cand.set = Cand.models, modnames = Modnames, sort = TRUE,
  nsim = 10) #number of iterations should be much higher

##Burnham and Anderson (2002) flour beetle data
## Not run:
data(beetle)
##models as suggested by Burnham and Anderson p. 198
Cand.set <- list( )
Cand.set[[1]] <- glm(Mortality_rate ~ Dose, family =
  binomial(link = "logit"), weights = Number_tested,
  data = beetle)
Cand.set[[2]] <- glm(Mortality_rate ~ Dose, family =
  binomial(link = "probit"), weights = Number_tested,
  data = beetle)
Cand.set[[3]] <- glm(Mortality_rate ~ Dose, family =
  binomial(link = "cloglog"), weights = Number_tested,
  data = beetle)

##create a vector of names to trace back models in set
Modnames <- paste("Mod", 1:length(Cand.set), sep = " ")
```

```
##model selection table with bootstrapped
##relative frequencies
boot.wt(cand.set = Cand.set, modnames = Modnames)

## End(Not run)
```

bullfrog

Bullfrog Occupancy and Common Reed Invasion

Description

This is a data set from Mazerolle et al. (2014) on the occupancy of Bullfrogs (*Lithobates catesbeianus*) in 50 wetlands sampled in 2009 in the area of Montreal, QC.

Usage

```
data(bullfrog)
```

Format

A data frame with 50 observations on the following 23 variables.

Location a factor with a unique identifier for each wetland.

Reed.presence a binary variable, either 1 (reed present) or 0 (reed absent).

V1 a binary variable for detection (1) or non detection (0) of bullfrogs during the first survey.

V2 a binary variable for detection (1) or non detection (0) of bullfrogs during the second survey.

V3 a binary variable for detection (1) or non detection (0) of bullfrogs during the third survey.

V4 a binary variable for detection (1) or non detection (0) of bullfrogs during the fourth survey.

V5 a binary variable for detection (1) or non detection (0) of bullfrogs during the fifth survey.

V6 a binary variable for detection (1) or non detection (0) of bullfrogs during the sixth survey.

V7 a binary variable for detection (1) or non detection (0) of bullfrogs during the seventh survey.

Effort1 a numeric variable for the centered number of sampling stations during the first survey.

Effort2 a numeric variable for the centered number of sampling stations during the second survey.

Effort3 a numeric variable for the centered number of sampling stations during the third survey.

Effort4 a numeric variable for the centered number of sampling stations during the fourth survey.

Effort5 a numeric variable for the centered number of sampling stations during the fifth survey.

Effort6 a numeric variable for the centered number of sampling stations during the sixth survey.

Effort7 a numeric variable for the centered number of sampling stations during the seventh survey.

Type1 a binary variable to identify the survey type, either minnow trap (1) or call survey (0) during the first sampling occasion.

Type2 a binary variable to identify the survey type, either minnow trap (1) or call survey (0) during the second sampling occasion.

Type3 a binary variable to identify the survey type, either minnow trap (1) or call survey (0) during the third sampling occasion.

Type4 a binary variable to identify the survey type, either minnow trap (1) or call survey (0) during the fourth sampling occasion.

Type5 a binary variable to identify the survey type, either minnow trap (1) or call survey (0) during the fifth sampling occasion.

Type6 a binary variable to identify the survey type, either minnow trap (1) or call survey (0) during the sixth sampling occasion.

Type7 a binary variable to identify the survey type, either minnow trap (1) or call survey (0) during the seventh sampling occasion.

Details

This data set is used to illustrate single-species single-season occupancy models (MacKenzie et al. 2002) in Mazerolle (2015). The average number of sampling stations on each visit was 8.665714, and was used to center Effort on each visit.

Source

MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002). Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.

Mazerolle, M. J., Perez, A., Brisson, J. (2014) Common reed (*Phragmites australis*) invasion and amphibian distribution in freshwater wetlands. *Wetlands Ecology and Management* **22**, 325–340.

Mazerolle, M. J. (2015) Estimating detectability and biological parameters of interest with the use of the R environment. *Journal of Herpetology* **49**, 541–559.

Examples

```
data(bullfrog)
str(bullfrog)
```

calcium

Blood Calcium Concentration in Birds

Description

This data set features calcium concentration in the plasma of birds of both sexes following a hormonal treatment.

Usage

```
data(calcium)
```

Format

A data frame with 20 rows and 3 variables.

Calcium calcium concentration in mg/100 ml in the blood of birds.

Hormone a factor with two levels indicating whether the bird received a hormonal treatment or not.

Sex a factor with two levels coding for the sex of birds.

Details

Zar (1984, p. 206) illustrates a two-way ANOVA with interaction with this data set.

Source

Zar, J. H. (1984) *Biostatistical analysis*. Second edition. Prentice Hall: Englewood Cliffs, New Jersey.

Examples

```
data(calcium)
str(calcium)
```

cement

Heat Expended Following Hardening of Portland Cement

Description

This data set illustrates the heat expended (calories) from mixtures of four different ingredients of Portland cement expressed as a percentage by weight.

Usage

```
data(cement)
```

Format

A data frame with 13 observations on the following 5 variables.

x1 calcium aluminate.

x2 tricalcium silicate.

x3 tetracalcium alumino ferrite.

x4 dicalcium silicate.

y calories of heat per gram of cement following 180 days of hardening.

Details

Burnham and Anderson (2002, p. 101) use this data set originally from Woods et al. (1932) to select among a set of multiple regression models.

Source

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

Woods, H., Steinour, H. H., Starke, H. R. (1932) Effect of composition of Portland cement on heat evolved during hardening. *Industrial and Engineering Chemistry* **24**, 1207–1214.

Examples

```
data(cement)
## maybe str(cement) ; plot(cement) ...
```

checkConv

Check Convergence of Fitted Model

Description

This function checks the convergence information contained in models of various classes.

Usage

```
checkConv(mod, ...)
```

S3 method for class 'betareg'

```
checkConv(mod, ...)
```

S3 method for class 'clm'

```
checkConv(mod, ...)
```

S3 method for class 'clmm'

```
checkConv(mod, ...)
```

S3 method for class 'glm'

```
checkConv(mod, ...)
```

S3 method for class 'glmmTMB'

```
checkConv(mod, ...)
```

S3 method for class 'hurdle'

```
checkConv(mod, ...)
```

S3 method for class 'lavaan'

```
checkConv(mod, ...)
```

S3 method for class 'maxlikeFit'

```
checkConv(mod, ...)
```

```
## S3 method for class 'merMod'  
checkConv(mod, ...)  
  
## S3 method for class 'lmerModLmerTest'  
checkConv(mod, ...)  
  
## S3 method for class 'multinom'  
checkConv(mod, ...)  
  
## S3 method for class 'nls'  
checkConv(mod, ...)  
  
## S3 method for class 'polr'  
checkConv(mod, ...)  
  
## S3 method for class 'unmarkedFit'  
checkConv(mod, ...)  
  
## S3 method for class 'zeroinfl'  
checkConv(mod, ...)
```

Arguments

mod	an object containing the output of a model of the classes mentioned above.
...	additional arguments passed to the function.

Details

This function checks the element of a model object that contains the convergence information from the optimization function. The function is currently implemented for models of classes `betareg`, `clm`, `clmm`, `glm`, `glmmTMB`, `hurdle`, `lavaan`, `maxlikeFit`, `merMod`, `lmerModLmerTest`, `multinom`, `nls`, `polr`, `unmarkedFit`, and `zeroinfl`. The function is particularly useful for functions with several groups of parameters, such as those of the `unmarked` package (Fiske and Chandler, 2011).

Value

`checkConv` returns a list with the following components:

converged	a logical value indicating whether the algorithm converged or not.
message	a string containing the message from the optimization function.

Author(s)

Marc J. Mazerolle

References

Fiske, I., Chandler, R. (2011) unmarked: An R Package for fitting hierarchical models of wildlife occurrence and abundance. *Journal of Statistical Software* **43**, 1–23.

See Also

[checkParms](#), [covDiag](#), [mb.gof.test](#), [Nmix.gof.test](#)

Examples

```
##example modified from ?pcount
## Not run:
if(require(unmarked)){
##Simulate data
set.seed(3)
nSites <- 100
nVisits <- 3
##covariate
x <- rnorm(nSites)
beta0 <- 0
beta1 <- 1
##expected counts
lambda <- exp(beta0 + beta1*x)
N <- rpois(nSites, lambda)
y <- matrix(NA, nSites, nVisits)
p <- c(0.3, 0.6, 0.8)
for(j in 1:nVisits) {
  y[,j] <- rbinom(nSites, N, p[j])
}
## Organize data
visitMat <- matrix(as.character(1:nVisits),
                  nSites, nVisits, byrow=TRUE)

umf <- unmarkedFramePCount(y=y, siteCovs=data.frame(x=x),
                          obsCovs=list(visit=visitMat))

## Fit model
fm1 <- pcount(~ visit ~ 1, umf, K=50)
checkConv(fm1)
}

## End(Not run)
```

checkParms

Identify Parameters with Large Standard Errors

Description

This function identifies parameter estimates with large standard errors in a model. It is particularly useful for complex models with different parameter types such as those of `unmarkedFit` classes implemented in package `unmarked` (Fiske and Chandler, 2011), as well as other types of regression models.

Usage

```
checkParms(mod, se.max = 25, simplify = TRUE, ...)

## S3 method for class 'betareg'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'clm'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'clmm'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'coxme'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'coxph'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'glm'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'glmmTMB'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'glsl'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'gnls'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'hurdle'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'lm'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'lme'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'lmeKin'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'maxlikeFit'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'merMod'
checkParms(mod, se.max = 25, ...)
```

```

## S3 method for class 'lmerModLmerTest'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'multinom'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'nlme'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'nls'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'polr'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'rlm'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'survreg'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'unmarkedFit'
checkParms(mod, se.max = 25, simplify = TRUE,
...)

## S3 method for class 'vglm'
checkParms(mod, se.max = 25, ...)

## S3 method for class 'zeroinfl'
checkParms(mod, se.max = 25, ...)

```

Arguments

mod	a model of unmarkedFit classes or other regression model. This model is checked to determine the occurrence of large standard errors for parameter estimates.
se.max	specifies the value beyond which standard errors are deemed high for the model at hand. The function will determine the number of estimates with standard errors that exceed se.max.
simplify	this argument is only valid for models of unmarkedFit classes which consist of several parameter types for detection probability and demographic parameters (e.g., abundance, occupancy, extinction). If TRUE, the function returns a matrix with a single row identifying the parameter type and estimate with the highest standard error. If FALSE, the function returns a matrix with as many rows as there are parameter types in the model. In the latter case, the estimate with the highest standard error for each parameter type is presented.
...	additional arguments passed to the function.

Details

In some complex models such as certain hierarchical models (Royle and Dorazio 2008, Kéry and Royle 2015), issues in estimating parameters and their standard errors can occur. Large standard errors can be indicative of problems in estimating certain parameters due to sparse data, parameters on the boundary, or model misspecification. The `checkParms` function computes the number of parameter estimates with standard errors larger than `se.max` and identifies the parameter estimate with the largest standard error across all parameter types (`simplify = TRUE`) or for each parameter type (`simplify = FALSE`).

To help identify large standard errors, users can standardize numeric explanatory variables to zero mean and unit variance. The `checkParms` function can also be useful to identify boundary estimates in classic generalized models or their extensions (Venables and Ripley 2002).

Value

`checkParms` returns a list of class `checkParms` with the following components:

<code>model.class</code>	the class of the model for which diagnostics are requested.
<code>se.max</code>	the value of SE used as a threshold in diagnostics. The function reports the number of parameter estimates with $SE > se.max$.
<code>result</code>	a matrix consisting of three columns, namely, the identity of the parameter estimate with the highest SE (<code>variable</code>), its standard error (<code>max.se</code>), and the number of parameter estimates with SE larger than <code>se.max</code> (<code>n.high.se</code>). For classical regression models with a single response variable, the row name is labeled <code>beta</code> . For <code>unmarkedFit</code> models, the matrix either consists of a single row (<code>simplify = TRUE</code>) labeled with the name of the parameter type (e.g., <code>psi</code> , <code>gam</code> , <code>eps</code> , <code>p</code>) where the highest SE occurs, or consists of as many rows as there are parameter types (<code>simplify = FALSE</code>).

Author(s)

Marc J. Mazerolle

References

- Agresti, A. (2002) *Categorical data analysis*. John Wiley and Sons, Inc.: Hoboken.
- Fiske, I., Chandler, R. (2011) `unmarked`: An R Package for fitting hierarchical models of wildlife occurrence and abundance. *Journal of Statistical Software* **43**, 1–23.
- Kéry, M., Royle, J. A. (2015) *Applied hierarchical modeling in ecology: analysis of distribution, abundance and species richness in R and BUGS*. Academic Press, New York, USA.
- Royle, J. A., Dorazio, R. M. (2008) *Hierarchical modeling and inference in ecology: the analysis of data from populations, metapopulations and communities*. Academic Press: New York.
- Venables, W. N., Ripley, B. D. (2002) *Modern applied statistics with S*, 2nd edition. Springer-Verlag: New York.

See Also

[c_hat](#), [detHist](#), [checkConv](#), [countDist](#), [countHist](#), [extractCN](#), [mb.gof.test](#), [Nmix.gof.test](#), [parboot](#)

Examples

```
##example with multiple-season occupancy model modified from ?colext
## Not run:
require(unmarked)
data(frogs)
umf <- formatMult(masspcru)
obsCovs(umf) <- scale(obsCovs(umf))
siteCovs(umf) <- rnorm(numSites(umf))
yearlySiteCovs(umf) <- data.frame(year = factor(rep(1:7,
                                                numSites(umf))))

##model with with year-dependent transition rates
fm.yearly <- colext(psiformula = ~ 1, gammaformula = ~ year,
                  epsilonformula = ~ year,
                  pformula = ~ JulianDate + I(JulianDate^2),
                  data = umf)

##check for high SE's and report highest
##across all parameter types
checkParms(fm.yearly, simplify = TRUE)

##check for high SE's and report highest
##for each parameter type
checkParms(fm.yearly, simplify = FALSE)
detach(package:unmarked)

## End(Not run)

##example from Agresti 2002 of logistic regression
##with parameters estimated at the boundary (complete separation)
## Not run:
x <- c(10, 20, 30, 40, 60, 70, 80, 90)
y <- c(0, 0, 0, 0, 1, 1, 1, 1)

m1 <- glm(y ~ x, family = binomial)
checkParms(m1)

## End(Not run)
```

confset

Computing Confidence Set for the Kullback-Leibler Best Model

Description

This function computes the confidence set on the best model given the data and model set. `confset` implements three different methods proposed by Burnham and Anderson (2002).

Usage

```
confset(cand.set, modnames = NULL, second.ord = TRUE, nobs = NULL,
       method = "raw", level = 0.95, delta = 6, c.hat = 1)
```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table. If NULL, the function uses the names in the <code>cand.set</code> list of candidate models. If no names appear in the list, generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as in the list of candidate models.
<code>second.ord</code>	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., <code>nobs</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .
<code>method</code>	a character value, either as <code>raw</code> , <code>ordinal</code> , or <code>ratio</code> , indicating the method for determining the confidence set for the best model (see 'Description' above for details).
<code>level</code>	the level of confidence (i.e., sum of model probabilities) used to determine the confidence set on the best model when using the <code>raw</code> method. Note that the argument is not used for the other methods of determining the confidence set on the best model.
<code>delta</code>	the delta (Q)AIC(c) value associated with the cutoff point to determine the confidence set for the best model. Note that the argument is only used when <code>method = ratio</code> .
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c.hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with <code>trials > 1</code> (i.e., <code>success/trial</code> or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If <code>c.hat > 1</code> , <code>confset</code> will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by $\sqrt{c.hat}$). This option is not supported for generalized linear mixed models of the <code>mer</code> or <code>merMod</code> classes.

Details

The first and simplest (`method = 'raw'`), relies on summing the Akaike weights (i.e., model probabilities) of the ranked models until we reach a given cutpoint (e.g., 0.95 for a 95 percent set).

The second method (`method = 'ordinal'`) suggested is based on the classification of the models on an ordinal scale based on the delta (Q)AIC(c). The models are grouped in different classes based on their weight of support as determined by the delta (Q)AIC(c) values: substantial support ($\text{delta (Q)AIC(c)} \leq 2$), some support ($2 < \text{delta (Q)AIC(c)} \leq 7$), little support ($7 < \text{delta (Q)AIC(c)} \leq 10$), no support ($\text{delta (Q)AIC(c)} > 10$).

The third method (method = 'ratio') is based on identifying the ratios of model likelihoods (i.e., $\exp(-\delta_{(Q)AIC(c)}/2)$) that exceed a cutpoint, similar to the building of profile likelihood intervals. An evidence ratio of each model relative to the top-ranked model is computed and the ratios exceeding the cutpoint determine which models are included in the confidence set. Note here that small cutoff points are suggested (e.g., 0.125, 0.050). The cutoff point is linked to $\delta_{(Q)AIC(c)}$ by the following relationship: $cutoff = \exp(-1 * \delta_{(Q)AIC(c)}/2)$.

Value

confset returns an object of class confset as a list with the following components, depending on which method is used:

when method = 'raw':

method	identifies the method of determining the confidence set on the best model.
level	the confidence level used to determine the confidence set on the best model.
table	a reduced table with the models included in the confidence set.

when method = 'ordinal':

method	identifies the method of determining the confidence set on the best model.
substantial	a reduced table with the models included in the confidence set for which $\delta_{(Q)AIC(c)} \leq 2$.
some	a reduced table with the models included in the confidence set for which $2 < \delta_{(Q)AIC(c)} \leq 7$.
little	a reduced table with the models included in the confidence set for which $7 < \delta_{(Q)AIC(c)} \leq 10$.
none	a reduced table with the models included in the confidence set for which $\delta_{(Q)AIC(c)} > 10$.

when method = 'ratio':

method	identifies the method of determining the confidence set on the best model.
cutoff	the cutoff value for the ratios used to determine the confidence set on the best model.
delta	the $\delta_{(Q)AIC(c)}$ used to compute the cutoff value for ratios to determine the confidence set on the best model.
table	a reduced table with the models included in the confidence set.

Author(s)

Marc J. Mazerolle

References

- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.

MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.

MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.

Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICc](#), [aictab](#), [c_hat](#), [evidence](#), [importance](#), [modavg](#), [modavgShrink](#), [modavgPred](#)

Examples

```
##anuran larvae example from Mazerolle (2006)
data(min.trap)
##assign "UPLAND" as the reference level as in Mazerolle (2006)
min.trap$type <- relevel(min.trap$type, ref = "UPLAND")

##set up candidate models
Cand.mod <- list()
##global model
Cand.mod[[1]] <- glm(Num_anura ~ Type + log.Perimeter + Num_ranatra,
  family = poisson, offset = log(Effort),
  data = min.trap)
Cand.mod[[2]] <- glm(Num_anura ~ Type + log.Perimeter, family = poisson,
  offset = log(Effort), data = min.trap)
Cand.mod[[3]] <- glm(Num_anura ~ Type + Num_ranatra, family = poisson,
  offset = log(Effort), data = min.trap)
Cand.mod[[4]] <- glm(Num_anura ~ Type, family = poisson,
  offset = log(Effort), data = min.trap)
Cand.mod[[5]] <- glm(Num_anura ~ log.Perimeter + Num_ranatra,
  family = poisson, offset = log(Effort),
  data = min.trap)
Cand.mod[[6]] <- glm(Num_anura ~ log.Perimeter, family = poisson,
  offset = log(Effort), data = min.trap)
Cand.mod[[7]] <- glm(Num_anura ~ Num_ranatra, family = poisson,
  offset = log(Effort), data = min.trap)
Cand.mod[[8]] <- glm(Num_anura ~ 1, family = poisson,
  offset = log(Effort), data = min.trap)

##check c-hat for global model
c_hat(Cand.mod[[1]]) #uses Pearson's chi-square/df
##note the very low overdispersion: in this case, the analysis could be
##conducted without correcting for c-hat as its value is reasonably close
##to 1

##assign names to each model
Modnames <- c("type + logperim + invertpred", "type + logperim",
  "type + invertpred", "type", "logperim + invertpred",
```

```

"logperim", "invertpred", "intercept only")

##compute confidence set based on 'raw' method
confset(cand.set = Cand.mod, modnames = Modnames, second.ord = TRUE,
        method = "raw")

##example with linear mixed model
## Not run:
require(nlme)

##set up candidate model list for Orthodont data set shown in Pinheiro
##and Bates (2000: Mixed-effect models in S and S-PLUS. Springer Verlag:
##New York.)
Cand.models <- list()
Cand.models[[1]] <- lme(distance ~ age, random = ~age | Subject,
                       data = Orthodont, method = "ML")
Cand.models[[2]] <- lme(distance ~ age + Sex, data = Orthodont,
                       random = ~ 1 | Subject, method = "ML")
Cand.models[[3]] <- lme(distance ~ 1, data = Orthodont,
                       random = ~ 1 | Subject, method = "ML")

##create a vector of model names
Modnames <- paste("mod", 1:length(Cand.models), sep = "")

##compute confidence set based on 'raw' method
confset(cand.set = Cand.models, modnames = Modnames, second.ord = TRUE,
        method = "raw")
##round to 4 digits after decimal point
print(confset(cand.set = Cand.models, modnames = Modnames,
             second.ord = TRUE, method = "raw"), digits = 4)

confset(cand.set = Cand.models, modnames = Modnames, second.ord = TRUE,
        level = 0.9, method = "raw")

##compute confidence set based on 'ordinal' method
confset(cand.set = Cand.models, modnames = Modnames, second.ord = TRUE,
        method = "ordinal")

##compute confidence set based on 'ratio' method
confset(cand.set = Cand.models, modnames = Modnames, second.ord = TRUE,
        method = "ratio", delta = 4)

confset(cand.set = Cand.models, modnames = Modnames, second.ord = TRUE,
        method = "ratio", delta = 8)
detach(package:nlme)

## End(Not run)

```

Description

This function extracts various summary statistics from distance sampling data of various `unmarkedFrame` and `unmarkedFit` classes.

Usage

```
countDist(object, plot.freq = TRUE, plot.distance = TRUE, cex.axis = 1,
          cex.lab = 1, cex.main = 1, ...)
```

```
## S3 method for class 'unmarkedFrameDS'
countDist(object, plot.freq = TRUE,
          plot.distance = TRUE, cex.axis = 1, cex.lab = 1,
          cex.main = 1, ...)
```

```
## S3 method for class 'unmarkedFitDS'
countDist(object, plot.freq = TRUE,
          plot.distance = TRUE, cex.axis = 1, cex.lab = 1,
          cex.main = 1, ...)
```

```
## S3 method for class 'unmarkedFrameGDS'
countDist(object, plot.freq = TRUE,
          plot.distance = TRUE, cex.axis = 1,
          cex.lab = 1, cex.main = 1, ...)
```

```
## S3 method for class 'unmarkedFitGDS'
countDist(object, plot.freq = TRUE,
          plot.distance = TRUE, cex.axis = 1,
          cex.lab = 1, cex.main = 1, ...)
```

```
## S3 method for class 'unmarkedFrameDSO'
countDist(object, plot.freq = TRUE,
          plot.distance = TRUE, cex.axis = 1,
          cex.lab = 1, cex.main = 1, plot.seasons = FALSE,
          ...)
```

```
## S3 method for class 'unmarkedFitDSO'
countDist(object, plot.freq = TRUE,
          plot.distance = TRUE, cex.axis = 1,
          cex.lab = 1, cex.main = 1, plot.seasons = FALSE,
          ...)
```

Arguments

<code>object</code>	an object of various <code>unmarkedFrame</code> or <code>unmarkedFit</code> classes containing distance sampling data.
<code>plot.freq</code>	logical. Specifies if the count data (pooled across seasons and distance classes) should be plotted.

plot.distance	logical. Specifies if the counts in each distance class (pooled across seasons) should be plotted.
cex.axis	expansion factor influencing the size of axis annotations on plots produced by the function.
cex.lab	expansion factor influencing the size of axis labels on plots produced by the function.
cex.main	expansion factor influencing the size of the main title above plots produced by the function.
plot.seasons	logical. Specifies if the count data should be plotted for each distance class and season separately. This argument is only relevant for data collected across more than a single season.
...	additional arguments passed to the function.

Details

This function computes a number of summary statistics in data sets used for the distance sampling models of Royle et al. (2004), Chandler et al. (2011), and distance-sampling versions of models of Dail and Madsen (2011) and Hostetler and Chandler (2015) based on Sollmann et al. (2015).

countDist can take data frames of the unmarkedFrameDS, unmarkedFrameGDS, unmarkedFrameDSO classes as input. For convenience, the function can also extract the raw data from model objects of classes unmarkedFitDS, unmarkedFitGDS, and unmarkedFitDSO. Note that different model objects using the same data set will have identical values.

Value

countDist returns a list with the following components:

count.table.full	a table with the frequency of each observed count pooled across distances classes.
count.table.seasons	a list of tables with the frequency of each season-specific count pooled across distance classes.
dist.sums.full	a table with the frequency of counts in each distance class across the entire sampling seasons.
dist.table.seasons	a list of tables with the frequency of counts in each distance class for each primary period.
dist.names	a character string of labels for the distance classes.
n.dist.classes	the number of distance classes.
out.freqs	a matrix where the rows correspond to each sampling season and where columns consist of the number of sites sampled in season t (sampled) and the number of sites with at least one detection in season t (detected). For multiseason data, the matrix includes the number of sites sampled in season $t - 1$ with colonizations observed in season t (colonized), the number of sites sampled in season $t - 1$ with extinctions observed in season t (extinct), the number of sites sampled in season $t - 1$ without changes observed in season t (static), and the

	number of sites sampled in season t that were also sampled in season $t - 1$ (common).
out.props	a matrix where the rows correspond to each sampling season and where columns consist of the proportion of sites in season t with at least one detection (<code>naive.occ</code>). For multiseason data, the matrix includes the proportion of sites sampled in season $t - 1$ with colonizations observed in season t (<code>naive.colonization</code>), the proportion of sites sampled in season $t - 1$ with extinctions observed in season t (<code>naive.extinction</code>), and the proportion of sites sampled in season $t - 1$ with no changes observed in season t .
n.seasons	the number of seasons (primary periods) in the data set.
n.visits.season	the maximum number of visits per season in the data set.
missing.seasons	logical vector indicating whether data were collected or not during a given season (primary period), where TRUE indicates that no data were collected during the season.

Author(s)

Marc J. Mazerolle

References

- Chandler, R. B., Royle, J. A., King, D. I. (2011) Inference about density and temporary emigration in unmarked populations. *Ecology* **92**, 1429–1435.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Hostetler, J. A., Chandler, R. B. (2015) Improved state-space models for inference about spatial and temporal variation in abundance from count data. *Ecology* **96**, 1713–1723.
- Royle, J. A., Dawson, D. K., Bates, S. (2004) Modeling abundance effects in distance sampling. *Ecology* **85**, 1591–1597.
- Sollmann, R., Gardner, B., Chandler, R. B., Royle, J. A., Sillett, T. S. (2015) An open-population hierarchical distance sampling model. *Ecology* **96**, 325–331.

See Also

[covDiag](#), [detHist](#), [detTime](#), [countHist](#), [Nmix.chisq](#), [Nmix.gof.test](#)

Examples

```
##modified example from ?distsamp
## Not run:
if(require(unmarked)){
data(linetran)
##format data
ltUMF <- with(linetran, {
  unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
    siteCovs = data.frame(Length, area, habitat),
```

```

        dist.breaks = c(0, 5, 10, 15, 20),
        tlength = linetran$Length * 1000, survey = "line",
        unitsIn = "m")
    })

##compute descriptive stats from data object
countDist(ltUMF)

##Half-normal detection function
fm1 <- distsamp(~ 1 ~ 1, ltUMF)
##compute descriptive stats from model object
countDist(fm1)
}

## End(Not run)

```

countHist

Compute Summary Statistics from Count Histories

Description

This function extracts various summary statistics from count data of various unmarkedFrame and unmarkedFit classes.

Usage

```

countHist(object, plot.freq = TRUE, cex.axis = 1, cex.lab = 1,
          cex.main = 1, ...)

## S3 method for class 'unmarkedFramePCount'
countHist(object, plot.freq = TRUE,
          cex.axis = 1, cex.lab = 1, cex.main = 1, ...)

## S3 method for class 'unmarkedFitPCount'
countHist(object, plot.freq = TRUE,
          cex.axis = 1, cex.lab = 1, cex.main = 1, ...)

## S3 method for class 'unmarkedFrameGPC'
countHist(object, plot.freq = TRUE,
          cex.axis = 1, cex.lab = 1, cex.main = 1, ...)

## S3 method for class 'unmarkedFitGPC'
countHist(object, plot.freq = TRUE,
          cex.axis = 1, cex.lab = 1, cex.main = 1, ...)

## S3 method for class 'unmarkedFrameMPois'
countHist(object, plot.freq = TRUE,
          cex.axis = 1, cex.lab = 1, cex.main = 1, ...)

```

```

## S3 method for class 'unmarkedFitMPois'
countHist(object, plot.freq = TRUE,
           cex.axis = 1, cex.lab = 1, cex.main = 1, ...)

## S3 method for class 'unmarkedFramePCO'
countHist(object, plot.freq = TRUE,
           cex.axis = 1, cex.lab = 1, cex.main = 1,
           plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFitPCO'
countHist(object, plot.freq = TRUE,
           cex.axis = 1, cex.lab = 1, cex.main = 1,
           plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFrameGMM'
countHist(object, plot.freq = TRUE,
           cex.axis = 1, cex.lab = 1, cex.main = 1,
           plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFitGMM'
countHist(object, plot.freq = TRUE,
           cex.axis = 1, cex.lab = 1, cex.main = 1,
           plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFrameMMO'
countHist(object, plot.freq = TRUE,
           cex.axis = 1, cex.lab = 1, cex.main = 1,
           plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFitMMO'
countHist(object, plot.freq = TRUE,
           cex.axis = 1, cex.lab = 1, cex.main = 1,
           plot.seasons = FALSE, ...)

```

Arguments

object	an object of various unmarkedFrame or unmarkedFit classes containing count history data.
plot.freq	logical. Specifies if the count data (pooled across seasons) should be plotted.
cex.axis	expansion factor influencing the size of axis annotations on plots produced by the function.
cex.lab	expansion factor influencing the size of axis labels on plots produced by the function.
cex.main	expansion factor influencing the size of the main title above plots produced by the function.

`plot.seasons` logical. Specifies if the count data should be plotted for each season separately. This argument is only relevant for data collected across more than a single season.

`...` additional arguments passed to the function.

Details

This function computes a number of summary statistics in data sets used for various N -mixture models including those of Royle (2004a, b), Dail and Madsen (2011), and Chandler et al. (2011).

`countHist` can take data frames of the `unmarkedFramePCount`, `unmarkedFrameGPC`, `unmarkedFrameMPois`, `unmarkedFramePCO`, `unmarkedFrameGMM`, `unmarkedFrameMMO` classes as input. For convenience, the function can also extract the raw data from model objects of classes `unmarkedFitPCount`, `unmarkedFitGPC`, `unmarkedFitMPois`, `unmarkedFitPCO`, `unmarkedFitGMM`, and `unmarkedFitMMO`. Note that different model objects using the same data set will have identical values.

Value

`countHist` returns a list with the following components:

`count.table.full`
a table with the frequency of each observed count.

`count.table.seasons`
a list of tables with the frequency of each season-specific count.

`hist.table.full`
a table with the frequency of each count history across the entire sampling period.

`hist.table.seasons`
a list of tables with the frequency of each count history for each primary period (season).

`out.freqs`
a matrix where the rows correspond to each sampling season and where columns consist of the number of sites sampled in season t (`sampled`) and the number of sites with at least one detection in season t (`detected`). For multiseason data, the matrix includes the number of sites sampled in season $t - 1$ with colonizations observed in season t (`colonized`), the number of sites sampled in season $t - 1$ with extinctions observed in season t (`extinct`), the number of sites sampled in season $t - 1$ without changes observed in season t (`static`), and the number of sites sampled in season t that were also sampled in season $t - 1$ (`common`).

`out.props`
a matrix where the rows correspond to each sampling season and where columns consist of the proportion of sites in season t with at least one detection (`naive.occ`). For multiseason data, the matrix includes the proportion of sites sampled in season $t - 1$ with colonizations observed in season t (`naive.colonization`), the proportion of sites sampled in season $t - 1$ with extinctions observed in season t (`naive.extinction`), and the proportion of sites sampled in season $t - 1$ with no changes observed in season t .

`n.seasons`
the number of seasons (primary periods) in the data set.

n.visits.season
the maximum number of visits per season in the data set.

missing.seasons
logical vector indicating whether data were collected or not during a given season (primary period), where TRUE indicates that no data were collected during the season.

Author(s)

Marc J. Mazerolle

References

- Chandler, R. B., Royle, J. A., King, D. I. (2011) Inference about density and temporary emigration in unmarked populations. *Ecology* **92**, 1429–1435.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Royle, J. A. (2004a) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.
- Royle, J. A. (2004b) Generalized estimators of avian abundance from count survey data. *Animal Biodiversity and Conservation* **27**, 375–386.

See Also

[covDiag](#), [detHist](#), [detTime](#), [countDist](#), [Nmix.chisq](#), [Nmix.gof.test](#)

Examples

```
##modified example from ?pcount
## Not run:
if(require(unmarked)){
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
                                obsCovs = mallard.obs)

##compute descriptive stats from data object
countHist(mallardUMF)

##run single season model
fm.mallard <- pcount(~ ivel+ date + I(date^2) ~ length + elev +
                    forest, mallardUMF, K=30)
##compute descriptive stats from model object
countHist(fm.mallard)
}

## End(Not run)
```

`covDiag`*Compute Covariance Diagnostic for Lambda in N-mixture Models*

Description

This function extracts the covariance diagnostic of Dennis et al. (2015) for lambda in N -mixture models (Royle 2004) of the `unmarkedFitPCount` class as well as in data frames of the `unmarkedFramePcount` class.

Usage

```
covDiag(object, ...)  
  
## S3 method for class 'unmarkedFitPCount'  
covDiag(object, ...)  
  
## S3 method for class 'unmarkedFramePcount'  
covDiag(object, ...)
```

Arguments

`object` an object of class `unmarkedFitPCount` or `unmarkedFramePcount`.
`...` additional arguments passed to the function.

Details

This function extracts the covariance diagnostic developed by Dennis et al. (2015) for lambda in N -mixture models. Values ≤ 0 suggest sparse data and potential problems during model fitting. `covDiag` can take data frames of the `unmarkedFramePcount` class as input. For convenience, the function also takes the repeated count model object as input, extracts the raw data, and computes the covariance diagnostic. Thus, different models on the same data set will have identical values for this covariance diagnostic.

Value

`covDiag` returns a list with the following components:

`cov.diag` the value of the covariance diagnostic.
`message` a string indicating whether a warning was issued (i.e., "Warning: lambda is infinite, data too sparse") or not (i.e., NULL).

Author(s)

Marc J. Mazerolle

References

- Dennis, E. B., Morgan, B. J. T., Ridout, M. S. (2015) Computational aspects of N -mixture models. *Biometrics* **71**, 237–246.
- Royle, J. A. (2004) N -mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[modavg](#), [modavgPred](#), [Nmix.chisq](#), [Nmix.gof.test](#), [predictSE](#), [pcount](#)

Examples

```
##modified example from ?pcount
## Not run:
if(require(unmarked)){
##Simulate data
set.seed(3)
nSites <- 100
nVisits <- 3
##covariate
x <- rnorm(nSites)
beta0 <- 0
beta1 <- 1
##expected counts
lambda <- exp(beta0 + beta1*x)
N <- rpois(nSites, lambda)
y <- matrix(NA, nSites, nVisits)
p <- c(0.3, 0.6, 0.8)
for(j in 1:nVisits) {
  y[,j] <- rbinom(nSites, N, p[j])
}
## Organize data
visitMat <- matrix(as.character(1:nVisits),
                  nSites, nVisits, byrow=TRUE)

umf <- unmarkedFramePCount(y=y, siteCovs=data.frame(x=x),
                          obsCovs=list(visit=visitMat))

## Fit model
fm1 <- pcount(~ visit ~ 1, umf, K=50)
covDiag(fm1)

##sparser data
p <- c(0.01, 0.001, 0.01)
for(j in 1:nVisits) {
  y[,j] <- rbinom(nSites, N, p[j])
}
## Organize data
visitMat <- matrix(as.character(1:nVisits),
                  nSites, nVisits, byrow=TRUE)

umf <- unmarkedFramePCount(y=y, siteCovs=data.frame(x=x),
```

```

                                obsCovs=list(visit=visitMat))
## Fit model
fm.sparse <- pcount(~ visit ~ 1, umf, K=50)
covDiag(fm.sparse)
}

## End(Not run)

```

c_hat

Estimate Dispersion for Poisson and Binomial GLM's and GLMM's

Description

Functions to compute an estimate of c-hat for binomial or Poisson GLM's and GLMM's using different estimators of overdispersion.

Usage

```

c_hat(mod, method = "pearson", ...)

## S3 method for class 'glm'
c_hat(mod, method = "pearson", ...)

## S3 method for class 'glmmTMB'
c_hat(mod, method = "pearson", ...)

## S3 method for class 'merMod'
c_hat(mod, method = "pearson", ...)

## S3 method for class 'vglm'
c_hat(mod, method = "pearson", ...)

```

Arguments

mod	an object of class glm, glmmTMB, merMod, or vglm for which a c-hat estimate is required.
method	this argument defines the estimator used. The default "pearson" uses the Pearson chi-square divided by the residual degrees of freedom. Other methods include "deviance" consisting of the residual deviance divided by the residual degrees of freedom, "farrington" for the estimator suggested by Farrington (1996), and "fletcher" for the estimator suggested by Fletcher (2012).
...	additional arguments passed to the function.

Details

Poisson and binomial GLM's do not have a parameter for the variance and it is usually held fixed to 1 (i.e., mean = variance). However, one must check whether this assumption is appropriate by estimating the overdispersion parameter (c-hat). Though one can obtain an estimate of c-hat by dividing the residual deviance by the residual degrees of freedom (i.e., method = "deviance"), McCullagh and Nelder (1989) and Venables and Ripley (2002) recommend using Pearson's chi-square divided by the residual degrees of freedom (method = "pearson"). An estimator based on Farrington (1996) is also implemented by the function using the argument method = "farrington". Recent work by Fletcher (2012) suggests that an alternative estimator performs better than the above-mentioned methods in the presence of sparse data and is now implemented with method = "fletcher". For GLMM's, only the Pearson chi-square estimator of overdispersion is currently implemented.

Note that values of c-hat > 1 indicate overdispersion (variance > mean), but that values much higher than 1 (i.e., > 4) probably indicate lack-of-fit. In cases of moderate overdispersion, one usually multiplies the variance-covariance matrix of the estimates by c-hat. As a result, the SE's of the estimates are inflated (c-hat is also known as a variance inflation factor).

In model selection, c-hat should be estimated from the global model of the candidate model set and the same value of c-hat applied to the entire model set. Specifically, a global model is the most complex model which can be simplified to obtain all the other (nested) models of the set. When no single global model exists in the set of models considered, such as when sample size does not allow a complex model, one can estimate c-hat from 'subglobal' models. Here, 'subglobal' models denote models from which only a subset of the models of the candidate set can be derived. In such cases, one can use the smallest value of c-hat for model selection (Burnham and Anderson 2002).

Note that c-hat counts as an additional parameter estimated and should be added to K . All functions in package AICcmodavg automatically add 1 when the c.hat argument > 1 and apply the same value of c-hat for the entire model set. When c.hat > 1, functions compute quasi-likelihood information criteria (either QAICc or QAIC, depending on the value of the second.ord argument) by scaling the log-likelihood of the model by c.hat. The value of c.hat can influence the ranking of the models: as c-hat increases, QAIC or QAICc will favor models with fewer parameters. As an additional check against this potential problem, one can create several model selection tables by incrementing values of c-hat to assess the model selection uncertainty. If ranking changes little up to the c-hat value observed, one can be confident in making inference.

In cases of underdispersion (c-hat < 1), it is recommended to keep the value of c.hat to 1. However, note that values of c-hat \ll 1 can also indicate lack-of-fit and that an alternative model (and distribution) should be investigated.

Note that c_hat only supports the estimation of c-hat for binomial models with trials > 1 (i.e., success/trial or cbind(success, failure) syntax) or with Poisson GLM's or GLMM's.

Value

c_hat returns an object of class c_hat with the estimated c-hat value and an attribute for the type of estimator used.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Farrington, C. P. (1996) On assessing goodness of fit of generalized linear models to sparse data. *Journal of the Royal Statistical Society B* **58**, 349–360.
- Fletcher, D. J. (2012) Estimating overdispersion when fitting a generalized linear model to sparse data. *Biometrika* **99**, 230–237.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.
- McCullagh, P., Nelder, J. A. (1989) *Generalized Linear Models*. Second edition. Chapman and Hall: New York.
- Venables, W. N., Ripley, B. D. (2002) *Modern Applied Statistics with S*. Second edition. Springer: New York.

See Also

[AICc](#), [confset](#), [evidence](#), [modavg](#), [importance](#), [modavgPred](#), [mb.gof.test](#), [Nmix.gof.test](#), [anova0D](#), [summary0D](#)

Examples

```
#binomial glm example
set.seed(seed = 10)
resp <- rbinom(n = 60, size = 1, prob = 0.5)
set.seed(seed = 10)
treat <- as.factor(sample(c(rep(x = "m", times = 30), rep(x = "f",
                    times = 30))))
age <- as.factor(c(rep("young", 20), rep("med", 20), rep("old", 20)))
#each individual has its own response (n = 1)
mod1 <- glm(resp ~ treat + age, family = binomial)
## Not run:
c_hat(mod1) #gives an error because model not appropriate for
##computation of c-hat

## End(Not run)

##computing table to summarize successes
table(resp, treat, age)
dat2 <- as.data.frame(table(resp, treat, age)) #not quite what we need
data2 <- data.frame(success = c(9, 4, 2, 3, 5, 2),
                    sex = c("f", "m", "f", "m", "f", "m"),
                    age = c("med", "med", "old", "old", "young",
                            "young"), total = c(13, 7, 10, 10, 7, 13))
data2$prop <- data2$success/data2$total
```

```

data2$fail <- data2$total - data2$success

##run model with success/total syntax using weights argument
mod2 <- glm(prop ~ sex + age, family = binomial, weights = total,
            data = data2)
c_hat(mod2)

##run model with other syntax cbind(success, fail)
mod3 <- glm(cbind(success, fail) ~ sex + age, family = binomial,
            data = data2)
c_hat(mod3)

```

detHist

Compute Summary Statistics from Detection Histories

Description

This function extracts various summary statistics from detection history data of various `unmarkedFrame` and `unmarkedFit` classes.

Usage

```

detHist(object, ...)

## S3 method for class 'unmarkedFitColExt'
detHist(object, ...)

## S3 method for class 'unmarkedFitOccu'
detHist(object, ...)

## S3 method for class 'unmarkedFitOccuFP'
detHist(object, ...)

## S3 method for class 'unmarkedFitOccuRN'
detHist(object, ...)

## S3 method for class 'unmarkedFitOccuMulti'
detHist(object, ...)

## S3 method for class 'unmarkedFitOccuMS'
detHist(object, ...)

## S3 method for class 'unmarkedFrameOccu'
detHist(object, ...)

## S3 method for class 'unmarkedFrameOccuFP'
detHist(object, ...)

```

```

## S3 method for class 'unmarkedMultFrame'
detHist(object, ...)

## S3 method for class 'unmarkedFrameOccuMulti'
detHist(object, ...)

## S3 method for class 'unmarkedFrameOccuMS'
detHist(object, ...)

## S3 method for class 'unmarkedFitGOccu'
detHist(object, ...)

## S3 method for class 'unmarkedFrameOccuComm'
detHist(object, ...)

## S3 method for class 'unmarkedFitOccuComm'
detHist(object, ...)

```

Arguments

object	an object of various unmarkedFrame or unmarkedFit classes containing detection history data.
...	additional arguments passed to the function.

Details

This function computes a number of summary statistics in data sets used for single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), Royle-Nichols models (Royle and Nichols 2003), false-positive occupancy models (Royle and Link 2006, Miller et al. 2011), multispecies occupancy models (Rota et al. 2016), and multistate occupancy models (Nichols et al. 2007, MacKenzie et al. 2009).

detHist can take data frames of the unmarkedFrameOccu, unmarkedFrameOccuFP, unmarkedMultFrame, unmarkedFrameOccuMulti, unmarkedFrameOccuMS classes as input. For convenience, the function can also extract the raw data from model objects of classes unmarkedFitColExt, unmarkedFitOccu, unmarkedFitOccuFP, unmarkedFitOccuRN, unmarkedFitOccuMulti, unmarkedFitOccuMS, unmarkedFitGOccu, and unmarkedFitOccuComm. Note that different model objects using the same data set will have identical values.

Value

For objects of classes unmarkedFitOccu, unmarkedFitOccuRN, unmarkedFitOccuFP, unmarkedFitColExt, unmarkedFitOccuMS, unmarkedFitGOccu, unmarkedFitOccuComm, unmarkedFrameOccu, unmarkedFrameOccuFP, unmarkedMultFrame, unmarkedFrameOccuMS, unmarkedFrameOccuComm, detHist returns a list with the following components:

hist.table.full	a table with the frequency of each observed detection history. For unmarkedFitGOccu objects, this element consists in a summary of the detections pooled across de-
-----------------	---

vices (secondary periods) within each primary period. This element is NULL for unmarkedFrameOccuComm and unmarkedFitOccuComm objects.

<code>hist.table.seasons</code>	a list of tables with the frequency of each season-specific detection history.
<code>out.freqs</code>	a matrix where the rows correspond to each sampling season and where columns consist of the number of sites sampled in season t (<code>sampled</code>) and the number of sites with at least one detection in season t (<code>detected</code>). For multiseason data, the matrix includes the number of sites sampled in season $t - 1$ with colonizations observed in season t (<code>colonized</code>), the number of sites sampled in season $t - 1$ with extinctions observed in season t (<code>extinct</code>), the number of sites sampled in season $t - 1$ without changes observed in season t (<code>static</code>), and the number of sites sampled in season t that were also sampled in season $t - 1$ (<code>common</code>). For multispecies data, <code>out.freqs</code> presents, for each species, the number of sites sampled and the number of sites with at least one detection.
<code>out.props</code>	a matrix where the rows correspond to each sampling season and where columns consist of the proportion of sites in season t with at least one detection (<code>naive.occ</code>). For multiseason data, the matrix includes the proportion of sites sampled in season $t - 1$ with colonizations observed in season t (<code>naive.colonization</code>), the proportion of sites sampled in season $t - 1$ with extinctions observed in season t (<code>naive.extinction</code>), and the proportion of sites sampled in season $t - 1$ with no changes observed in season t . For multispecies data, <code>out.props</code> presents the proportion of sites with a least one detection for each species.
<code>n.seasons</code>	the number of seasons (primary periods) in the data set.
<code>n.visits.season</code>	the maximum number of visits per season in the data set.
<code>n.species</code>	the number of species in the data set.
<code>missing.seasons</code>	logical vector indicating whether data were collected or not during a given season (primary period), where TRUE indicates that no data were collected during the season.

For objects of classes `unmarkedFitOccuMulti` and `unmarkedFrameOccuMulti`, `detHist` returns a list with the following components:

<code>hist.table.full</code>	a table with the frequency of each observed detection history. The species are coded with letters and follow the same order of presentation as in the other parts of the output.
<code>hist.table.species</code>	a list of tables with the frequency of each species-specific detection history. The last element of <code>hist.table.species</code> features the number of sites with co-occurrence of the different species (<code>coOcc</code>).
<code>out.freqs</code>	a matrix where the rows correspond to each species and where columns consist of the number of sites sampled during the season (<code>sampled</code>) and the number of sites with at least one detection (<code>detected</code>).

out.props	a matrix where the rows correspond to each species and where columns consist of the proportion of sites with at least one detection during the season (naive.occ).
n.seasons	the number of seasons (primary periods) in the data set.
n.visits.season	the maximum number of visits per season in the data set.
n.species	the number of species in the data set.
missing.seasons	logical vector indicating whether data were collected or not during a given season (primary period), where TRUE indicates that no data were collected during the season.

Author(s)

Marc J. Mazerolle

References

- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- MacKenzie, D. I., Nichols, J. D., Seamans, M. E., Gutierrez, R. J. (2009) Modeling species occurrence dynamics with multiple states and imperfect detection. *Ecology* **90**, 823–835.
- Mazerolle, M. J. (2015) Estimating detectability and biological parameters of interest with the use of the R environment. *Journal of Herpetology* **49**, 541–559.
- Miller, D. A. W., Nichols, J. D., McClintock, B. T., Campbell Grant, E. H., Bailey, L. L. (2011) Improving occupancy estimation when two types of observational error occur: non-detection and species misidentification. *Ecology* **92**, 1422–1428.
- Nichols, J. D., Hines, J. E., Mackenzie, D. I., Seamans, M. E., Gutierrez, R. J. (2007) Occupancy estimation and modeling with multiple states and state uncertainty. *Ecology* **88**, 1395–1400.
- Rota, C. T., Ferreira, M. A. R., Kays, R. W., Forrester, T. D., Kalies, E. L., McShea, W. J., Parsons, A. W., Millsbaugh, J. J. (2016) A multispecies occupancy model for two or more interacting species. *Methods in Ecology and Evolution* **7**, 1164–1173.
- Royle, J. A., Link, W. A. (2006) Generalized site occupancy models allowing for false positive and false negative errors. *Ecology* **87**, 835–841.
- Royle, J. A., Nichols, J. D. (2003) Estimating abundance from repeated presence-absence data or point counts. *Ecology* **84**, 777–790.

See Also

[covDiag](#), [countHist](#), [countDist](#), [detTime](#), [mb.chisq](#), [mb.gof.test](#)

Examples

```
##data from Mazerolle (2015)
## Not run:
data(bullfrog)

##detection data
detections <- bullfrog[, 3:9]

##load unmarked package
if(require(unmarked)){

##assemble in unmarkedFrameOccu
bfrog <- unmarkedFrameOccu(y = detections)

##compute descriptive stats from data object
detHist(bfrog)

##run model
fm <- occu(~ 1 ~ 1, data = bfrog)
##compute descriptive stats from model object
detHist(fm)
}

## End(Not run)
```

detTime

Compute Summary Statistics from Time to Detection Data

Description

This function extracts various summary statistics from time to detection data of various unmarkedFrame and unmarkedFit classes.

Usage

```
detTime(object, plot.time = TRUE, plot.seasons = FALSE,
        cex.axis = 1, cex.lab = 1, cex.main = 1, ...)

## S3 method for class 'unmarkedFrameOccuTTD'
detTime(object, plot.time = TRUE,
        plot.seasons = FALSE, cex.axis = 1, cex.lab = 1,
        cex.main = 1, ...)

## S3 method for class 'unmarkedFitOccuTTD'
detTime(object, plot.time = TRUE,
        plot.seasons = FALSE, cex.axis = 1, cex.lab = 1,
        cex.main = 1, ...)
```

Arguments

<code>object</code>	an object of various <code>unmarkedFrame</code> or <code>unmarkedFit</code> classes containing time to detection data.
<code>plot.time</code>	logical. Specifies if the time to detection data (pooled across seasons) should be plotted.
<code>plot.seasons</code>	logical. Specifies if the time to detection data should be plotted for each season separately. This argument is only relevant for data collected across more than a single season.
<code>cex.axis</code>	expansion factor influencing the size of axis annotations on plots produced by the function.
<code>cex.lab</code>	expansion factor influencing the size of axis labels on plots produced by the function.
<code>cex.main</code>	expansion factor influencing the size of the main title above plots produced by the function.
<code>...</code>	additional arguments passed to the function.

Details

This function computes a number of summary statistics in data sets used for the time to detection models of Garrard et al. (2008, 2013).

`detTime` can take data frames of the `unmarkedFrameOccuTTD` class as input, or can also extract the raw data from model objects of the `unmarkedFitOccuTTD` class. Note that different model objects using the same data set will have identical values.

Value

`detTime` returns a list with the following components:

<code>time.table.full</code>	a table with the quantiles of time to detection data pooled across seasons, but excluding censored observations.
<code>time.table.seasons</code>	a list of tables with the quantiles of season-specific time to detection data, but excluding censored observations.
<code>out.freqs</code>	a matrix where the rows correspond to each sampling season and where columns consist of the number of sites sampled in season t (<code>sampled</code>) and the number of sites with at least one detection in season t (<code>detected</code>). For multiseason data, the matrix includes the number of sites sampled in season $t - 1$ with colonizations observed in season t (<code>colonized</code>), the number of sites sampled in season $t - 1$ with extinctions observed in season t (<code>extinct</code>), the number of sites sampled in season $t - 1$ without changes observed in season t (<code>static</code>), and the number of sites sampled in season t that were also sampled in season $t - 1$ (<code>common</code>).
<code>out.props</code>	a matrix where the rows correspond to each sampling season and where columns consist of the proportion of sites in season t with at least one detection (<code>naive.occ</code>).

For multiseason data, the matrix includes the proportion of sites sampled in season $t - 1$ with colonizations observed in season t (`naive.colonization`), the proportion of sites sampled in season $t - 1$ with extinctions observed in season t (`naive.extinction`), and the proportion of sites sampled in season $t - 1$ with no changes observed in season t .

`n.seasons` the number of seasons (primary periods) in the data set.

`n.visits.season` the maximum number of visits per season in the data set.

`missing.seasons` logical vector indicating whether data were collected or not during a given season (primary period), where TRUE indicates that no data were collected during the season.

Author(s)

Marc J. Mazerolle

References

Garrard, G. E., Bekessy, S. A., McCarthy, M. A., Wintle, B. A. (2008) When have we looked hard enough? A novel method for setting minimum survey effort protocols for flora surveys. *Austral Ecology* **33**, 986–998.

Garrard, G. E., McCarthy, M. A., Williams, N. S., Bekessy, S. A., Wintle, B. A. (2013) A general model of detectability using species traits. *Methods in Ecology and Evolution* **4**, 45–52.

See Also

[countDist](#), [countHist](#), [detHist](#), [Nmix.chisq](#), [Nmix.gof.test](#)

Examples

```
##example from ?occuTTD
## Not run:
if(require(unmarked)){
N <- 500; J <- 1

##Simulate occupancy
scovs <- data.frame(elev=c(scale(runif(N, 0,100))),
                    forest=runif(N,0,1),
                    wind=runif(N,0,1))

beta_psi <- c(-0.69, 0.71, -0.5)
psi <- plogis(cbind(1, scovs$elev, scovs$forest))
z <- rbinom(N, 1, psi)

##Simulate detection
Tmax <- 10 #Same survey length for all observations
beta_lam <- c(-2, -0.2, 0.7)
rate <- exp(cbind(1, scovs$elev, scovs$wind))
ttd <- rexp(N, rate)
```

```

ttd[z==0] <- Tmax #Censor unoccupied sites
ttd[ttd>Tmax] <- Tmax #Censor when ttd was greater than survey length

##Build unmarkedFrame
umf <- unmarkedFrameOccuTTD(y=ttd, surveyLength=Tmax, siteCovs=scovs)

##compute descriptive stats from data object
detTime(umf)

##Fit model
fit.occuTTD <- occuTTD(psiformula=~elev+forest, detformula=~elev+wind, data=umf)
##extract info from model object
detTime(fit.occuTTD)

}

## End(Not run)

```

DIC

Computing DIC

Description

Functions to extract deviance information criterion (DIC).

Usage

```

DIC(mod, return.pD = FALSE, ...)

## S3 method for class 'bugs'
DIC(mod, return.pD = FALSE, ...)

## S3 method for class 'rjags'
DIC(mod, return.pD = FALSE, ...)

## S3 method for class 'jagsUI'
DIC(mod, return.pD = FALSE, ...)

```

Arguments

<code>mod</code>	an object of class <code>bugs</code> , <code>rjags</code> , or <code>jagsUI</code> containing the output of a model.
<code>return.pD</code>	logical. If <code>FALSE</code> , the function returns the DIC. If <code>TRUE</code> , the function returns the effective number of estimated parameters (<code>pD</code>) for a given model.
<code>...</code>	additional arguments passed to the function.

Details

DIC is implemented for `bugs`, `rjags`, and `jagsUI` classes. The function extracts the deviance information criterion (DIC, Spiegelhalter et al. 2002) or the effective number of parameters (`pD`).

Value

DIC the DIC or pD depending on the values of the arguments.

Note

The actual DIC values are not really interesting in themselves, as they depend directly on the data, parameters estimated, and likelihood function. Furthermore, a single value does not tell much about model fit. Information criteria become relevant when compared to Yone another for a given data set and set of candidate models. Model selection with hierarchical models is problematic as the classic DIC is not appropriate for such types of models (Millar 2009).

Author(s)

Marc J. Mazerolle

References

Millar, R. B. (2009) Comparison of hierarchical Bayesian models for overdispersed count data using DIC and Bayes' factors. *Biometrics*, **65**, 962–969.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., van der Linde, A. (2002). Bayesian measures of complexity and fit. *Journal of the Royal Statistical Society, Series B* **64**, 583–639.

See Also

[AICcCustom](#), [aictab](#), [dictab](#), [confset](#), [evidence](#)

Examples

```
##from ?jags example in R2jags package
## Not run:
require(R2jags)
##example model file
model.file <- system.file(package="R2jags", "model", "schools.txt")
file.show(model.file)

##data
J <- 8.0
y <- c(28.4,7.9,-2.8,6.8,-0.6,0.6,18.0,12.2)
sd <- c(14.9,10.2,16.3,11.0,9.4,11.4,10.4,17.6)

##arrange data in list
jags.data <- list (J = J, y = y, sd = sd)

##initial values
jags.inits <- function(){
  list(theta=rnorm(J, 0, 100), mu=rnorm(1, 0, 100),
        sigma=runif(1, 0, 100))
}

##parameters to be monitored
jags.parameters <- c("theta", "mu", "sigma")
```

```
##run model
schools.sim <- jags(data = jags.data, inits = jags.inits,
                   parameters = jags.parameters,
                   model.file = model.file,
                   n.chains = 3, n.iter = 10)
##note that n.iter should be higher

##extract DIC
DIC(schools.sim)
##extract pD
DIC(schools.sim, return.pD = TRUE)
detach(package:R2jags)

## End(Not run)
```

dictab

Create Model Selection Tables from Bayesian Analyses

Description

This function creates a model selection table based on the deviance information criterion (DIC). The table ranks the models based on the DIC and also provides delta DIC and DIC weights. `dictab` selects the appropriate function to create the model selection table based on the object class. The current version works with objects of `bugs`, `rjags`, `jagsUI` classes.

Usage

```
dictab(cand.set, modnames = NULL, sort = TRUE, ...)

## S3 method for class 'AICbugs'
dictab(cand.set, modnames = NULL, sort = TRUE, ...)

## S3 method for class 'AICrjags'
dictab(cand.set, modnames = NULL, sort = TRUE, ...)

## S3 method for class 'AICjagsUI'
dictab(cand.set, modnames = NULL, sort = TRUE, ...)
```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table. If <code>NULL</code> , the function uses the names in the <code>cand.set</code> list of candidate models. If no names appear in the list, generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as in the list of candidate models.
<code>sort</code>	logical. If <code>TRUE</code> , the model selection table is ranked according to the DIC values.
<code>...</code>	additional arguments passed to the function.

Details

dictab internally creates a new class for the cand. set list of candidate models, according to the contents of the list. The current function is implemented for bugs, jags, jagsUI classes. The function constructs a model selection table based on the DIC (Spiegelhalter et al. 2002). Note that DIC might not be appropriate to select among a set of hierarchical models and that modifications to the information criterion have been proposed (Millar 2009).

Value

dictab creates an object of class dictab with the following components:

Modname	the name of each model of the candidate model set.
pD	the effective number of estimated parameters for each model.
DIC	the deviance information criterion for each model.
Delta_DIC	the delta DIC of each model, measuring the difference in DIC between each model and the top-ranked model.
Modellik	the relative likelihood of the model given the data ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
DICwt	the DIC weights, sensu Burnham and Anderson (2002) and Anderson (2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
Cum.Wt	the cumulative DIC weights. These are only meaningful if results in table are sorted in decreasing order of DIC weights (i.e., sort = TRUE).
Deviance	the deviance of each model.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Spiegelhalter, D. J., Best, N. G., Carlin, B. P., van der Linde, A. (2002). Bayesian measures of complexity and fit. *Journal of the Royal Statistical Society, Series B* **64**, 583–639.

See Also

[aictabCustom](#), [aictab](#), [confset](#), [DIC](#), [evidence](#)

Examples

```

##from ?jags example in R2jags package
## Not run:
require(R2jags)
model.file <- system.file(package="R2jags", "model", "schools.txt")
file.show(model.file)

##data
J <- 8.0
y <- c(28.4,7.9,-2.8,6.8,-0.6,0.6,18.0,12.2)
sd <- c(14.9,10.2,16.3,11.0,9.4,11.4,10.4,17.6)

jags.data <- list (J = J, y = y, sd = sd)
jags.inits <- function(){
  list(theta=rnorm(J, 0, 100), mu=rnorm(1, 0, 100),
        sigma=runif(1, 0, 100))
}
jags.parameters <- c("theta", "mu", "sigma")

##run model
schools.sim <- jags(data = jags.data, inits = jags.inits,
                    parameters = jags.parameters,
                    model.file = model.file,
                    n.chains = 3, n.iter = 10)
#note that n.iter should be higher

##set up in list
Cand.mods <- list(schools.sim)
Model.names <- "hierarchical model"
##other models can be added to Cand.mods
##to compare them to the top model

##model selection table
dictab(cand.set = Cand.mods, modnames = Model.names)
detach(package:R2jags)

## End(Not run)

```

dry.frog

Frog Dehydration Experiment on Three Substrate Types

Description

This is a data set modified from Mazerolle and Desrochers (2005) on the mass lost by frogs after spending two hours on one of three substrates that are encountered in some landscape types.

Usage

```
data(dry.frog)
```

Format

A data frame with 121 observations on the following 16 variables.

`Individual` a numeric identifier unique to each individual.

`Species` a factor with levels Racla.

`Shade` a numeric vector, either 1 (shade) or 0 (no shade).

`SVL` the snout-vent length of the individual.

`Substrate` the substrate type, a factor with levels PEAT, SOIL, and SPHAGNUM.

`Initial_mass` the initial mass of individuals.

`Mass_lost` the mass lost in g.

`Airtemp` the air temperature in degrees C.

`Wind_cat` the wind intensity, either 0 (no wind), 1 (low wind), 2 (moderate wind), or 3 (strong wind).

`Cloud` cloud cover expressed as a percentage.

`cent_Initial_mass` centered initial mass.

`Initial_mass2` initial mass squared.

`cent_Air` centered air temperature.

`Perc.cloud` proportion of cloud cover

`Wind` wind intensity, either 1 (no or low wind) or 1 (moderate to strong wind).

`log_Mass_lost` log of mass lost.

Details

Note that the original analysis in Mazerolle and Desrochers (2005) consisted of generalized estimating equations for three mass measurements: mass at time 0, 1 hour, and 2 hours following exposure on the substrate.

Source

Mazerolle, M. J., Desrochers, A. (2005) Landscape resistance to frog movements. *Canadian Journal of Zoology* **83**, 455–464.

Examples

```
data(dry.frog)
## maybe str(dry.frog) ; plot(dry.frog) ...
```

evidence	<i>Compute Evidence Ratio Between Two Models</i>
----------	--

Description

This function compares two models of a candidate model set based on their evidence ratio (i.e., ratio of model weights). The default computes the evidence ratio of the model weights between the top-ranked model and the second-ranked model. You must supply a model selection table of class `aictab`, `bictab`, `boot.wt`, `dictab`, `ictab` as the first argument.

Usage

```
evidence(aic.table, model.high = "top", model.low = "second.ranked")
```

Arguments

<code>aic.table</code>	a model selection table of class <code>aictab</code> such as that produced by <code>aictab</code> or of classes <code>bictab</code> , <code>boot.wt</code> , <code>dictab</code> , or <code>ictab</code> . The table may be sorted or not, as the function sorts the table internally.
<code>model.high</code>	the top-ranked model (default), or alternatively, the name of another model as it appears in the model selection table.
<code>model.low</code>	the second-ranked model (default), or alternatively, the name of a lower-ranked model such as it appears in the model selection table.

Details

The default compares the model weights of the top-ranked model to the second-ranked model in the candidate model set. The evidence ratio can be interpreted as the number of times a given model is more parsimonious than a lower-ranked model. If one desires an evidence ratio that does not involve a comparison with the top-ranking model, the label of the required model must be specified in the `model.high` argument as it appears in the model selection table.

Value

`evidence` produces an object of class `evidence` with the following components:

<code>Model.high</code>	the model specified in <code>model.high</code> .
<code>Model.low</code>	the model specified in <code>model.low</code> .
<code>Ev.ratio</code>	the evidence ratio between the two models compared.

Author(s)

Marc J. Mazerolle

References

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

See Also

[AICc](#), [aictab](#), [bictab](#), [c_hat](#), [confset](#), [importance](#), [modavg](#), [modavgShrink](#), [modavgPred](#)

Examples

```
##run example from Burnham and Anderson (2002, p. 183) with two
##non-nested models
data(pine)
Cand.set <- list( )
Cand.set[[1]] <- lm(y ~ x, data = pine)
Cand.set[[2]] <- lm(y ~ z, data = pine)

##assign model names
Modnames <- c("raw density", "density corrected for resin content")

##compute model selection table
aicctable.out <- aicctab(cand.set = Cand.set, modnames = Modnames)

##compute evidence ratio
evidence(aic.table = aicctable.out, model.low = "raw density")
evidence(aic.table = aicctable.out) #gives the same answer
##round to 4 digits after decimal point
print(evidence(aic.table = aicctable.out, model.low = "raw density"),
      digits = 4)

##example with bictab
## Not run:
##compute model selection table
bictable.out <- bictab(cand.set = Cand.set, modnames = Modnames)
##compute evidence ratio
evidence(bictable.out, model.low = "raw density")

## End(Not run)

##run models for the Orthodont data set in nlme package
## Not run:
require(nlme)

##set up candidate model list
Cand.models <- list()
Cand.models[[1]] <- lme(distance ~ age, data = Orthodont, method = "ML")
##random is ~ age | Subject
Cand.models[[2]] <- lme(distance ~ age + Sex, data = Orthodont,
                        random = ~ 1, method = "ML")
Cand.models[[3]] <- lme(distance ~ 1, data = Orthodont, random = ~ 1,
                        method = "ML")

##create a vector of model names
Modnames <- paste("mod", 1:length(Cand.models), sep = " ")
```

```

##compute AICc table
aic.table.1 <- aictab(cand.set = Cand.models, modnames = Modnames,
                    second.ord = TRUE)

##compute evidence ratio between best model and second-ranked model
evidence(aic.table = aic.table.1)

##compute the same value but from an unsorted model selection table
evidence(aic.table = aictab(cand.set = Cand.models,
                          modnames = Modnames, second.ord = TRUE, sort = FALSE))

##compute evidence ratio between second-best model and third-ranked
##model
evidence(aic.table = aic.table.1, model.high = "mod1",
        model.low = "mod3")
detach(package:nlme)

## End(Not run)

```

extractCN

Compute Condition Number

Description

This function computes the condition number for models of `unmarkedFit` classes as the ratio of the largest eigenvalue of the Hessian matrix to the smallest eigenvalue of the Hessian matrix.

Usage

```

extractCN(mod, method = "svd", ...)

## S3 method for class 'unmarkedFit'
extractCN(mod, method = "svd", ...)

```

Arguments

<code>mod</code>	a model of one the <code>unmarkedFit</code> classes for which a condition number is requested.
<code>method</code>	specifies the method used to extract the singular values or eigenvalues from the Hessian matrix using singular value decomposition (<code>method = "svd"</code>) or eigenvalue decomposition (<code>method = "eigen"</code>).
<code>...</code>	additional arguments passed to the function.


```
##run model
fm.mallard <- pcount(~ ivel+ date + I(date^2) ~ length + elev + forest,
                    mallardUMF, K=30)

##compute condition number
extractCN(fm.mallard)

##compare against 'kappa'
kappa(fm.mallard@opt$hessian, exact = TRUE)
detach(package:unmarked)

## End(Not run)
```

extractLL

Extract Log-Likelihood of Model

Description

This function extracts the log-likelihood from an object of `coxme`, `coxph`, `lmekin`, `maxlikeFit`, `vglm`, or various `unmarkedFit` classes.

Usage

```
extractLL(mod, ...)

## S3 method for class 'coxme'
extractLL(mod, type = "Integrated", ...)

## S3 method for class 'coxph'
extractLL(mod, ...)

## S3 method for class 'lmekin'
extractLL(mod, ...)

## S3 method for class 'maxlikeFit'
extractLL(mod, ...)

## S3 method for class 'unmarkedFit'
extractLL(mod, ...)

## S3 method for class 'vglm'
extractLL(mod, ...)
```

Arguments

<code>mod</code>	an object of <code>coxme</code> , <code>coxph</code> , <code>lmekin</code> , <code>maxlikeFit</code> , <code>vglm</code> , or <code>unmarkedFit</code> class resulting from the fit of <code>distsamp</code> , <code>gdistsamp</code> , <code>gmultmix</code> , <code>multinomPois</code> , <code>gpcount</code> , <code>occu</code> , <code>occuRN</code> , <code>colext</code> , <code>pcount</code> , or <code>pcountOpen</code> .
------------------	---

... additional arguments passed to the function.

type a character string indicating whether the integrated partial likelihood ("Integrated") or penalized likelihood ("Penalized") is to be used for a coxme object.

Details

This utility function extracts the information from a `coxme`, `coxph`, `lmekin`, `maxlikeFit`, `vglm`, or `unmarkedFit` object resulting from `distsamp`, `gdistsamp`, `gmultmix`, `multinomPois`, `gpcount`, `occu`, `occuRN`, `colext`, `pcount`, or `pcountOpen`.

Value

These functions return the value of the log-likelihood of the model and associated degrees of freedom.

Author(s)

Marc J. Mazerolle

See Also

[AICc](#), [aictab](#), [coxme](#), [coxph](#), [lmekin](#), [maxlike](#), [distsamp](#), [gdistsamp](#), [occu](#), [occuRN](#), [colext](#), [pcount](#), [pcountOpen](#)

Examples

```
##single-season occupancy model example modified from ?occu
## Not run:
require(unmarked)
##single season
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
## add some fake covariates for illustration
siteCovs(pferUMF) <- data.frame(sitevar1 = rnorm(numSites(pferUMF)),
                               sitevar2 = rnorm(numSites(pferUMF)))

## observation covariates are in site-major, observation-minor order
obsCovs(pferUMF) <- data.frame(obsvar1 = rnorm(numSites(pferUMF) *
                                             obsNum(pferUMF)))

##run model set
fm1 <- occu(~ obsvar1 ~ sitevar1, pferUMF)

##extract log-likelihood
extractLL(fm1)
detach(package:unmarked)

## End(Not run)
```

extractSE	<i>Extract SE of Fixed Effects</i>
-----------	------------------------------------

Description

This function extracts the standard errors (SE) of the fixed effects of a mixed model fit with `coxme`, `glmer`, `lmer`, `lmerModLmerTest`, and `lme4` and adds the appropriate labels.

Usage

```
extractSE(mod, ...)  
  
## S3 method for class 'coxme'  
extractSE(mod, ...)  
  
## S3 method for class 'lme4'  
extractSE(mod, ...)  
  
## S3 method for class 'mer'  
extractSE(mod, ...)  
  
## S3 method for class 'merMod'  
extractSE(mod, ...)  
  
## S3 method for class 'lmerModLmerTest'  
extractSE(mod, ...)
```

Arguments

<code>mod</code>	an object of <code>coxme</code> , <code>lme4</code> , <code>mer</code> , <code>merMod</code> , or <code>lmerModTest</code> class.
<code>...</code>	additional arguments passed to the function.

Details

These extractor functions use `vcov.coxme`, `vcov.lme4`, `vcov.mer`, and `vcov.merMod`. Some of these functions are called by `modavg` and `modavgShrink`, depending on the class of the objects.

Value

Returns the SE's of the fixed effects with the appropriate labels for each.

Author(s)

Marc J. Mazerolle

See Also

[modavg](#), [glmer](#), [lmer](#), [coxme](#), [lme4](#)

Examples

```

##modified example from ?glmer
## Not run:
if(require(lme4)) {
  ##create proportion of incidence
  cbpp$prop <- cbpp$incidence/cbpp$size
  gm1 <- glmer(prop ~ period + (1 | herd), family = binomial,
              weights = size, data = cbpp)
  ##print summary
  summary(gm1)
  ##extract variance-covariance matrix of fixed effects
  vcov(gm1)
  ##extract SE's of fixed effects - no labels
  sqrt(diag(vcov(gm1))) #no labels
  extractSE(gm1) #with labels
  detach(package:lme4)
}

## End(Not run)

```

 extractX

Extract Predictors from Candidate Model List

Description

This function extracts the predictors used in candidate models. The function is currently implemented for `glm`, `glmmTMB`, `gls`, `lm`, `lme`, `merMod`, `lmerModLmerTest`, `r1m`, `survreg` object classes that are stored in a list as well as various models of `unmarkedFit` classes.

Usage

```

extractX(cand.set, ...)

## S3 method for class 'AICaov.lm'
extractX(cand.set, ...)

## S3 method for class 'AICglm.lm'
extractX(cand.set, ...)

## S3 method for class 'AICglmmTMB'
extractX(cand.set, ...)

## S3 method for class 'AIClm'
extractX(cand.set, ...)

## S3 method for class 'AICgls'
extractX(cand.set, ...)

```

```
## S3 method for class 'AIClme'
extractX(cand.set, ...)

## S3 method for class 'AICglmerMod'
extractX(cand.set, ...)

## S3 method for class 'AIClmerMod'
extractX(cand.set, ...)

## S3 method for class 'AIClmerModLmerTest'
extractX(cand.set, ...)

## S3 method for class 'AICr1m.lm'
extractX(cand.set, ...)

## S3 method for class 'AICsurvreg'
extractX(cand.set, ...)

## S3 method for class 'AICunmarkedFitOccu'
extractX(cand.set,
        parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitColExt'
extractX(cand.set,
        parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuRN'
extractX(cand.set,
        parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCount'
extractX(cand.set,
        parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCO'
extractX(cand.set,
        parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDS'
extractX(cand.set,
        parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGDS'
extractX(cand.set,
        parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuFP'
extractX(cand.set,
```

```

    parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMPois'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGMM'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGPC'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMulti'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMS'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuTTD'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMMO'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDSO'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGOccu'
extractX(cand.set,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuComm'
extractX(cand.set,
  parm.type = NULL, ...)

```

Arguments

cand.set	a list storing each of the models in the candidate model set.
parm.type	this argument specifies the parameter type on which the predictors will be extracted and is only relevant for models of unmarkedFit classes. The character strings supported vary with the type of model fitted. For unmarkedFitOccu,

unmarkedFitOccuMulti, and unmarkedFitOccuComm objects, either psi or detect can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For unmarkedFitColExt and unmarkedFitOccuTTD, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the initial year, colonization, extinction, and detectability, respectively. For unmarkedFitOccuFP objects, one can specify psi, detect, falsepos, and certain, for occupancy, detectability, probability of assigning false-positives, and probability detections are certain, respectively. For unmarkedFitOccuMS objects, possible values are psi, phi, or detect, denoting occupancy, transition, and detection probabilities, respectively. For unmarkedFitOccuRN objects, either lambda or detect can be entered for abundance and detectability parameters, respectively. For unmarkedFitPCount and unmarkedFitMPois objects, lambda or detect denote parameters on abundance and detectability, respectively. For unmarkedFitPCO, unmarkedFitMMO, and unmarkedFitDSO objects, one can enter lambda, gamma, omega, iota, or detect, to specify parameters on abundance, recruitment, apparent survival, immigration, and detectability, respectively. For unmarkedFitDS objects, lambda and detect are supported. For unmarkedFitGDS, lambda, phi, and detect denote abundance, availability, and detection probability, respectively. For unmarkedFitGMM and unmarkedFitGPC objects, lambda, phi, and detect denote abundance, availability, and detectability, respectively. For unmarkedFitGOccu objects, possible values are psi, phi, or detect, denoting occupancy, availability, and detection probabilities, respectively.

... additional arguments passed to the function.

Details

The candidate models must be stored in a list. The results of extractX are useful in preparing a newdata data frame to use in computing model-averaged predictions with modavgPred or differences between groups with modavgEffect (Burnham and Anderson 2002, Anderson 2008, Burnham et al. 2011).

Value

extractX returns an object of class extractX with the following components:

predictors	a character vector of the names of the predictors included in the model, excluding the intercept term.
data	a data frame or, in the case of unmarkedFit objects, a list of data frames (e.g., obsCovs, siteCovs, yearlySiteCovs).

Author(s)

Marc J. Mazerolle

References

Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

Burnham, K. P., Anderson, D. R., Huyvaert, K. P. (2011) AIC model selection and multimodel inference in behavioral ecology: some background, observations and comparisons. *Behavioral Ecology and Sociobiology* **65**, 23–25.

Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.

Pinheiro, J. C., Bates, D. M. (2000). *Mixed-effects Models in S and S-PLUS*. Springer Verlag: New York.

Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[extractCN](#), [extractSE](#), [modavgPred](#), [modavgCustom](#), [modavgEffect](#), [predict](#), [predictSE](#)

Examples

```
##example from subset of models in Table 1 in Mazerolle (2006)
data(dry.frog)

Cand.models <- list( )
Cand.models[[1]] <- lm(log_Mass_lost ~ Shade + Substrate +
                      cent_Initial_mass + Initial_mass2,
                      data = dry.frog)
Cand.models[[2]] <- lm(log_Mass_lost ~ Shade + Substrate +
                      cent_Initial_mass + Initial_mass2 +
                      Shade:Substrate, data = dry.frog)
Cand.models[[3]] <- lm(log_Mass_lost ~ cent_Initial_mass +
                      Initial_mass2, data = dry.frog)
Cand.models[[4]] <- lm(log_Mass_lost ~ Shade + cent_Initial_mass +
                      Initial_mass2, data = dry.frog)
Cand.models[[4]] <- lm(log_Mass_lost ~ Shade + cent_Initial_mass +
                      Initial_mass2, data = dry.frog)
Cand.models[[5]] <- lm(log_Mass_lost ~ Substrate + cent_Initial_mass +
                      Initial_mass2, data = dry.frog)

##assign names
names(Cand.models) <- paste(1:length(Cand.models))

##extract predictors from candidate model set
orig.data <- extractX(cand.set = Cand.models)
orig.data
str(orig.data)

## Not run:
##model-averaged prediction with original variables
modavgPred(Cand.models, newdata = orig.data$data)

## End(Not run)
```

```

##example of model-averaged predictions from N-mixture model (e.g., Royle 2004)
##modified from ?pcount
##each variable appears twice on lambda in the models
## Not run:
require(unmarked)
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
                                obsCovs = mallard.obs)
##set up models so that each variable on abundance appears twice
fm.mall.one <- pcount(~ ivel + date ~ length + forest, mallardUMF,
                    K = 30)
fm.mall.two <- pcount(~ ivel + date ~ elev + forest, mallardUMF,
                    K = 30)
fm.mall.three <- pcount(~ ivel + date ~ length + elev, mallardUMF,
                      K = 30)
fm.mall.four <- pcount(~ ivel + date ~ 1, mallardUMF, K = 30)

##model list
Cands <- list(fm.mall.one, fm.mall.two, fm.mall.three, fm.mall.four)
names(Cands) <- c("length + forest", "elev + forest", "length + elev",
                "null")

##extract predictors on lambda
lam.dat <- extractX(cand.set = Cands, parm.type = "lambda")
lam.dat
str(lam.dat)

##extract predictors on detectability
extractX(cand.set = Cands, parm.type = "detect")

##model-averaged predictions on lambda
##extract data
siteCovs <- lam.dat$data$siteCovs
##create vector of forest values
forest <- seq(min(siteCovs$forest),
             max(siteCovs$forest),
             length.out = 40)
dframe <- data.frame(forest = forest,
                   length = mean(siteCovs$length),
                   elev = mean(siteCovs$elev))
modavgPred(Cands, parm.type = "lambda",
          newdata = dframe)
detach(package:unmarked)

## End(Not run)

##example of model-averaged abundance from distance model
## Not run:
require(unmarked)
data(linetran) #example from ?distsamp

```

```

ltUMF <- with(linetran, {
  unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
    siteCovs = data.frame(Length, area, habitat),
    dist.breaks = c(0, 5, 10, 15, 20),
    tlength = linetran$Length * 1000, survey = "line",
    unitsIn = "m")
})

## Half-normal detection function. Density output (log scale). No covariates.
fm1 <- distsamp(~ 1 ~ 1, ltUMF)

## Halfnormal. Covariates affecting both density and and detection.
fm2 <- distsamp(~area + habitat ~ habitat, ltUMF)

## Hazard function. Covariates affecting both density and and detection.
fm3 <- distsamp(~area + habitat ~ habitat, ltUMF, keyfun="hazard")

##assemble model list
Cands <- list(fm1, fm2, fm3)

##model-average predictions on abundance
extractX(cand.set = Cands, parm.type = "lambda")
detach(package:unmarked)

## End(Not run)

##example using Orthodont data set from Pinheiro and Bates (2000)
## Not run:
require(nlme)

##set up candidate models
m1 <- gls(distance ~ age, correlation = corCompSymm(value = 0.5, form = ~ 1 | Subject),
  data = Orthodont, method = "ML")

m2 <- gls(distance ~ 1, correlation = corCompSymm(value = 0.5, form = ~ 1 | Subject),
  data = Orthodont, method = "ML")

##assemble in list
Cand.models <- list("age effect" = m1, "null model" = m2)

##model-averaged predictions
extractX(cand.set = Cand.models)
detach(package:nlme)

## End(Not run)

```

Description

This function extracts the distribution family and link function of a generalized linear mixed model fit with `glmer` or `lmer`.

Usage

```
fam.link.mer(mod)
```

Arguments

`mod` an object of `mer` or `merMod` class resulting from the fit of `glmer` or `lmer`.

Details

This utility function extracts the information from an `mer` or `merMod` object resulting from `glmer` or `lmer`. The function is called by `modavg`, `modavgEffect`, `modavgPred`, and `predictSE`.

Value

`fam.link.mer` returns a list with the following components:

<code>family</code>	the family of the distribution of the model.
<code>link</code>	the link function of the model.
<code>supp.link</code>	a character value indicating whether the link function used is supported by <code>predictSE</code> and <code>modavgPred</code> .

Author(s)

Marc J. Mazerolle

See Also

[modavg](#), [modavgPred](#), [predictSE](#), [glmer](#), [lmer](#)

Examples

```
##modified example from ?glmer
## Not run:
if(require(lme4)){
  ##create proportion of incidence
  cbpp$prop <- cbpp$incidence/cbpp$size
  gm1 <- glmer(prop ~ period + (1 | herd), family = binomial,
              weights = size, data = cbpp)
  fam.link.mer(gm1)
  gm2 <- glmer(prop ~ period + (1 | herd),
              family = binomial(link = "cloglog"), weights = size,
              data = cbpp)
  fam.link.mer(gm2)
}
```

```
## End(Not run)

##example with linear mixed model with Orthodont data from
##Pinheiro and Bates (2000)
## Not run:
data(Orthodont, package = "nlme")
m1 <- lmer(distance ~ Sex + (1 | Subject), data = Orthodont,
           REML = FALSE)
fam.link.mer(m1)
m2 <- glmer(distance ~ Sex + (1 | Subject),
            family = gaussian(link = "log"), data = Orthodont,
            REML = FALSE)
fam.link.mer(m2)
detach(package:lme4)

## End(Not run)
```

 fat

Fat Data and Body Measurements

Description

This data set illustrates the relationship between body measurements and body fat in 252 males aged between 21 and 81 years.

Usage

```
data(fat)
```

Format

A data frame with 252 rows and 26 variables.

Obs observation number.

Perc.body.fat.Brozek percent body fat using Brozek's equation, i.e., $457/Density - 414.2$.

Perc.body.fat.Siri percent body fat using Siri's equation, i.e., $495/Density - 450$.

Density density ($\frac{g}{cm^3}$).

Age age (years).

Weight weight (lbs).

Height height (inches).

Adiposity.index adiposity index computed as $Weight/Height^2$ ($\frac{kg}{m^2}$).

Fat.free.weight fat free weight computed as $(1 - Brozek's\ percent\ body\ fat) * Weight$ (lbs).

Neck.circ neck circumference (cm).

Chest.circ chest circumference (cm).

Abdomen.circ abdomen circumference (cm) measured at the umbilicus and level with the iliac crest.

Hip.circ hip circumference (cm).

Thigh.circ thigh circumference (cm).

Knee.circ knee circumference (cm).

Ankle.circ ankle circumference (cm).

Biceps.circ extended biceps circumference (cm).

Forearm.circ forearm circumference (cm).

Wrist.circ wrist circumference (cm).

inv.Density inverse of density ($\frac{cm^3}{g}$).

z1 log of weight divided by log of height (allometric measure).

z2 abdomen circumference divided by chest circumference (beer gut factor).

z3 index based on knee, wrist, and ankle circumference relative to height ($\frac{(Knee.circ*Wrist.circ*Ankle.circ)^{1/3}}{Height}$).

z4 fleshiness index based on biceps, thigh, forearm, knee, wrist, and ankle circumference ($\frac{Biceps.circ*Thigh.circ*Forearm.circ}{Knee.circ*Wrist.circ*Ankle.circ}$).

z5 age standardized to zero mean and unit variance.

z6 square of standardized age.

Details

Burnham and Anderson (2002, p. 268) use this data set to show model selection uncertainty in the context of all possible combinations of explanatory variables. The data are originally from Penrose et al. (1985) who used only the first 143 cases of the 252 observations in the data set. Johnson (1996) later used these data as an example of multiple regression. Note that observation number 42 originally had an erroneous height of 29.5 inches and that this value was changed to 69.5 inches.

Burnham and Anderson (2002, p. 274) created six indices based on the original measurements (i.e., z1 – z6). Although Burnham and Anderson (2002) indicate that the fleshiness index (z4) involved the cubic root in the equation, the result table for the full model on p. 276 suggests that the index did not include the cubic root for z4. The latter is the version of z4 used in the data set here.

Source

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

Johnson, J. W. (1996). Fitting percentage of body fat to simple body measurements. *Journal of Statistics Education* **4** [online].

Penrose, K., Nelson, A., Fisher, A. (1985) Generalized body composition prediction equation for men using simple measurement techniques *Medicine and Science in Sports and Exercise* **17**, 189.

Examples

```
data(fat)
str(fat)
```

gpa

GPA Data and Standardized Test Scores

Description

This data set features the first-year college GPA and four standardized tests conducted before matriculation.

Usage

```
data(gpa)
```

Format

A data frame with 20 rows and 5 variables.

gpa.y first-year GPA.

sat.math.x1 SAT math score.

sat.verb.x2 SAT verbal score.

hs.math.x3 high school math score.

hs.engl.x4 high school English score.

Details

Burnham and Anderson (2002, p. 225) use this data set originally from Graybill and Iyer (1994) to show model selection for all subsets regression.

Source

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

Graybill, F. A., Iyer, H. K. (1994) *Regression analysis: concepts and applications*. Duxbury Press: Belmont.

Examples

```
data(gpa)  
str(gpa)
```

ictab	<i>Create Model Selection Tables from User-supplied Information Criterion</i>
-------	---

Description

This function creates a model selection table from information criterion values supplied by the user. The table ranks the models based on the values of the information criterion and also displays delta values and information criterion weights.

Usage

```
ictab(ic, K, modnames = NULL, sort = TRUE, ic.name = NULL)
```

Arguments

ic	a vector of information criterion values for each model in the candidate model set.
K	a vector containing the number of estimated parameters for each model in the candidate model set.
modnames	a character vector of model names to identify each model in the model selection table. If NULL, generic names (e.g., Mod1, Mod2) are supplied in the table in the same order as the information criterion values.
sort	logical. If TRUE, the model selection table is ranked according to the values of the information criterion.
ic.name	a character string denoting the name of the information criterion input by the user. This character string will appear in certain column labels of the model selection table.

Details

ictab constructs a model selection table based on the information criterion values supplied by the user. This function is most useful for information criterion other than AIC, AICc, QAIC, and QAICc (e.g., WAIC: Watanabe 2010) or for classes not supported by aictab or bictab.

Value

ictab creates an object of class ictab with the following components:

Modname	the name of each model of the candidate model set.
K	the number of estimated parameters for each model.
IC	the values of the information criterion input by the user. If a value for ic.name is provided, the table modifies the labels of the table.
Delta_IC	the delta information criterion component comparing each model to the top-ranked model.

ModelLik	the relative likelihood of the model given the data ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
ICWt	the information criterion weights, also termed "model probabilities" sensu Burnham and Anderson (2002) and Anderson (2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
Cum.Wt	the cumulative information criterion weights. These are only meaningful if results in table are sorted in decreasing order of Akaike weights (i.e., <code>sort = TRUE</code>).

Author(s)

Marc J. Mazerolle

References

Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

Watanabe, S. (2010) Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research* **11**, 3571–3594.

See Also

[aictabCustom](#), [confset](#), [evidence](#), [modavgCustom](#), [modavgIC](#)

Examples

```
##create a vector of names to trace back models in set
Modnames <- c("global model", "interactive model",
             "additive model", "invertpred model")

##WAIC values
waic <- c(105.74, 107.36, 108.24, 100.57)
##number of effective parameters
effK <- c(7.45, 5.61, 6.14, 6.05)

##generate WAIC table
ictab(ic = waic, K = effK, modnames = Modnames,
      sort = TRUE, ic.name = "WAIC")
```

importance	<i>Compute Importance Values of Variable</i>
------------	--

Description

This function calculates the relative importance of variables (w+) based on the sum of Akaike weights (model probabilities) of the models that include the variable. Note that this measure of evidence is only appropriate when the variable appears in the same number of models as those that do not include the variable.

Usage

```
importance(cand.set, parm, modnames = NULL, second.ord = TRUE,
           nobs = NULL, ...)

## S3 method for class 'AICaov.lm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICbetareg'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICsc1m.clm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICclm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICclmm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICclogit.coxph'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICcoxme'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICcoxph'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)
```

```
## S3 method for class 'AICglm.lm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, ...)

## S3 method for class 'AICglmerMod'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AIClmerModLmerTest'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICglmmTMB'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, ...)

## S3 method for class 'AICgls'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AIClm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AIClme'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AIClmeKin'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICmaxlikeFit.list'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, ...)

## S3 method for class 'AICmer'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICmultinom.nnet'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, ...)

## S3 method for class 'AICnegbin.glm.lm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)
```

```
## S3 method for class 'AICnlmerMod'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICpolr'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICrIm.lm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICsurvreg'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, ...)

## S3 method for class 'AICunmarkedFitColExt'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccu'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuFP'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuRN'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCount'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCO'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDS'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGDS'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitMPois'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGMM'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGPC'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMulti'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMS'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL,
           ...)

## S3 method for class 'AICunmarkedFitOccuTTD'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL,
           ...)

## S3 method for class 'AICunmarkedFitMM0'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL,
           ...)

## S3 method for class 'AICunmarkedFitDS0'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL,
           ...)

## S3 method for class 'AICunmarkedFitG0ccu'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL,
           ...)

## S3 method for class 'AICunmarkedFitOccuComm'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobs = NULL, c.hat = 1, parm.type = NULL,
           ...)

## S3 method for class 'AICvglm'
importance(cand.set, parm, modnames = NULL,
```

```
second.ord = TRUE, nobS = NULL, c.hat = 1, ...)
```

```
## S3 method for class 'AICzeroinfl'
importance(cand.set, parm, modnames = NULL,
           second.ord = TRUE, nobS = NULL, ...)
```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>parm</code>	the parameter of interest for which a measure of relative importance is required.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table. If <code>NULL</code> , the function uses the names in the <code>cand.set</code> list of candidate models. If no names appear in the list, generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as in the list of candidate models.
<code>second.ord</code>	logical. If <code>TRUE</code> , the function returns the second-order Akaike information criterion (i.e., <code>AICc</code>).
<code>nobS</code>	this argument allows to specify a numeric value other than total sample size to compute the <code>AICc</code> (i.e., <code>nobS</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobS</code> .
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If <code>c.hat</code> > 1, <code>importance</code> will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>). This option is not supported for generalized linear mixed models of the <code>mer</code> or <code>merMod</code> classes.
<code>parm.type</code>	this argument specifies the parameter type on which the variable of interest will be extracted and is only relevant for models of <code>unmarkedFit</code> classes. The character strings supported vary with the type of model fitted. For <code>unmarkedFitOccu</code> , <code>unmarkedFitOccuMulti</code> , <code>unmarkedFitOccuComm</code> objects, either <code>psi</code> or <code>detect</code> can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For <code>unmarkedFitColExt</code> objects, possible values are <code>psi</code> , <code>gamma</code> , <code>epsilon</code> , and <code>detect</code> , for parameters on occupancy in the initial year, colonization, extinction, and detectability, respectively. For <code>unmarkedFitOccuTTD</code> objects, possible values are <code>psi</code> , <code>gamma</code> , <code>epsilon</code> , and <code>detect</code> , for parameters on occupancy in the initial year, colonization, extinction, and time-to-detection (lambda rate parameter), respectively. For <code>unmarkedFitOccuFP</code> objects, one can specify <code>psi</code> , <code>detect</code> , <code>falsepos</code> , and <code>certain</code> , for occupancy, detectability, probability of assigning false-positives, and probability detections are certain, respectively. For <code>unmarkedFitOccuMS</code> objects, possible values are <code>psi</code> , <code>phi</code> ,

or detect, denoting occupancy, transition, and detection probabilities, respectively. For `unmarkedFitOccuRN` objects, either `lambda` or `detect` can be entered for abundance and detectability parameters, respectively. For `unmarkedFitPCount` and `unmarkedFitMPois` objects, `lambda` or `detect` denote parameters on abundance and detectability, respectively. For `unmarkedFitPCO`, `unmarkedFitMMO`, and `unmarkedFitDSO` objects, one can enter `lambda`, `gamma`, `omega`, `iota`, or `detect`, to specify parameters on abundance, recruitment, apparent survival, immigration, and detectability, respectively. For `unmarkedFitDS` objects, `lambda` and `detect` are supported. For `unmarkedFitGDS`, `lambda`, `phi`, and `detect` denote abundance, availability, and detection probability, respectively. For `unmarkedFitGMM` and `unmarkedFitGPC` objects, `lambda`, `phi`, and `detect` denote abundance, availability, and detectability, respectively. For `unmarkedFitGOccu` objects, possible values are `psi`, `phi`, or `detect`, denoting occupancy, availability, and detection probabilities, respectively.

... additional arguments passed to the function.

Value

`importance` returns an object of class `importance` consisting of the following components:

<code>parm</code>	the parameter for which an importance value is required.
<code>w.plus</code>	the sum of Akaike weights for the models that include the parameter of interest.
<code>w.minus</code>	the sum of Akaike weights for the models that exclude the parameter of interest.

Author(s)

Marc J. Mazerolle

References

- Burnham, K. P., and Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICc](#), [aictab](#), [c_hat](#), [confset](#), [evidence](#), [modavg](#), [modavgShrink](#), [modavgPred](#)

Examples

```

##example on Orthodont data set in nlme
## Not run:
require(nlme)

##set up candidate model list
Cand.models <- list( )
Cand.models[[1]] <- lme(distance ~ age, data = Orthodont, method = "ML")
##random is ~ age | Subject
Cand.models[[2]] <- lme(distance ~ age + Sex, data = Orthodont,
                        random = ~ 1, method = "ML")
Cand.models[[3]] <- lme(distance ~ 1, data = Orthodont, random = ~ 1,
                        method = "ML")
Cand.models[[4]] <- lme(distance ~ Sex, data = Orthodont, random = ~ 1,
                        method = "ML")

##create a vector of model names
Modnames <- paste("mod", 1:length(Cand.models), sep = "")

importance(cand.set = Cand.models, parm = "age", modnames = Modnames,
           second.ord = TRUE, nobs = NULL)
##round to 4 digits after decimal point
print(importance(cand.set = Cand.models, parm = "age", modnames = Modnames,
                second.ord = TRUE, nobs = NULL), digits = 4)
detach(package:nlme)

## End(Not run)

##single-season occupancy model example modified from ?occu
## Not run:
require(unmarked)
##single season
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
## add some fake covariates for illustration
siteCovs(pferUMF) <- data.frame(sitevar1 = rnorm(numSites(pferUMF)),
                               sitevar2 = rnorm(numSites(pferUMF)))

## observation covariates are in site-major, observation-minor order
obsCovs(pferUMF) <- data.frame(obsvar1 = rnorm(numSites(pferUMF) *
                                              obsNum(pferUMF)))

##set up candidate model set
fm1 <- occu(~ obsvar1 ~ sitevar1, pferUMF)
fm2 <- occu(~ 1 ~ sitevar1, pferUMF)
fm3 <- occu(~ obsvar1 ~ sitevar2, pferUMF)
fm4 <- occu(~ 1 ~ sitevar2, pferUMF)
Cand.mods <- list(fm1, fm2, fm3, fm4)
Modnames <- c("fm1", "fm2", "fm3", "fm4")

##compute importance value for 'sitevar1' on occupancy

```

```

importance(cand.set = Cand.mods, modnames = Modnames, parm = "sitevar1",
           parm.type = "psi")
##compute importance value for 'obsvar1' on detectability
importance(cand.set = Cand.mods, modnames = Modnames, parm = "obsvar1",
           parm.type = "detect")

##example with multispecies occupancy modify from ?occuMulti
##Simulate 3 species data
N <- 80
nspecies <- 3
J <- 4

occ_covs <- as.data.frame(matrix(rnorm(N * 10),ncol=10))
names(occ_covs) <- paste('par',1:10,sep='')

det_covs <- list()
for (i in 1:nspecies){
  det_covs[[i]] <- matrix(rnorm(N*J),nrow=N)
}
names(det_covs) <- paste('par',1:nspecies,sep='')

##True vals
beta <- c(0.5,0.2,0.4,0.5,-0.1,-0.3,0.2,0.1,-1,0.1)
f1 <- beta[1] + beta[2]*occ_covs$par1
f2 <- beta[3] + beta[4]*occ_covs$par2
f3 <- beta[5] + beta[6]*occ_covs$par3
f4 <- beta[7]
f5 <- beta[8]
f6 <- beta[9]
f7 <- beta[10]
f <- cbind(f1,f2,f3,f4,f5,f6,f7)
z <- expand.grid(rep(list(1:0),nspecies))[,nspecies:1]
colnames(z) <- paste('sp',1:nspecies,sep='')
dm <- model.matrix(as.formula(paste0("~.^",nspecies,"-1")),z)

psi <- exp(f)
psi <- psi/rowSums(psi)

##True state
ztruth <- matrix(NA,nrow=N,ncol=nspecies)
for (i in 1:N){
  ztruth[i,] <- as.matrix(z[sample(8,1,prob=psi[i,]),])
}

p_true <- c(0.6,0.7,0.5)

## fake y data
y <- list()

for (i in 1:nspecies){
  y[[i]] <- matrix(NA,N,J)
  for (j in 1:N){

```

```

      for (k in 1:J){
        y[[i]][j,k] <- rbinom(1,1,ztruth[j,i]*p_true[i])
      }
    }
  }
names(y) <- c('coyote','tiger','bear')

##Create the unmarked data object
data <- unmarkedFrameOccuMulti(y=y,siteCovs=occ_covs,obsCovs=det_covs)

## Formulas for state and detection processes
## Length should match number/order of columns in fDesign
occFormulas <- c('~par1 + par2','~par2','~par3','~1','~1','~1','~1')
occFormulas2 <- c('~par1 + par3','~par1 + par2','~par1 + par2 + par3',
  "~ 1", "~1", "~ 1", "~1")

##Length should match number/order of species in data@ylist
detFormulas <- c('~1','~1','~1')

fit <- occuMulti(detFormulas,occFormulas,data)
fit2 <- occuMulti(detFormulas,occFormulas2,data)

##importance
importance(cand.set = list(fit, fit2), parm = "[coyote] par2",
  parm.type = "psi")

detach(package:unmarked)

## End(Not run)

```

 iron

Iron Content in Food

Description

This data set, originally from Adish et al. (1999), describes the iron content of food cooked in different pot types.

Usage

```
data(iron)
```

Format

A data frame with 36 rows and 3 variables.

Pot pot type, one of "aluminium", "clay", or "iron".

Food food type, one of "legumes", "meat", or "vegetables".

Iron iron content measured in mg/100 g of food.

Details

Heiberger and Holland (2004, p. 378) use these data as an exercise on two-way ANOVA with interaction.

Source

Heiberger, R. M., Holland, B. (2004) *Statistical Analysis and Data Display: an intermediate course with examples in S-Plus, R, and SAS*. Springer: New York.

Adish, A. A., Esrey, S. A., Gyorkos, T. W., Jean-Baptiste, J., Rojhani, A. (1999) Effect of consumption of food cooked in iron pots on iron status and growth of young children: a randomised trial. *The Lancet* **353**, 712–716.

Examples

```
data(iron)
str(iron)
```

 lizards

Habitat Preference of Lizards

Description

This data set describes the habitat preference of two species of lizards, *Anolis grahami* and *A. opalinus*, on the island of Jamaica and is originally from Schoener (1970). McCullagh and Nelder (1989) and Burnham and Anderson (2002) reanalyzed the data. Note that a typo occurs in table 3.11 of Burnham and Anderson (2002).

Usage

```
data(lizards)
```

Format

A data frame with 48 rows and 6 variables.

Insolation position of perch, either shaded or sunny.

Diameter diameter of the perch, either < 2 in or >= 2 in.

Height perch height, either < 5 or >= 5.

Time time of day, either morning, midday, or afternoon.

Species species observed, either grahami or opalinus.

Counts number of individuals observed.

Details

Burnham and Anderson (2002, p. 137) use this data set originally from Schoener (1970) to illustrate model selection for log-linear models.

Source

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

McCullagh, P., Nelder, J. A. (1989) *Generalized Linear Models*. Second edition. Chapman and Hall: New York.

Schoener, T. W. (1970) Nonsynchronous spatial overlap of lizards in patchy habitats. *Ecology* **51**, 408–418.

Examples

```
data(lizards)
## Not run:
##log-linear model as in Burnham and Anderson 2002, p. 137
##main effects
m1 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species,
          family = poisson, data = lizards)

##main effects and all second order interactions = base
m2 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
          Insolation:Diameter + Insolation:Height + Insolation:Time +
          Insolation:Species + Diameter:Height + Diameter:Time +
          Diameter:Species + Height:Time + Height:Species +
          Time:Species, family = poisson, data = lizards)

##base - DT
m3 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
          Insolation:Diameter + Insolation:Height + Insolation:Time +
          Insolation:Species + Diameter:Height + Diameter:Species +
          Height:Time + Height:Species + Time:Species,
          family = poisson, data = lizards)

##base + HDI + HDT + HDS
m4 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
          Insolation:Diameter + Insolation:Height + Insolation:Time +
          Insolation:Species + Diameter:Height + Diameter:Time +
          Diameter:Species + Height:Time + Height:Species +
          Time:Species + Height:Diameter:Insolation +
          Height:Diameter:Time + Height:Diameter:Species,
          family = poisson, data = lizards)

##base + HDI + HDS + HIT + HIS + HTS + ITS
m5 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
          Insolation:Diameter + Insolation:Height + Insolation:Time +
          Insolation:Species + Diameter:Height + Diameter:Time +
          Diameter:Species + Height:Time + Height:Species +
          Time:Species + Height:Diameter:Insolation +
          Height:Diameter:Species + Height:Insolation:Time +
          Height:Insolation:Species + Height:Time:Species +
          Insolation:Time:Species, family = poisson, data = lizards)
```

```

##base + HIT + HIS + HTS + ITS
m6 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
  Insolation:Diameter + Insolation:Height + Insolation:Time +
  Insolation:Species + Diameter:Height + Diameter:Time +
  Diameter:Species + Height:Time + Height:Species +
  Time:Species + Height:Insolation:Time +
  Height:Insolation:Species + Height:Time:Species +
  Insolation:Time:Species, family = poisson, data = lizards)

##base + HIS + HTS + ITS
m7 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
  Insolation:Diameter + Insolation:Height + Insolation:Time +
  Insolation:Species + Diameter:Height + Diameter:Time +
  Diameter:Species + Height:Time + Height:Species +
  Time:Species + Height:Insolation:Species +
  Height:Time:Species + Insolation:Time:Species,
  family = poisson, data = lizards)

##base + HIT + HIS + HTS + ITS - DT
m8 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
  Insolation:Diameter + Insolation:Height + Insolation:Time +
  Insolation:Species + Diameter:Height + Diameter:Species +
  Height:Time + Height:Species + Time:Species +
  Height:Insolation:Time + Height:Insolation:Species +
  Height:Time:Species + Insolation:Time:Species,
  family = poisson, data = lizards)

##base + HIT + HIS + ITS - DT
m9 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
  Insolation:Diameter + Insolation:Height + Insolation:Time +
  Insolation:Species + Diameter:Height + Diameter:Species +
  Height:Time + Height:Species + Time:Species +
  Height:Insolation:Time + Height:Insolation:Species +
  Insolation:Time:Species,
  family = poisson, data = lizards)

##base + HIT + HIS - DT
m10 <- glm(Counts ~ Insolation + Diameter + Height + Time + Species +
  Insolation:Diameter + Insolation:Height + Insolation:Time +
  Insolation:Species + Diameter:Height + Diameter:Species +
  Height:Time + Height:Species + Time:Species +
  Height:Insolation:Time + Height:Insolation:Species,
  family = poisson, data = lizards)

##set up in list
Cands <- list(m1, m2, m3, m4, m5, m6, m7, m8, m9, m10)
Modnames <- paste("m", 1:length(Cands), sep = "")

##model selection
library(AICcmodavg)
aictab(Cands, Modnames)

## End(Not run)

```

`mb.gof.test`*Compute MacKenzie and Bailey Goodness-of-fit Test for Single Season, Dynamic, and Royle-Nichols Occupancy Models*

Description

These functions compute the MacKenzie and Bailey (2004) goodness-of-fit test for single season occupancy models based on Pearson's chi-square and extend it to dynamic (multiple season) and Royle-Nichols (2003) occupancy models.

Usage

```
mb.chisq(mod, print.table = TRUE, ...)

## S3 method for class 'unmarkedFitOccu'
mb.chisq(mod, print.table = TRUE, ...)

## S3 method for class 'unmarkedFitColExt'
mb.chisq(mod, print.table = TRUE, ...)

## S3 method for class 'unmarkedFitOccuRN'
mb.chisq(mod, print.table = TRUE,
          maxK = NULL, ...)

mb.gof.test(mod, nsim = 5, plot.hist = TRUE, report = NULL,
            parallel = TRUE, ncores, cex.axis = 1, cex.lab = 1,
            cex.main = 1, lwd = 1, ...)

## S3 method for class 'unmarkedFitOccu'
mb.gof.test(mod, nsim = 5, plot.hist = TRUE,
            report = NULL, parallel = TRUE, ncores,
            cex.axis = 1, cex.lab = 1, cex.main = 1,
            lwd = 1, ...)

## S3 method for class 'unmarkedFitColExt'
mb.gof.test(mod, nsim = 5, plot.hist = TRUE,
            report = NULL, parallel = TRUE,
            ncores, cex.axis = 1, cex.lab = 1, cex.main = 1,
            lwd = 1, plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFitOccuRN'
mb.gof.test(mod, nsim = 5, plot.hist = TRUE,
            report = NULL, parallel = TRUE, ncores,
            cex.axis = 1, cex.lab = 1, cex.main = 1,
            lwd = 1, maxK = NULL, ...)
```

Arguments

<code>mod</code>	the model for which a goodness-of-fit test is required.
<code>print.table</code>	logical. Specifies if the detailed table of observed and expected values is to be included in the output.
<code>nsim</code>	the number of bootstrapped samples.
<code>plot.hist</code>	logical. Specifies that a histogram of the bootstrapped test statistic is to be included in the output. For dynamic occupancy models, this produces a histogram of the sum of the season-specific chi-squares for each bootstrap sample.
<code>report</code>	If NULL, the test statistic for each iteration is not printed in the terminal. Otherwise, an integer indicating the number of values of the test statistic that should be printed on the same line. For example, if <code>report = 3</code> , the values of the test statistic for three iterations are reported on each line.
<code>parallel</code>	logical. If TRUE, requests that <code>parboot</code> use multiple cores to accelerate computations of the bootstrap.
<code>ncores</code>	integer indicating the number of cores to use when bootstrapping in parallel during the analysis of simulated data sets. If <code>ncores</code> is not specified, one less than the number of available cores on the computer is used.
<code>cex.axis</code>	expansion factor influencing the size of axis annotations on plots produced by the function.
<code>cex.lab</code>	expansion factor influencing the size of axis labels on plots produced by the function.
<code>cex.main</code>	expansion factor influencing the size of the main title above plots produced by the function.
<code>lwd</code>	expansion factor of line width on plots produced by the function.
<code>plot.seasons</code>	logical. For dynamic occupancy models, specifies that a histogram of the bootstrapped test statistic for each primary period (season) is to be included in the output.
<code>maxK</code>	the number of support points used as the summation index in the likelihood of the Royle-Nichols model (2003). If NULL, extracts the value of <code>K</code> used by <code>occuRN</code> to fit the model to the original data.
<code>...</code>	additional arguments passed to the function.

Details

MacKenzie and Bailey (2004) and MacKenzie et al. (2006) suggest using the Pearson chi-square to assess the fit of single season occupancy models (MacKenzie et al. 2002). Given low expected frequencies, the chi-square statistic will deviate from the theoretical distribution and it is recommended to use a parametric bootstrap approach to obtain P-values with the `parboot` function of the `unmarked` package. `mb.chisq` computes the table of observed and expected values based on the detection histories and single season occupancy model used. `mb.gof.test` calls internally `mb.chisq` and `parboot` to generate simulated data sets based on the model and compute the MacKenzie and Bailey test statistic. Missing values are accommodated by creating cohorts for each pattern of missing values.

It is also possible to obtain an estimate of the overdispersion parameter (\hat{c}) for the model at hand by dividing the observed chi-square statistic by the mean of the statistics obtained from simulation.

This test is extended to dynamic occupancy models of MacKenzie et al. (2003) by using the occupancy estimates for each season obtained from the model. These estimates are then used to compute the predicted and observed frequencies separately within each season. The chi-squares are then summed to be used as the test statistic for the dynamic occupancy model.

Note that values of $\hat{c} > 1$ indicate overdispersion (variance $>$ mean), but that values much higher than 1 (i.e., > 4) probably indicate lack-of-fit. In cases of moderate overdispersion, one usually multiplies the variance-covariance matrix of the estimates by \hat{c} . As a result, the SE's of the estimates are inflated (\hat{c} is also known as a variance inflation factor).

In model selection, \hat{c} should be estimated from the global model and the same value of \hat{c} applied to the entire model set. Specifically, a global model is the most complex model from which all the other models of the set are simpler versions (nested). When no single global model exists in the set of models considered, such as when sample size does not allow a complex model, one can estimate \hat{c} from 'subglobal' models. Here, 'subglobal' models denote models from which only a subset of the models of the candidate set can be derived. In such cases, one can use the smallest value of \hat{c} for model selection (Burnham and Anderson 2002).

Note that \hat{c} counts as an additional parameter estimated and should be added to K . All functions in package `AICcmodavg` automatically add 1 when the `c.hat` argument > 1 and apply the same value of \hat{c} for the entire model set. When $\hat{c} > 1$, functions compute quasi-likelihood information criteria (either QAICc or QAIC, depending on the value of the `second.ord` argument) by scaling the log-likelihood of the model by \hat{c} . The value of \hat{c} can influence the ranking of the models: as \hat{c} increases, QAIC or QAICc will favor models with fewer parameters. As an additional check against this potential problem, one can generate several model selection tables by incrementing values of \hat{c} to assess the model selection uncertainty. If ranking changes little up to the \hat{c} value observed, one can be confident in making inference.

In cases of underdispersion ($\hat{c} < 1$), it is recommended to keep the value of \hat{c} to 1. However, note that values of $\hat{c} \ll 1$ can also indicate lack-of-fit and that an alternative model should be investigated.

Value

`mb.chisq` returns the following components for single-season and Royle-Nichols occupancy models:

<code>chisq.table</code>	the table of observed and expected values for each detection history and its chi-square component (if <code>print.table = TRUE</code>). Note that the table only shows the observed detection histories. Unobserved detection histories are not shown, but are included in the computation of the test statistic.
<code>chi.square</code>	the Pearson chi-square statistic. This test statistic should be compared against a bootstrap distribution instead of the theoretical chi-square distribution because low expected frequencies invalidate the chi-square assumption.
<code>model.type</code>	the model type, either <code>single-season</code> , <code>royle-nichols</code> , or <code>dynamic</code> .

`mb.chisq` returns the following additional components for dynamic occupancy models:

<code>tables</code>	a list containing the season-specific chi-square tables (if <code>print.table = TRUE</code>).
---------------------	--

<code>all.chisq</code>	an element containing the season-specific chi-squares.
<code>n.seasons</code>	the number of primary periods (seasons).
<code>n.seasons</code>	the number of primary periods (seasons).
<code>missing.seasons</code>	logical vector indicating whether data were collected or not during a given season (primary period), where TRUE indicates that no data were collected during the season.

`mb.gof.test` returns the following components for single-season and Royle-Nichols occupancy models:

<code>chisq.table</code>	the table of observed and expected values for each detection history and its chi-square component.
<code>chi.square</code>	the Pearson chi-square statistic.
<code>t.star</code>	the bootstrapped chi-square test statistics (i.e., obtained for each of the simulated data sets).
<code>p.value</code>	the P-value assessed from the parametric bootstrap, computed as the proportion of the simulated test statistics greater than or equal to the observed test statistic.
<code>c.hat.est</code>	the estimate of the overdispersion parameter, \hat{c} , computed as the observed test statistic divided by the mean of the simulated test statistics.
<code>nsim</code>	the number of bootstrap samples. The recommended number of samples varies with the data set, but should be on the order of 1000 or 5000, and in cases with a large number of visits, even 10 000 samples, namely to reduce the effect of unusually small values of the test statistics.

`mb.gof.test` returns the following additional components for dynamic occupancy models:

<code>chisq.table</code>	a list including the table of observed and expected values for each detection history and its chi-square component for each primary period (season).
<code>chi.square</code>	the chi-square test statistic, as the sum of the chi-squares across the primary periods.
<code>p.value</code>	a list of the P-values for each of the primary periods, computed separately as the proportion of the simulated test statistics greater than or equal to the observed test statistic.
<code>p.global</code>	the P-value of the chi-square test statistic for the dynamic occupancy model. This P-value is computed as the proportion of the simulated sums of chi-squares greater than or equal to the observed sum of chi-squares across the primary periods.

Author(s)

Marc J. Mazerolle

References

- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- MacKenzie, D. I., Bailey, L. L. (2004) Assessing the fit of site-occupancy models. *Journal of Agricultural, Biological, and Environmental Statistics* **9**, 300–318.
- MacKenzie, D. I., Nichols, J. D., Royle, J. A., Pollock, K. H., Bailey, L. L., Hines, J. E. (2006) *Occupancy estimation and modeling: inferring patterns and dynamics of species occurrence*. Academic Press: New York.
- Royle, J. A., Nichols, J. D. (2003) Estimating abundance from repeated presence-absence data or point counts. *Ecology* **84**, 777–790.

See Also

[AICc](#), [c_hat](#), [colext](#), [evidence](#), [modavg](#), [importance](#), [modavgPred](#), [Nmix.gof.test](#), [occu](#), [parboot](#)

Examples

```
##single-season occupancy model example modified from ?occu
## Not run:
require(unmarked)
##single season
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
## add some fake covariates for illustration
siteCovs(pferUMF) <- data.frame(sitevar1 = rnorm(numSites(pferUMF)),
                               sitevar2 = rnorm(numSites(pferUMF)))

## observation covariates are in site-major, observation-minor order
obsCovs(pferUMF) <- data.frame(obsvar1 = rnorm(numSites(pferUMF) *
                                             obsNum(pferUMF)))

##run model
fm1 <- occu(~ obsvar1 ~ sitevar1, pferUMF)

##compute observed chi-square
obs <- mb.chisq(fm1)
obs
##round to 4 digits after decimal point
print(obs, digits.vals = 4)

##compute observed chi-square, assess significance, and estimate c-hat
obs.boot <- mb.gof.test(fm1, nsim = 3)
```

```

##note that more bootstrap samples are recommended
##(e.g., 1000, 5000, or 10 000)
obs.boot
print(obs.boot, digits.vals = 4, digits.chisq = 4)

##data with missing values
mat1 <- matrix(c(0, 0, 0), nrow = 120, ncol = 3, byrow = TRUE)
mat2 <- matrix(c(0, 0, 1), nrow = 23, ncol = 3, byrow = TRUE)
mat3 <- matrix(c(1, NA, NA), nrow = 42, ncol = 3, byrow = TRUE)
mat4 <- matrix(c(0, 1, NA), nrow = 33, ncol = 3, byrow = TRUE)
y.mat <- rbind(mat1, mat2, mat3, mat4)
y.sim.data <- unmarkedFrameOccu(y = y.mat)
m1 <- occu(~ 1 ~ 1, data = y.sim.data)

mb.gof.test(m1, nsim = 3)
##note that more bootstrap samples are recommended
##(e.g., 1000, 5000, or 10 000)

##modifying graphical parameters
mb.gof.test(m1, nsim = 3,
cex.axis = 1.2, #axis annotations are 1.2 the default size
cex.lab = 1.2, #axis labels are 1.2 the default size
lwd = 2) #line width is twice the default width
detach(package:unmarked)

## End(Not run)

```

min.trap

Anuran Larvae Counts in Minnow Traps Across Pond Type

Description

This data set consists of counts of anuran larvae as a function of pond type, pond perimeter, and presence of water scorpions (*Ranatra* sp.).

Usage

```
data(min.trap)
```

Format

A data frame with 24 observations on the following 6 variables.

Type pond type, denotes the location of ponds in either bog or upland environment

Num_anura number of anuran larvae in minnow traps

Effort number of trap nights (i.e., number of traps x days of trapping) in each pond

Perimeter pond perimeter in meters

Num_ranatra number of water scorpions trapped in minnow traps

log.Perimeter natural log of perimeter

Details

Mazerolle (2006) uses this data set to illustrate model selection for Poisson regression with low overdispersion.

Source

Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.

Examples

```
data(min.trap)
## maybe str(min.trap) ; plot(min.trap) ...
```

modavg	<i>Compute Model-averaged Parameter Estimate (Multimodel Inference)</i>
--------	---

Description

This function model-averages the estimate of a parameter of interest among a set of candidate models, computes the unconditional standard error and unconditional confidence intervals as described in Buckland et al. (1997) and Burnham and Anderson (2002). This model-averaged estimate is also referred to as a natural average of the estimate by Burnham and Anderson (2002, p. 152).

Usage

```
modavg(cand.set, parm, modnames = NULL, second.ord = TRUE, nobs = NULL,
       uncond.se = "revised", conf.level = 0.95, exclude = NULL, warn =
       TRUE, ...)

## S3 method for class 'AICaov.lm'
modavg(cand.set, parm, modnames = NULL, second.ord =
       TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICbetareg'
modavg(cand.set, parm, modnames = NULL, second.ord =
       TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICsc1m.c1m'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICc1m'
modavg(cand.set, parm, modnames = NULL,
```

```
        second.ord = TRUE, nobs = NULL, uncond.se = "revised",
        conf.level = 0.95, exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICclmm'
modavg(cand.set, parm, modnames = NULL, second.ord
       = TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICcoxme'
modavg(cand.set, parm, modnames = NULL, second.ord
       = TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICcoxph'
modavg(cand.set, parm, modnames = NULL, second.ord
       = TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICglm.lm'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       gamdisp = NULL, ...)

## S3 method for class 'AICglmTMB'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       ...)

## S3 method for class 'AICgls'
modavg(cand.set, parm, modnames = NULL, second.ord =
       TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICHurdle'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AIClm'
modavg(cand.set, parm, modnames = NULL, second.ord =
       TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AIClme'
modavg(cand.set, parm, modnames = NULL, second.ord =
       TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
```

```
        exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AIClmekin'
```

```
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AICmaxlikeFit.list'
```

```
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       ...)
```

```
## S3 method for class 'AICmer'
```

```
modavg(cand.set, parm, modnames = NULL, second.ord =
       TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AIClmerMod'
```

```
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AIClmerModLmerTest'
```

```
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AICglmerMod'
```

```
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AICmultinom.nnet'
```

```
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       ...)
```

```
## S3 method for class 'AICnegbin.glm.lm'
```

```
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AICpolr'
```

```
modavg(cand.set, parm, modnames = NULL, second.ord
       = TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)
```

```
## S3 method for class 'AICr1m.lm'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICsurvreg'
modavg(cand.set, parm, modnames = NULL, second.ord =
       TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICvglm'
modavg(cand.set, parm, modnames = NULL, second.ord
       = TRUE, nobs = NULL, uncond.se = "revised", conf.level = 0.95,
       exclude = NULL, warn = TRUE, c.hat = 1, ...)

## S3 method for class 'AICzeroinfl'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, ...)

## S3 method for class 'AICunmarkedFitOccu'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitColExt'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuRN'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCount'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCO'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
```

```
        conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
        parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitDS'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitGDS'
modavg(cand.set, parm, modnames = NULL,
       second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitOccuFP'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitMPois'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitGMM'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitGPC'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitOccuMulti'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
       conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
       parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitOccuMS'
modavg(cand.set, parm, modnames =
       NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
```

```

    conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
    parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuTTD'
modavg(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMMO'
modavg(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDSO'
modavg(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGOccu'
modavg(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuComm'
modavg(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, exclude = NULL, warn = TRUE, c.hat = 1,
  parm.type = NULL, ...)

```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>parm</code>	the parameter of interest, enclosed between quotes, for which a model-averaged estimate is required. For a categorical variable, the label of the estimate must be included as it appears in the output (see 'Details' below).
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table. If <code>NULL</code> , the function uses the names in the <code>cand.set</code> list of candidate models. If no names appear in the list, generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as in the list of candidate models.
<code>second.ord</code>	logical. If <code>TRUE</code> , the function returns the second-order Akaike information criterion (i.e., <code>AICc</code>).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the <code>AICc</code> (i.e., <code>nobs</code> defaults to total number of observations). This

is relevant only for mixed models or various models of `unmarkedFit` classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of `nobs`.

<code>uncond.se</code>	either, "old", or "revised", specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With <code>uncond.se = "old"</code> , computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With <code>uncond.se = "revised"</code> , equation 6.12 of Burnham and Anderson (2002) is used. Anderson (2008, p. 111) recommends use of the revised version for the computation of unconditional standard errors and it is now the default. Note that versions of package <code>AICcmodavg < 1.04</code> used the old method to compute unconditional standard errors.
<code>conf.level</code>	the confidence level ($1 - \alpha$) requested for the computation of unconditional confidence intervals.
<code>exclude</code>	this argument excludes models based on the terms specified for the computation of a model-averaged estimate of <code>parm</code> . The <code>exclude</code> argument is set to <code>NULL</code> by default and does not exclude any models other than those without the <code>parm</code> . When <code>parm</code> is a main effect but is also involved in interactions/polynomial terms in some models, one should specify the interaction/polynomial terms as a list to exclude models with these terms from the computation of model-averaged estimate of the main effect (e.g., <code>exclude = list("sex:mass", "mass2")</code>). See 'Details' and 'Examples' below.
<code>warn</code>	logical. If <code>TRUE</code> , <code>modavg</code> performs a check and issues a warning when the value in <code>parm</code> occurs more than once in any given model. This is a check for potential interaction/polynomial terms in the model when such terms are constructed with the usual operators (e.g., <code>I()</code> for polynomial terms, <code>:</code> for interaction terms).
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with <code>trials > 1</code> (i.e., <code>success/trial</code> or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If <code>c.hat > 1</code> , <code>modavg</code> will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>). This option is not supported for generalized linear mixed models of the <code>mer</code> or <code>merMod</code> classes.
<code>gamdisp</code>	if gamma GLM is used, the dispersion parameter should be specified here to apply the same value to each model.
<code>parm.type</code>	this argument specifies the parameter type on which the model-averaged estimate of a predictor will be computed and is only relevant for models of <code>unmarkedFit</code> classes. The character strings supported vary with the type of model fitted. For <code>unmarkedFitOccu</code> , <code>unmarkedFitOccuMulti</code> , and <code>unmarkedFitOccuComm</code> objects, either <code>psi</code> or <code>detect</code> can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For <code>unmarkedFitColExt</code> objects, possible values are <code>psi</code> , <code>gamma</code> , <code>epsilon</code> , and <code>detect</code> , for parameters

on occupancy in the initial year, colonization, extinction, and detectability, respectively. For `unmarkedFitOccuTTD` objects, possible values are `psi`, `gamma`, `epsilon`, and `detect`, for parameters on occupancy in the initial year, colonization, extinction, and time-to-detection (lambda rate parameter), respectively. For `unmarkedFitOccuFP` objects, one can specify `psi`, `detect`, `falsepos`, and `certain`, for occupancy, detectability, probability of assigning false-positives, and probability detections are certain, respectively. For `unmarkedFitOccuMS` objects, possible values are `psi`, `phi`, or `detect`, denoting occupancy, transition, and detection probabilities, respectively. For `unmarkedFitOccuRN` objects, either `lambda` or `detect` can be entered for abundance and detectability parameters, respectively. For `unmarkedFitPCount` and `unmarkedFitMPois` objects, `lambda` or `detect` denote parameters on abundance and detectability, respectively. For `unmarkedFitPCO`, `unmarkedFitMMO`, and `unmarkedFitDSO` objects, one can enter `lambda`, `gamma`, `omega`, `iota`, or `detect`, to specify parameters on abundance, recruitment, apparent survival, immigration, and detectability, respectively. For `unmarkedFitDS` objects, `lambda` and `detect` are supported. For `unmarkedFitGDS`, `lambda`, `phi`, and `detect` denote abundance, availability, and detection probability, respectively. For `unmarkedFitGMM` and `unmarkedFitGPC` objects, `lambda`, `phi`, and `detect` denote abundance, availability, and detectability, respectively. For `unmarkedFitGOccu` objects, possible values are `psi`, `phi`, or `detect`, denoting occupancy, availability, and detection probabilities, respectively.

... additional arguments passed to the function.

Details

The parameter for which a model-averaged estimate is requested must be specified with the `parm` argument and must be identical to its label in the model output (e.g., from `summary`). For factors, one must specify the name of the variable and the level of interest. `modavg` includes checks to find variations of interaction terms specified in the `parm` and `exclude` arguments. However, to avoid problems, one should specify interaction terms consistently for all models: e.g., either `a:b` or `b:a` for all models, but not a mixture of both.

You must exercise caution when some models include interaction or polynomial terms, because main effect terms do not have the same interpretation when they also appear in an interaction/polynomial term in the same model. In such cases, one should exclude models containing interaction terms where the main effect is involved with the `exclude` argument of `modavg`. Note that `modavg` checks for potential cases of multiple instances of a variable appearing more than once in a given model (presumably in an interaction) and issues a warning. To correctly compute the model-averaged estimate of a main effect involved in interaction/polynomial terms, specify the interaction term(s) that should not appear in the same model with the `exclude` argument. This will effectively exclude models from the computation of the model-averaged estimate.

When `warn = TRUE`, `modavg` looks for matches among the labels of the estimates with `identical`. It then compares the results to partial matches with `regexpr`, and issues a warning whenever they are different. As a result, `modavg` may issue a warning when some variables or levels of categorical variables have nested names (e.g., `treat`, `treat10`; `L`, `TL`). When this warning is only due to the presence of similarly named variables in the models (and NOT due to interaction terms), you can suppress this warning by setting `warn = FALSE`.

The model-averaging estimator implemented in `modavg` is known to be biased away from 0 when there is substantial model selection uncertainty (Cade 2015). In such instances, it is recommended to use the model-averaging shrinkage estimator (i.e., `modavgShrink`) for inference on beta estimates or to focus on model-averaged effect sizes (`modavgEffect`) and model-averaged predictions (`modavgPred`).

`modavg` is implemented for a list containing objects of `aov`, `betareg`, `clm`, `clmm`, `clogit`, `coxme`, `coxph`, `glm`, `glmmTMB`, `gls`, `hurdle`, `lm`, `lme`, `lmeKin`, `maxlikeFit`, `mer`, `glmerMod`, `lmerMod`, `lmerModLmerTest`, `multinom`, `polr`, `rlm`, `survreg`, `vglm`, `zeroinfl` classes as well as various models of `unmarkedFit` classes.

Value

`modavg` creates an object of class `modavg` with the following components:

Parameter	the parameter for which a model-averaged estimate was obtained.
Mod.avg.table	the reduced model selection table based on models including the parameter of interest.
Mod.avg.beta	the model-averaged estimate based on all models including the parameter of interest (see 'Details' above regarding the exclusion of models where parameter of interest is involved in an interaction).
Uncond.SE	the unconditional standard error for the model-averaged estimate (as opposed to the conditional SE based on a single model).
Conf.level	the confidence level used to compute the confidence interval.
Lower.CL	the lower confidence limit.
Upper.CL	the upper confidence limit.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Buckland, S. T., Burnham, K. P., Augustin, N. H. (1997) Model selection: an integral part of inference. *Biometrics* **53**, 603–618.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Cade, B. S. (2015) Model averaging and muddled multimodel inferences. *Ecology* **96**, 2370–2382.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67–118.

MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.

MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.

Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.

Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICc](#), [aictab](#), [c_hat](#), [confset](#), [evidence](#), [importance](#), [modavgCustom](#), [modavgEffect](#), [modavgShrink](#), [modavgPred](#)

Examples

```
##anuran larvae example modified from Mazerolle (2006)
##these are different models than in the paper
data(min.trap)
##assign "UPLAND" as the reference level as in Mazerolle (2006)
min.trap$Type <- relevel(min.trap$Type, ref = "UPLAND")

##set up candidate models
Cand.mod <- list( )
##global model
Cand.mod[[1]] <- glm(Num_anura ~ Type + log.Perimeter +
                    Type:log.Perimeter + Num_ranatra,
                    family = poisson, offset = log(Effort),
                    data = min.trap)
##interactive model
Cand.mod[[2]] <- glm(Num_anura ~ Type + log.Perimeter +
                    Type:log.Perimeter, family = poisson,
                    offset = log(Effort), data = min.trap)
##additive model
Cand.mod[[3]] <- glm(Num_anura ~ Type + log.Perimeter, family = poisson,
                    offset = log(Effort), data = min.trap)
##Predator model
Cand.mod[[4]] <- glm(Num_anura ~ Type + Num_ranatra, family = poisson,
                    offset = log(Effort), data = min.trap)

##check c-hat for global model
c_hat(Cand.mod[[1]]) #uses Pearson's chi-square/df
##note the very low overdispersion: in this case, the analysis could be
##conducted without correcting for c-hat as its value is reasonably close
##to 1

##assign names to each model
Modnames <- c("global model", "interactive model",
              "additive model", "invertpred model")
```

```

##model selection
aictab(Cand.mod, Modnames)

##compute model-averaged estimates for parameters appearing in top
##models
modavg(parm = "Num_ranatra", cand.set = Cand.mod, modnames = Modnames)
##round to 4 digits after decimal point
print(modavg(parm = "Num_ranatra", cand.set = Cand.mod,
             modnames = Modnames), digits = 4)

##model-averaging a variable involved in an interaction
##the following produces an error - because the variable is involved
##in an interaction in some candidate models
## Not run: modavg(parm = "TypeBOG", cand.set = Cand.mod,
                  modnames = Modnames)
## End(Not run)

##exclude models where the variable is involved in an interaction
##to get model-averaged estimate of main effect
modavg(parm = "TypeBOG", cand.set = Cand.mod, modnames = Modnames,
       exclude = list("Type:log.Perimeter"))

##to get model-averaged estimate of interaction
modavg(parm = "TypeBOG:log.Perimeter", cand.set = Cand.mod,
       modnames = Modnames)

##beware of variables that have similar names
set.seed(seed = 4)
resp <- rnorm(n = 40, mean = 3, sd = 1)
size <- rep(c("small", "medsmall", "high", "medhigh"), times = 10)
set.seed(seed = 4)
mass <- rnorm(n = 40, mean = 2, sd = 0.1)
mass2 <- mass^2
age <- rpois(n = 40, lambda = 3.2)
agecorr <- rpois(n = 40, lambda = 2)
sizecat <- rep(c("a", "ab"), times = 20)
data1 <- data.frame(resp = resp, size = size, sizecat = sizecat,
                   mass = mass, mass2 = mass2, age = age,
                   agecorr = agecorr)

##set up models in list
Cand <- list( )
Cand[[1]] <- lm(resp ~ size + agecorr, data = data1)
Cand[[2]] <- lm(resp ~ size + mass + agecorr, data = data1)
Cand[[3]] <- lm(resp ~ age + mass, data = data1)
Cand[[4]] <- lm(resp ~ age + mass + mass2, data = data1)
Cand[[5]] <- lm(resp ~ mass + mass2 + size, data = data1)
Cand[[6]] <- lm(resp ~ mass + mass2 + sizecat, data = data1)
Cand[[7]] <- lm(resp ~ sizecat, data = data1)

```

```

Cand[[8]] <- lm(resp ~ sizecat + mass + sizecat:mass, data = data1)
Cand[[9]] <- lm(resp ~ agecorr + sizecat + mass + sizecat:mass,
                data = data1)

##create vector of model names
Modnames <- paste("mod", 1:length(Cand), sep = "")

aictab(cand.set = Cand, modnames = Modnames, sort = TRUE) #correct

##as expected, issues warning as mass occurs sometimes with "mass2" or
##"sizecat:mass" in some of the models
## Not run: modavg(cand.set = Cand, parm = "mass", modnames = Modnames)

##no warning issued, because "age" and "agecorr" never appear in same model
modavg(cand.set = Cand, parm = "age", modnames = Modnames)

##as expected, issues warning because warn=FALSE, but it is a very bad
##idea in this example since "mass" occurs with "mass2" and "sizecat:mass"
##in some of the models - results are INCORRECT
## Not run: modavg(cand.set = Cand, parm = "mass", modnames = Modnames,
                  warn = FALSE)
## End(Not run)

##correctly excludes models with quadratic term and interaction term
##results are CORRECT
modavg(cand.set = Cand, parm = "mass", modnames = Modnames,
       exclude = list("mass2", "sizecat:mass"))

##correctly computes model-averaged estimate because no other parameter
##occurs simultaneously in any of the models
modavg(cand.set = Cand, parm = "sizesmall", modnames = Modnames) #correct

##as expected, issues a warning because "sizecat:mass" occurs sometimes in
##an interaction in some models
## Not run: modavg(cand.set = Cand, parm = "sizecat:mass",
                  modnames = Modnames)
## End(Not run)

##exclude models with "sizecat:mass" interaction - results are CORRECT
modavg(cand.set = Cand, parm = "sizecat:mass", modnames = Modnames,
       exclude = list("sizecat:mass"))

##example with multiple-season occupancy model modified from ?colext
##this is a bit longer
## Not run:
require(unmarked)
data(frogs)
umf <- formatMult(masspcru)
obsCovs(umf) <- scale(obsCovs(umf))
siteCovs(umf) <- rnorm(numSites(umf))
yearlySiteCovs(umf) <- data.frame(year = factor(rep(1:7,

```

```

                                numSites(umf)))

##set up model with constant transition rates
fm <- colext(psiformula = ~ 1, gammaformula = ~ 1, epsilonformula = ~ 1,
             pformula = ~ JulianDate + I(JulianDate^2), data = umf,
             control = list(trace=1, maxit=1e4))

##model with with year-dependent transition rates
fm.yearly <- colext(psiformula = ~ 1, gammaformula = ~ year,
                   epsilonformula = ~ year,
                   pformula = ~ JulianDate + I(JulianDate^2),
                   data = umf)

##store in list and assign model names
Cand.mods <- list(fm, fm.yearly)
Modnames <- c("psi1(.)gam(.)eps(.)p(Date + Date2)",
              "psi1(.)gam(Year)eps(Year)p(Date + Date2)")

##compute model-averaged estimate of occupancy in the first year
modavg(cand.set = Cand.mods, modnames = Modnames, parm = "(Intercept)",
       parm.type = "psi")

##compute model-averaged estimate of Julian Day squared on detectability
modavg(cand.set = Cand.mods, modnames = Modnames,
       parm = "I(JulianDate^2)", parm.type = "detect")

## End(Not run)

##example of model-averaged estimate of area from distance model
##this is a bit longer
## Not run:
data(linetran) #example modified from ?distsamp

ltUMF <- with(linetran, {
  unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
                  siteCovs = data.frame(Length, area, habitat),
                  dist.breaks = c(0, 5, 10, 15, 20),
                  tlength = linetran$Length * 1000, survey = "line", unitsIn = "m")
})

## Half-normal detection function. Density output (log scale). No covariates.
fm1 <- distsamp(~ 1 ~ 1, ltUMF)

## Halfnormal. Covariates affecting both density and detection.
fm2 <- distsamp(~ area + habitat ~ area + habitat, ltUMF)

## Hazard function. Covariates affecting both density and detection.
fm3 <- distsamp(~ habitat ~ area + habitat, ltUMF, keyfun="hazard")

##assemble model list
Cands <- list(fm1, fm2, fm3)
Modnames <- paste("mod", 1:length(Cands), sep = "")

```

```
##model-average estimate of area on abundance
modavg(cand.set = Cands, modnames = Modnames, parm = "area", parm.type = "lambda")
detach(package:unmarked)

## End(Not run)
```

modavg.utility

Various Utility Functions

Description

reverse.parm and reverse.exclude reverse the order of variables in an interaction term.

formatCands creates new classes for lists containing candidate models.

formulaShort prints a succinct formula from an unmarkedFit object.

Usage

```
reverse.parm(parm)
reverse.exclude(exclude)
formatCands(cand.set)
formulaShort(mod, unmarked.type = NULL)
```

Arguments

parm	a parameter to be model-averaged, enclosed between quotes, as it appears in the output of some models.
exclude	a list of interaction or polynomial terms appearing in some models, as they would appear in the call to the model function (i.e., A*B, A:B). Models containing elements from the list will be excluded to obtain a model-averaged estimate.
cand.set	a list storing each of the models in the candidate model set.
mod	an object storing the result of an unmarkedFit.
unmarked.type	a character string specifying the type of parameter in an unmarkedFit for which a formula is requested. This argument uses the character string for each parameter group as defined by unmarked.

Details

These utility functions are used internally by aictab, modavg, and other related functions.

reverse.parm and reverse.exclude enable the user to specify differently interaction terms (e.g., A:B, B:A) across models for model averaging. These functions have been added to avoid problems when users are not consistent in the specification of interaction terms across models.

formatCands creates new classes for the list of candidate models based on the contents of the list. These new classes are used for method dispatch.

formulaShort is used by anovaOD.

Value

`reverse.parm` returns all possible combinations of an interaction term to identify models that include the parm of interest and find the corresponding estimate and standard error in the model object.

`reverse.exclude` returns a list of all possible combinations of exclude to identify models that should be excluded when computing a model-averaged estimate.

`formatCands` adds a new class to the list of candidate models based on the classes of the models.

`formulaShort` creates a character string for the formula related to a given parameter type from an `unmarkedFit` object.

Author(s)

Marc J. Mazerolle

See Also

[aictab](#), [anova0D](#), [modavg](#), [modavgShrink](#), [modavgPred](#)

Examples

```
##a main effect
reverse.parm(parm = "Ageyoung") #does not return anything

##an interaction term as it might appear in the output
reverse.parm(parm = "Ageyoung:time") #returns the reverse

##exclude two interaction terms
reverse.exclude(exclude = list("Age*time", "A:B"))
##returns all combinations
reverse.exclude(exclude = list("Age:time", "A*B"))
##returns all combinations

##Mazerolle (2006) frog water loss example
data(dry.frog)

##setup a subset of models of Table 1
Cand.models <- list( )
Cand.models[[1]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2,
  data = dry.frog)
Cand.models[[2]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2 +
  Shade:Substrate, data = dry.frog)
Cand.models[[3]] <- lm(log_Mass_lost ~ cent_Initial_mass +
  Initial_mass2, data = dry.frog)

formatCands(Cand.models)
```

```
## Not run:
require(unmarked)
data(bullfrog)
bfrog <- unmarkedFrameOccu(y = bullfrog[, c("V1", "V2", "V3", "V4")],
                           siteCovs = bullfrog[, 1:2])
fm1 <- occu(~ 1 ~ Reed.presence, data = bfrog)
formulaShort(fm1, unmarked.type = "state")
formulaShort(fm1, unmarked.type = "det")

## End(Not run)
```

modavgCustom

Compute Model-averaged Parameter Estimate from User-supplied Input Based on (Q)AIC(c)

Description

This function model-averages the estimate of a parameter of interest among a set of candidate models, and computes the unconditional standard error and unconditional confidence intervals as described in Buckland et al. (1997) and Burnham and Anderson (2002).

Usage

```
modavgCustom(logL, K, modnames = NULL, estimate, se, second.ord = TRUE,
             nobs = NULL, uncond.se = "revised", conf.level = 0.95,
             c.hat = 1, useBIC = FALSE)
```

Arguments

logL	a vector of log-likelihood values for the models in the candidate model set.
K	a vector containing the number of estimated parameters for each model in the candidate model set.
modnames	a character vector of model names to facilitate the identification of each model in the model selection table. If NULL, the function uses the names in the cand.set list of candidate models. If no names appear in the list, generic names (e.g., Mod1, Mod2) are supplied in the table in the same order as in the list of candidate models.
estimate	a vector of estimates for each of the models in the candidate model set. Estimates can be either beta estimates for a parameter of interest or a single prediction from each model.
se	a vector of standard errors for each of the estimates appearing in the estimate vector.
second.ord	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc). This argument is ignored if useBIC = TRUE.
nobs	the sample size required to compute the AICc, QAICc, BIC, or QBIC.

uncond.se	either, "old", or "revised", specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With uncond.se = "old", computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With uncond.se = "revised", equation 6.12 of Burnham and Anderson (2002) is used. Anderson (2008, p. 111) recommends use of the revised version for the computation of unconditional standard errors and it is now the default.
conf.level	the confidence level $(1 - \alpha)$ requested for the computation of unconditional confidence intervals.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season and dynamic occupancy models (MacKenzie et al. 2002, 2003), <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If <code>c.hat</code> > 1, <code>modavgCustom</code> will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>).
useBIC	logical. If TRUE, the function returns the Bayesian information criterion (BIC) when <code>c.hat</code> = 1 or the quasi-likelihood BIC (QBIC) when <code>c.hat</code> > 1.

Details

`modavgCustom` computes a model-averaged estimate from the vector of parameter estimates specified in `estimate`. Estimates and their associated standard errors must be specified in the same order as the log-likelihood, number of estimated parameters, and model names. Estimates provided may be for a parameter of interest (i.e., beta estimates) or predictions from each model. This function is most useful when model input is imported into R from other software (e.g., Program MARK, PRESENCE) or for model classes that are not yet supported by the other model averaging functions such as `modavg` or `modavgPred`.

Value

`modavgCustom` creates an object of class `modavgCustom` with the following components:

Mod.avg.table	the model selection table
Mod.avg.est	the model-averaged estimate
Uncond.SE	the unconditional standard error for the model-averaged estimate
Conf.level	the confidence level used to compute the confidence interval
Lower.CL	the lower confidence limit
Upper.CL	the upper confidence limit

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Buckland, S. T., Burnham, K. P., Augustin, N. H. (1997) Model selection: an integral part of inference. *Biometrics* **53**, 603–618.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67–118.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICcCustom](#), [aictabCustom](#), [bictabCustom](#), [modavg](#), [modavgIC](#), [modavgShrink](#), [modavgPred](#)

Examples

```
## Not run:
##model averaging parameter estimate (natural average)
##vector with model LL's
LL <- c(-38.8876, -35.1783, -64.8970)

##vector with number of parameters
Ks <- c(7, 9, 4)

##create a vector of names to trace back models in set
Modnames <- c("Cm1", "Cm2", "Cm3")

##vector of beta estimates for a parameter of interest
model.ests <- c(0.0478, 0.0480, 0.0478)

##vector of SE's of beta estimates for a parameter of interest
model.se.ests <- c(0.0028, 0.0028, 0.0034)

##compute model-averaged estimate and unconditional SE based on AICc
modavgCustom(logL = LL, K = Ks, modnames = Modnames,
             estimate = model.ests, se = model.se.ests, nobs = 121)
##compute model-averaged estimate and unconditional SE based on BIC
```

```

modavgCustom(logL = LL, K = Ks, modnames = Modnames,
             estimate = model.ests, se = model.se.ests, nobs = 121,
             useBIC = TRUE)

##model-averaging with shrinkage based on AICc
##set up candidate models
data(min.trap)
Cand.mod <- list( )
##global model
Cand.mod[[1]] <- glm(Num_anura ~ Type + log.Perimeter,
                   family = poisson, offset = log(Effort),
                   data = min.trap)
Cand.mod[[2]] <- glm(Num_anura ~ Type + Num_ranatra, family = poisson,
                   offset = log(Effort), data = min.trap)
Cand.mod[[3]] <- glm(Num_anura ~ log.Perimeter + Num_ranatra,
                   family = poisson, offset = log(Effort), data = min.trap)
Model.names <- c("Type + log.Perimeter", "Type + Num_ranatra",
                "log.Perimeter + Num_ranatra")
##model-averaged estimate with shrinkage (glm model type is already supported)
modavgShrink(cand.set = Cand.mod, modnames = Model.names,
             parm = "log.Perimeter")

##equivalent manual version of model-averaging with shrinkage
##this is especially useful when model classes are not supported
##extract vector of LL
LLs <- sapply(Cand.mod, FUN = function(i) logLik(i)[1])
##extract vector of K
Ks <- sapply(Cand.mod, FUN = function(i) attr(logLik(i), "df"))
##extract betas
betas <- sapply(Cand.mod, FUN = function(i) coef(i)["log.Perimeter"])
##second model does not include log.Perimeter
betas[2] <- 0
##extract SE's
ses <- sapply(Cand.mod, FUN = function(i) sqrt(diag(vcov(i))["log.Perimeter"]))
ses[2] <- 0
##model-averaging with shrinkage based on AICc
modavgCustom(logL = LLs, K = Ks, modnames = Model.names,
             nobs = nrow(min.trap), estimate = betas, se = ses)
##model-averaging with shrinkage based on BIC
modavgCustom(logL = LLs, K = Ks, modnames = Model.names,
             nobs = nrow(min.trap), estimate = betas, se = ses,
             useBIC = TRUE)

## End(Not run)

```

Description

This function model-averages the effect size between two groups defined by a categorical variable based on the entire model set and computes the unconditional standard error and unconditional confidence intervals as described in Buckland et al. (1997) and Burnham and Anderson (2002). This can be particularly useful when dealing with data from an experiment (e.g., ANOVA) and when the focus is to determine the effect of a given factor. This is an information-theoretic alternative to multiple comparisons (e.g., Burnham et al. 2011).

Usage

```
modavgEffect(cand.set, modnames = NULL, newdata, second.ord = TRUE,
             nobs = NULL, uncond.se = "revised", conf.level = 0.95,
             ...)

## S3 method for class 'AICaov.lm'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICglm.lm'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, type = "response", c.hat = 1, gamdisp = NULL,
             ...)

## S3 method for class 'AICgls'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AIClm'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AIClme'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICmer'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, type = "response", ...)

## S3 method for class 'AICglmerMod'
modavgEffect(cand.set, modnames = NULL,
             newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
```

```
        conf.level = 0.95, type = "response", ...)
```

```
## S3 method for class 'AIClmerMod'
modavgEffect(cand.set, modnames = NULL,
             newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AIClmerModLmerTest'
modavgEffect(cand.set, modnames = NULL,
             newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AICnegbin.glm.lm'
modavgEffect(cand.set, modnames = NULL,
             newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, type = "response", ...)
```

```
## S3 method for class 'AICr1m.lm'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AICsurvreg'
modavgEffect(cand.set, modnames = NULL, newdata,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, type = "response", ...)
```

```
## S3 method for class 'AICunmarkedFitOccu'
modavgEffect(cand.set, modnames = NULL,
             newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, type = "response", c.hat = 1,
             parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitColExt'
modavgEffect(cand.set, modnames =
             NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
             "revised", conf.level = 0.95, type = "response",
             c.hat = 1, parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitOccuRN'
modavgEffect(cand.set, modnames =
             NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
             "revised", conf.level = 0.95, type = "response",
             c.hat = 1, parm.type = NULL, ...)
```

```
## S3 method for class 'AICunmarkedFitPCount'
modavgEffect(cand.set, modnames =
             NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
```

```
    "revised", conf.level = 0.95, type = "response",
    c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCO'
modavgEffect(cand.set, modnames = NULL,
  newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, type = "response", c.hat = 1,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDS'
modavgEffect(cand.set, modnames = NULL,
  newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, type = "response", c.hat = 1,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGDS'
modavgEffect(cand.set, modnames = NULL,
  newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, type = "response", c.hat = 1,
  parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuFP'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMPois'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGMM'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGPC'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuTTD'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
```

```

    "revised", conf.level = 0.95, type = "response",
    c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMMO'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDSO'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMS'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMulti'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGOccu'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuComm'
modavgEffect(cand.set, modnames =
  NULL, newdata, second.ord = TRUE, nobs = NULL, uncond.se =
  "revised", conf.level = 0.95, type = "response",
  c.hat = 1, parm.type = NULL, ...)

```

Arguments

cand.set	a list storing each of the models in the candidate model set.
modnames	a character vector of model names to facilitate the identification of each model in the model selection table. If NULL, the function uses the names in the cand.set list of candidate models. If no names appear in the list, generic names (e.g., Mod1, Mod2) are supplied in the table in the same order as in the list of candidate models.
newdata	a data frame with two rows and where the columns correspond to the explana-

tory variables specified in the candidate models. Note that this data set must have the same structure as that of the original data frame for which we want to make predictions, specifically, the same variable type and names that appear in the original data set. Each row of the data set defines one of the two groups compared. The first row in `newdata` defines the first group, whereas the second row defines the second group. The effect size is computed as the prediction in the first row minus the prediction in the second row (first row - second row). Only the column relating to the grouping variable can change value and all others must be held constant for the comparison (see 'Details'). For community occupancy models of `unmarkedFitOccuComm` classes, the `newdata` data frame must include a column specifying the name of the species for which each prediction is requested (see 'Examples' below).

<code>second.ord</code>	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
<code>nobs</code>	this argument allows the specification of a numeric value other than total sample size to compute the AICc (i.e., <code>nobs</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .
<code>uncond.se</code>	either, "old", or "revised", specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With <code>uncond.se = "old"</code> , computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With <code>uncond.se = "revised"</code> , equation 6.12 of Burnham and Anderson (2002) is used. Anderson (2008, p. 111) recommends use of the revised version for the computation of unconditional standard errors and it is now the default. Note that versions of package <code>AICcmodavg < 1.04</code> used the old method to compute unconditional standard errors.
<code>conf.level</code>	the confidence level ($1 - \alpha$) requested for the computation of unconditional confidence intervals. To obtain confidence intervals corrected for multiple comparisons between pairs of treatments, it is possible to adjust the α level according to various strategies such as the Bonferroni correction (Dunn 1961).
<code>type</code>	the scale of prediction requested, one of "response" or "link" (only relevant for <code>glm</code> , <code>mer</code> , and <code>unmarkedFit</code> classes). Note that the value "terms" is not defined for <code>modavgEffect</code> .
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season and dynamic occupancy models (MacKenzie et al. 2002, 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If <code>c.hat > 1</code> , <code>modavgEffect</code> will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>). This option is not supported for generalized linear mixed models of the <code>mer</code> class.

gamdisp	if gamma GLM is used, the dispersion parameter should be specified here to apply the same value to each model.
parm.type	this argument specifies the parameter type on which the effect size will be computed and is only relevant for models of unmarkedFit classes. The character strings supported vary with the type of model fitted. For unmarkedFitOccu, unmarkedFitOccuMulti, and unmarkedFitOccuComm objects, either psi or detect can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For unmarkedFitColExt objects, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the initial year, colonization, extinction, and detectability, respectively. For unmarkedFitOccuTTD objects, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the initial year, colonization, extinction, and time-to-detection (lambda rate parameter), respectively. For unmarkedFitOccuFP objects, one can specify psi, detect, falsepos, and certain, for occupancy, detectability, probability of assigning false-positives, and probability detections are certain, respectively. For unmarkedFitOccuRN objects, either lambda or detect can be entered for abundance and detectability parameters, respectively. For unmarkedFitPCount and unmarkedFitMPois objects, lambda or detect denote parameters on abundance and detectability, respectively. For unmarkedFitPCO, unmarkedFitMMO, and unmarkedFitDSO objects, one can enter lambda, gamma, omega, iota, or detect, to specify parameters on abundance, recruitment, apparent survival, immigration, and detectability, respectively. For unmarkedFitDS objects, lambda and detect are supported. For unmarkedFitGDS, lambda, phi, and detect denote abundance, availability, and detection probability, respectively. For unmarkedFitGMM and unmarkedFitGPC objects, lambda, phi, and detect denote abundance, availability, and detectability, respectively. For unmarkedFitOccuMS objects, psi, phi, and detect denote occupancy, transition, and detection probability, respectively. For unmarkedFitGOccu objects, possible values are psi, phi, or detect, denoting occupancy, availability, and detection probabilities, respectively.
...	additional arguments passed to the function.

Details

The strategy used here to compute effect sizes is to work from the newdata object to create two predictions from a given model and compute the differences and standard errors between both values. This step is executed for each model in the candidate model set, to obtain a model-averaged estimate of the effect size and unconditional standard error. As a result, the newdata argument is restricted to two rows, each for a given prediction. To specify each group, the values entered in the column for each explanatory variable can be identical, except for the grouping variable. In such a case, the function will identify the variable and the assign group names based on the values of the variable. If more than a single variable has different values in its respective column, the function will print generic names in the output to identify the two groups. A sensible choice of value for the explanatory variables to be held constant is the average of the variable.

Model-averaging effect sizes is most useful in true experiments (e.g., ANOVA-type designs), where one wants to obtain the best estimate of effect size given the support of each candidate model. This can be considered as an information-theoretic analog of traditional multiple comparisons, except

that the information contained in the entire model set is used instead of being restricted to a single model. See 'Examples' below for applications.

modavgEffect calls the appropriate method depending on the class of objects in the list. The current classes supported include aov, glm, gls, lm, lme, mer, glmerMod, lmerMod, lmerModLmerTest, rlm, survreg, as well as models of unmarkedFitOccu, unmarkedFitColExt, unmarkedFitOccuFP, unmarkedFitOccuRN, unmarkedFitOccuTTD, unmarkedFitPCount, unmarkedFitPCO, unmarkedFitDS, unmarkedFitDSO, unmarkedFitGDS, unmarkedFitMPois, unmarkedFitGMM, unmarkedFitMMO, unmarkedFitGPC, unmarkedFitOccuMS, unmarkedFitOccuMulti, unmarkedFitGOccu, and unmarkedFitOccuComm classes.

Value

The result is an object of class modavgEffect with the following components:

Group.variable	the grouping variable defining the two groups compared.
Group1	the first group considered in the comparison.
Group2	the second group considered in the comparison.
Type	the scale on which the model-averaged effect size was computed (e.g., response or link).
Mod.avg.table	the full model selection table including the entire set of candidate models.
Mod.avg.eff	the model-averaged effect size based on the entire candidate model set.
Uncond.SE	the unconditional standard error for the model-averaged effect size.
Conf.level	the confidence level used to compute the confidence interval.
Lower.CL	the lower confidence limit.
Upper.CL	the upper confidence limit.
Matrix.output	a matrix containing the model-averaged effect size, the unconditional standard error, and the lower and upper confidence limits.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Buckland, S. T., Burnham, K. P., Augustin, N. H. (1997) Model selection: an integral part of inference. *Biometrics* **53**, 603–618.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Burnham, K. P., Anderson, D. R., Huyvaert, K. P. (2011) AIC model selection and multimodel inference in behavioral ecology: some background, observations and comparisons. *Behavioral Ecology and Sociobiology* **65**, 23–25.

Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.

Dunn, O. J. (1961) Multiple comparisons among means. *Journal of the American Statistical Association* **56**, 52–64.

MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.

MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.

Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.

Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICc](#), [aictab](#), [c_hat](#), [confset](#), [evidence](#), [importance](#), [modavgShrink](#), [modavgPred](#)

Examples

```
##heights (cm) of plants grown under two fertilizers, Ex. 9.5 from
##Zar (1984): Biostatistical Analysis. Prentice Hall: New Jersey.
heights <- data.frame(Height = c(48.2, 54.6, 58.3, 47.8, 51.4, 52.0,
                               55.2, 49.1, 49.9, 52.6, 52.3, 57.4, 55.6, 53.2,
                               61.3, 58.0, 59.8, 54.8),
                    Fertilizer = c(rep("old", 10), rep("new", 8)))

##run linear model hypothesizing an effect of fertilizer
m1 <- lm(Height ~ Fertilizer, data = heights)

##run null model (no effect of fertilizer)
m0 <- lm(Height ~ 1, data = heights)

##assemble models in list
Cands <- list(m1, m0)
Modnames <- c("Fert", "null")

##compute model selection table to compare
##both hypotheses
aictab(cand.set = Cands, modnames = Modnames)
##note that model with fertilizer effect is much better supported
##than the null

##compute model-averaged effect sizes: one model hypothesizes a
##difference of 0, whereas the other assumes a difference

##prepare newdata object from which differences between groups
##will be computed
##the first row of the newdata data.frame relates to the first group,
```

```

##whereas the second row corresponds to the second group
pred.data <- data.frame(Fertilizer = c("new", "old"))

##compute best estimate of effect size accounting for model selection
##uncertainty
modavgEffect(cand.set = Cands, modnames = Modnames,
             newdata = pred.data)

##classical one-way ANOVA type-design
## Not run:
##generate data for two groups and control
set.seed(seed = 15)
y <- round(c(rnorm(n = 15, mean = 10, sd = 5),
            rnorm(n = 15, mean = 15, sd = 5),
            rnorm(n = 15, mean = 12, sd = 5)), digits = 2)
##groups
group <- c(rep("cont", 15), rep("trt1", 15), rep("trt2", 15))

##combine in data set
aov.data <- data.frame(Y = y, Group = group)
rm(y, group)

##run model with group effect
lm.eff <- lm(Y ~ Group, data = aov.data)
##null model
lm.0 <- lm(Y ~ 1, data = aov.data)

##compare both models
Cands <- list(lm.eff, lm.0)
Mods <- c("group effect", "no group effect")
aictab(cand.set = Cands, modnames = Mods)
##model with group effect has most of the weight

##compute model-averaged effect sizes
##trt1 - control
modavgEffect(cand.set = Cands, modnames = Modnames,
             newdata = data.frame(Group = c("trt1", "cont")))
##trt1 differs from cont

##trt2 - control
modavgEffect(cand.set = Cands, modnames = Modnames,
             newdata = data.frame(Group = c("trt2", "cont")))
##trt2 does not differ from cont

## End(Not run)

##two-way ANOVA type design, Ex. 13.1 (Zar 1984) of plasma calcium
##concentration (mg/100 ml) in birds as a function of sex and hormone
##treatment
## Not run:
birds <- data.frame(Ca = c(16.87, 16.18, 17.12, 16.83, 17.19, 15.86,

```

```

14.92, 15.63, 15.24, 14.8, 19.07, 18.77, 17.63,
16.99, 18.04, 17.2, 17.64, 17.89, 16.78, 16.92,
32.45, 28.71, 34.65, 28.79, 24.46, 30.54, 32.41,
28.97, 28.46, 29.65),
Sex = c("M", "M", "M", "M", "M", "F", "F", "F", "F",
"F", "M", "M", "M", "M", "M", "F", "F", "F", "F",
"F", "M", "M", "M", "M", "M", "F", "F", "F", "F",
"F"),
Hormone = as.factor(c(1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3)))

##candidate models
##interactive effects
m.inter <- lm(Ca ~ Sex + Hormone + Sex:Hormone, data = birds)

##additive effects
m.add <- lm(Ca ~ Sex + Hormone, data = birds)

##Sex only
m.sex <- lm(Ca ~ Sex, data = birds)

##Hormone only
m.horm <- lm(Ca ~ Hormone, data = birds)

##null
m.0 <- lm(Ca ~ 1, data = birds)

##model selection
Cands <- list(m.inter, m.add, m.sex, m.horm, m.0)
Mods <- c("interaction", "additive", "sex only", "horm only", "null")
aictab(Cands, Mods)
##there is some support for a hormone only treatment, but also for
##additive effects

##compute model-averaged effects of sex, and set the other variable
##to a constant value
##M - F
sex.data <- data.frame(Sex = c("M", "F"), Hormone = c("1", "1"))
modavgEffect(Cands, Mods, newdata = sex.data)
##no support for a sex main effect

##hormone 1 - 3, but set Sex to a constant value
horm1.data <- data.frame(Sex = c("M", "M"), Hormone = c("1", "3"))
modavgEffect(Cands, Mods, newdata = horm1.data)

##hormone 2 - 3, but set Sex to a constant value
horm2.data <- data.frame(Sex = c("M", "M"), Hormone = c("2", "3"))
modavgEffect(Cands, Mods, newdata = horm2.data)

## End(Not run)

```

```

##Poisson regression with anuran larvae example from Mazerolle (2006)
## Not run:
data(min.trap)
##assign "UPLAND" as the reference level as in Mazerolle (2006)
min.trap$type <- relevel(min.trap$type, ref = "UPLAND")

##set up candidate models
Cand.mod <- list( )
##global model
Cand.mod[[1]] <- glm(Num_anura ~ Type + log.Perimeter,
                    family = poisson, offset = log(Effort),
                    data = min.trap)
Cand.mod[[2]] <- glm(Num_anura ~ log.Perimeter, family = poisson,
                    offset = log(Effort), data = min.trap)
Cand.mod[[3]] <- glm(Num_anura ~ Type, family = poisson,
                    offset = log(Effort), data = min.trap)
Cand.mod[[4]] <- glm(Num_anura ~ 1, family = poisson,
                    offset = log(Effort), data = min.trap)

##check c-hat for global model
vif.hat <- c_hat(Cand.mod[[1]]) #uses Pearson's chi-square/df

##assign names to each model
Modnames <- c("type + logperim", "type", "logperim", "intercept only")

##compute model-averaged estimate of difference between abundance at bog
##pond and upland pond
##create newdata object to make predictions
pred.data <- data.frame(Type = c("BOG", "UPLAND"),
                        log.Perimeter = mean(min.trap$log.Perimeter),
                        Effort = mean(min.trap$Effort))
modavgEffect(Cand.mod, Modnames, newdata = pred.data, c.hat = vif.hat,
             type = "response")
##little suport for a pond type effect

## End(Not run)

##mixed linear model example from ?nlme
## Not run:
library(nlme)
Cand.models <- list( )
Cand.models[[1]] <- lme(distance ~ age, data = Orthodont, method="ML")
Cand.models[[2]] <- lme(distance ~ age + Sex, data = Orthodont,
                        random = ~ 1, method="ML")
Cand.models[[3]] <- lme(distance ~ 1, data = Orthodont, random = ~ 1,
                        method="ML")
Cand.models[[4]] <- lme(distance ~ Sex, data = Orthodont, random = ~ 1,
                        method="ML")

Modnames <- c("age", "age + sex", "null", "sex")

data.other <- data.frame(age = mean(Orthodont$age),

```

```

      Sex = factor(c("Male", "Female")))
modavgEffect(cand.set = Cand.models, modnames = Modnames,
             newdata = data.other, conf.level = 0.95, second.ord = TRUE,
             nobs = NULL, uncond.se = "revised")
detach(package:nlme)

## End(Not run)

##site occupancy analysis example
## Not run:
library(unmarked)
##single season model
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
##create a bogus site group
site.group <- c(rep(1, times = nrow(pfer.bin)/2), rep(0, nrow(pfer.bin)/2))

## add some fake covariates for illustration
siteCovs(pferUMF) <- data.frame(site.group, sitevar1 =
                               rnorm(numSites(pferUMF)),
                               sitevar2 = runif(numSites(pferUMF)))

## observation covariates are in site-major, observation-minor order
obsCovs(pferUMF) <- data.frame(obsvar1 =
                               rnorm(numSites(pferUMF) * obsNum(pferUMF)))

fm1 <- occu(~ obsvar1 ~ site.group, pferUMF)
fm2 <- occu(~ obsvar1 ~ 1, pferUMF)

Cand.mods <- list(fm1, fm2)
Modnames <- c("fm1", "fm2")

##model selection table
aictab(cand.set = Cand.mods, modnames = Modnames, second.ord = TRUE)

##model-averaged effect sizes comparing site.group 1 - site.group 0
newer.dat <- data.frame(site.group = c(0, 1))

modavgEffect(cand.set = Cand.mods, modnames = Modnames, type = "response",
             second.ord = TRUE, newdata = newer.dat, parm.type = "psi")
##no support for an effect of site group

## End(Not run)

##single season N-mixture models
## Not run:
data(mallard)
##this variable was created to illustrate the use of modavgEffect
##with detection variables
mallard.site$site.group <- c(rep(1, 119), rep(0, 120))
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,

```

```

                                obsCovs = mallard.obs)
siteCovs(mallardUMF)
tmp.covs <- obsCovs(mallardUMF)
obsCovs(mallardUMF)$date2 <- tmp.covs$date^2
(fm.mall <- pcount(~ site.group ~ length + elev + forest, mallardUMF, K=30))
(fm.mallb <- pcount(~ 1 ~ length + elev + forest, mallardUMF, K=30))

Cands <- list(fm.mall, fm.mallb)
Modnames <- c("one", "null")

##model averaged effect size of site.group 1 - site.group 0 on response
##scale (point estimate)
modavgEffect(Cands, Modnames, newdata = data.frame(site.group = c(0, 1)),
             parm.type = "detect", type = "response")

##model averaged effect size of site.group 1 - site.group 0 on link
##scale (here, logit link)
modavgEffect(Cands, Modnames, newdata = data.frame(site.group = c(0, 1)),
             parm.type = "detect", type = "link")

detach(package:unmarked)

## End(Not run)

##example of model-averaged occupancy in community occupancy model
## Not run:
##example adapted from ?occuComm in unmarked package
##simulate multispecies data
M <- 300
J <- 5
S <- 30

set.seed(123)
x <- rnorm(M)

mu_0 <- 0
sd_0 <- 0.4
beta0 <- rnorm(S, mu_0, sd_0)

mu_x <- 1
sd_x <- 0.3
beta_x <- rnorm(S, mu_x, sd_x)

mu_a <- 0
sd_a <- 0.2
alpha0 <- rnorm(S, mu_a, sd_a)

ylist <- list()
z <- matrix(NA, M, S)
for (s in 1:S){
  psi <- plogis(beta0[s] + beta_x[s] * x)
  z[,s] <- rbinom(M, 1, psi)
}

```

```

    p <- plogis(alpha0[s])

    y <- matrix(NA, M, J)
    for (m in 1:M){
      y[m,] <- rbinom(J, 1, p * z[m,s])
    }
    ylist[[s]] <- y
  }
  ##add species names
  names(ylist) <- paste0("sp", sprintf("%02d", 1:length(ylist)))

  ##create unmarkedFrame
  sc <- data.frame(x=x, a=factor(sample(letters[1:5], M, replace=TRUE)))
  umf <- unmarkedFrameOccuComm(ylist, siteCovs=sc)

  ##fit model with one covariate on occupancy
  ##structure of the model psi ~ (x | species), p ~ (1 | species)
  fit1 <- occuComm(~ 1 ~ x, umf)

  ##fit alternative model without covariate
  fit2 <- occuComm(~ 1 ~ 1, umf)

  ##assemble models in list
  occModels <- list("withX" = fit1, "withoutX" = fit2)

  ##check names of species in unmarkedFrameOccuComm object
  checkNames <- names(umf@ylist)
  checkNames

  ##create data frame for predictions on sp06
  newData <- data.frame(x = c(min(sc$x), max(sc$x)),
                       species = factor("sp06", levels = checkNames))
  resultsPred <- modavgPred(occModels, newData = newData,
                           parm.type = "psi")

  resultsPred

  ## End(Not run)

```

 modavgIC

Compute Model-averaged Parameter Estimate from User-supplied Information Criterion

Description

This function model-averages the estimate of a parameter of interest among a set of candidate models, and computes the unconditional standard error and unconditional confidence intervals as described in Buckland et al. (1997) and Burnham and Anderson (2002). Computations are based on the values of the information criterion supplied manually by the user.

Usage

```
modavgIC(ic, K, modnames = NULL, estimate, se, uncond.se = "revised",
         conf.level = 0.95, ic.name = NULL)
```

Arguments

<code>ic</code>	a vector of information criterion values for each model in the candidate model set.
<code>K</code>	a vector containing the number of estimated parameters for each model in the candidate model set.
<code>modnames</code>	a character vector of model names to identify each model in the model selection table. If <code>NULL</code> , generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as the information criterion values.
<code>estimate</code>	a vector of estimates for each of the models in the candidate model set. Estimates can be either beta estimates for a parameter of interest or a single prediction from each model.
<code>se</code>	a vector of standard errors for each of the estimates appearing in the estimate vector.
<code>uncond.se</code>	either, "old", or "revised", specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With <code>uncond.se = "old"</code> , computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With <code>uncond.se = "revised"</code> , equation 6.12 of Burnham and Anderson (2002) is used. Anderson (2008, p. 111) recommends use of the revised version for the computation of unconditional standard errors and it is now the default.
<code>conf.level</code>	the confidence level ($1 - \alpha$) requested for the computation of unconditional confidence intervals.
<code>ic.name</code>	a character string denoting the name of the information criterion input by the user. This character string will appear in certain column labels of the model selection table.

Details

`modavgIC` computes a model-averaged estimate from the vector of parameter estimates specified in `estimate`. Estimates and their associated standard errors must be specified in the same order as the values of the information criterion, the number of estimated parameters, and the model names. Estimates provided may be for a parameter of interest (i.e., beta estimates) or predictions from each model. This function is most useful for information criterion other than AIC, AICc, QAIC, and QAICc (e.g., WAIC: Watanabe 2010) or for classes not supported by `modavg`, `modavgCustom`, or `modavgPred`.

Value

`modavgIC` creates an object of class `modavgIC` with the following components:

`Mod.avg.table` the model selection table.

Mod.avg.est	the model-averaged estimate.
Uncond.SE	the unconditional standard error for the model-averaged estimate.
Conf.level	the confidence level used to compute the confidence interval.
Lower.CL	the lower confidence limit.
Upper.CL	the upper confidence limit.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Buckland, S. T., Burnham, K. P., Augustin, N. H. (1997) Model selection: an integral part of inference. *Biometrics* **53**, 603–618.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Watanabe, S. (2010) Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research* **11**, 3571–3594.

See Also

[aictabCustom](#), [ictab](#), [modavg](#), [modavgCustom](#), [modavgShrink](#), [modavgPred](#)

Examples

```
## Not run:
##model averaging parameter estimate based on WAIC
##create a vector of names to trace back models in set
Modnames <- c("global model", "interactive model",
             "additive model", "invertpred model")

##WAIC values
waic <- c(105.74, 107.36, 108.24, 100.57)
##number of effective parameters
effK <- c(7.45, 5.61, 6.14, 6.05)

##vector of predictions
Preds <- c(0.106, 0.137, 0.067, 0.050)
##vector of SE's for prediction
Ses <- c(0.128, 0.159, 0.054, 0.039)

##compute model-averaged estimate and unconditional SE based on WAIC
modavgIC(ic = waic, K = effK, modnames = Modnames,
        estimate = Preds, se = Ses,
        ic.name = "WAIC")

## End(Not run)
```

 modavgPred

Compute Model-averaged Predictions

Description

This function computes the model-averaged predictions, unconditional standard errors, and confidence intervals based on the entire candidate model set. The function is currently implemented for `glm`, `gls`, `lm`, `lme`, `mer`, `merMod`, `lmerMod`, `lmerTest`, `negbin`, `rlm`, `survreg` object classes that are stored in a list as well as various models of `unmarkedFit` classes.

Usage

```
modavgPred(cand.set, modnames = NULL, newdata, second.ord = TRUE,
           nobs = NULL, uncond.se = "revised", conf.level = 0.95, ...)
```

```
## S3 method for class 'AICaov.lm'
```

```
modavgPred(cand.set, modnames = NULL, newdata,
           second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, ...)
```

```
## S3 method for class 'AICglm.lm'
```

```
modavgPred(cand.set, modnames = NULL, newdata,
           second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           gamdisp = NULL, ...)
```

```
## S3 method for class 'AIClm'
```

```
modavgPred(cand.set, modnames = NULL, newdata,
           second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, ...)
```

```
## S3 method for class 'AICgls'
```

```
modavgPred(cand.set, modnames = NULL, newdata,
           second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, ...)
```

```
## S3 method for class 'AIClme'
```

```
modavgPred(cand.set, modnames = NULL, newdata,
           second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, ...)
```

```
## S3 method for class 'AICmer'
```

```
modavgPred(cand.set, modnames = NULL, newdata,
           second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1, ...)
```

```
## S3 method for class 'AICglmerMod'
```

```
modavgPred(cand.set, modnames = NULL, newdata,
            second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, type = "response", c.hat = 1, ...)

## S3 method for class 'AIClmerMod'
modavgPred(cand.set, modnames = NULL, newdata,
            second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, ...)

## S3 method for class 'AIClmerModLmerTest'
modavgPred(cand.set, modnames = NULL, newdata,
            second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, ...)

## S3 method for class 'AICnegbin.glm.lm'
modavgPred(cand.set, modnames = NULL,
            newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, type = "response", ...)

## S3 method for class 'AICr1m.lm'
modavgPred(cand.set, modnames = NULL, newdata,
            second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, ...)

## S3 method for class 'AICsurvreg'
modavgPred(cand.set, modnames = NULL, newdata,
            second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, type = "response", ...)

## S3 method for class 'AICunmarkedFitOccu'
modavgPred(cand.set, modnames = NULL,
            newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, type = "response", c.hat = 1,
            parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitColExt'
modavgPred(cand.set, modnames = NULL,
            newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, type = "response", c.hat = 1,
            parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuRN'
modavgPred(cand.set, modnames = NULL,
            newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
            conf.level = 0.95, type = "response", c.hat = 1,
            parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCount'
```

```
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCO'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDS'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGDS'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuFP'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMPois'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGMM'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGPC'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuTTD'
```

```

modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMMO'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDSO'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMS'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMulti'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGOccu'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuComm'
modavgPred(cand.set, modnames = NULL,
           newdata, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
           conf.level = 0.95, type = "response", c.hat = 1,
           parm.type = NULL, ...)

```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table. If <code>NULL</code> , the function uses the names in the <code>cand.set</code> list of candidate models. If no names appear in the list, generic names (e.g., <code>Mod1</code> , <code>Mod2</code>) are supplied in the table in the same order as in the list of candidate

	models.
<code>newdata</code>	a data frame with the same structure as that of the original data frame for which we want to make predictions. For community occupancy models of <code>unmarkedFitOccuComm</code> classes, the <code>newdata</code> data frame must include a column specifying the name of the species for which each prediction is requested (see 'Examples' below).
<code>second.ord</code>	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., <code>nobs</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>unmarkedFit</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .
<code>uncond.se</code>	either, <code>old</code> , or <code>revised</code> , specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With <code>uncond.se = "old"</code> , computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With <code>uncond.se = "revised"</code> , equation 6.12 of Burnham and Anderson (2002) is used. Anderson (2008, p. 111) recommends use of the revised version for the computation of unconditional standard errors and it is now the default. Note that versions of package <code>AICcmodavg < 1.04</code> used the old method to compute unconditional standard errors.
<code>conf.level</code>	the confidence level ($1 - \alpha$) requested for the computation of unconditional confidence intervals.
<code>type</code>	the scale of prediction requested, one of <code>response</code> or <code>link</code> . The latter is only relevant for <code>glm</code> , <code>mer</code> , and <code>unmarkedFit</code> classes. Note that the value <code>terms</code> is not defined for <code>modavgPred</code> .
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with <code>trials > 1</code> (i.e., <code>success/trial</code> or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, single-season and dynamic occupancy models (MacKenzie et al. 2002, 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If <code>c.hat > 1</code> , <code>modavgPred</code> will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>). This option is not supported for generalized linear mixed models of the <code>mer</code> class.
<code>gamdisp</code>	the value of the gamma dispersion parameter.
<code>parm.type</code>	this argument specifies the parameter type on which the predictions will be computed and is only relevant for models of <code>unmarkedFit</code> classes. The character strings supported vary with the type of model fitted. For <code>unmarkedFitOccu</code> , <code>unmarkedFitOccuMulti</code> , and <code>unmarkedFitOccuComm</code> objects, either <code>psi</code> or <code>detect</code> can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For <code>unmarkedFitColExt</code> objects, possible values are <code>psi</code> , <code>gamma</code> , <code>epsilon</code> , and <code>detect</code> , for parameters on occupancy in the initial year,

colonization, extinction, and detectability, respectively. For `unmarkedFitOccuTTD` objects, possible values are `psi`, `gamma`, `epsilon`, and `detect`, for parameters on occupancy in the initial year, colonization, extinction, and time-to-detection (lambda rate parameter), respectively. For `unmarkedFitOccuFP` objects, one can specify `psi`, `detect`, `falsepos`, and `certain`, for occupancy, detectability, probability of assigning false-positives, and probability detections are certain, respectively. For `unmarkedFitOccuRN` objects, either `lambda` or `detect` can be entered for abundance and detectability parameters, respectively. For `unmarkedFitPCount` and `unmarkedFitMPois` objects, `lambda` or `detect` denote parameters on abundance and detectability, respectively. For `unmarkedFitPCO`, `unmarkedFitMMO`, and `unmarkedFitDSO` objects, one can enter `lambda`, `gamma`, `omega`, `iota`, or `detect`, to specify parameters on abundance, recruitment, apparent survival, immigration, and detectability, respectively. For `unmarkedFitDS` objects, `lambda` and `detect` are supported. For `unmarkedFitGDS`, `lambda`, `phi`, and `detect` denote abundance, availability, and detection probability, respectively. For `unmarkedFitGMM` and `unmarkedFitGPC` objects, `lambda`, `phi`, and `detect` denote abundance, availability, and detectability, respectively. For `unmarkedFitOccuMS` objects, `psi`, `phi`, and `detect` denote occupancy, transition, and detection probability, respectively. For `unmarkedFitGOccu` objects, possible values are `psi`, `phi`, or `detect`, denoting occupancy, availability, and detection probabilities, respectively.

... additional arguments passed to the function.

Details

The candidate models must be stored in a list. Note that a data frame from which to make predictions must be supplied with the `newdata` argument and that all variables appearing in the model set must appear in this data frame. Variables must be of the same type as in the original analysis (e.g., factor, numeric).

One can compute unconditional confidence intervals around the predictions from the elements returned by `modavgPred`. The classic computation based on asymptotic normality of the estimator is appropriate to estimate confidence intervals on the linear predictor (i.e., link scale). For predictions of some types of response variables such as counts or binary variables, the normal approximation may be inappropriate. In such cases, it is often better to compute the confidence intervals on the linear predictor scale and then back-transform the limits to the scale of the response variable. These are the confidence intervals returned by `modavgPred`. Burnham et al. (1987), Burnham and Anderson (2002, p. 164), and Williams et al. (2002) suggest alternative methods of computing confidence intervals for small degrees of freedom with profile likelihood intervals or bootstrapping, but these approaches are not yet implemented in `modavgPred`.

Value

`modavgPred` returns an object of class `modavgPred` with the following components:

<code>type</code>	the scale of predicted values (response or link) for <code>glm</code> , <code>mer</code> , <code>merMod</code> , or <code>unmarkedFit</code> classes.
<code>mod.avg.pred</code>	the model-averaged prediction over the entire candidate model set.
<code>uncond.se</code>	the unconditional standard error of each model-averaged prediction.

conf.level	the confidence level used to compute the confidence interval.
lower.CL	the lower confidence limit.
upper.CL	the upper confidence limit.
matrix.output	a matrix with rows consisting of the model-averaged predictions, the unconditional standard errors, and the confidence limits.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Burnham, K. P., Anderson, D. R., White, G. C., Brownie, C., Pollock, K. H. (1987) Design and analysis methods for fish survival experiments based on release-recapture. *American Fisheries Society Monographs* **5**, 1–437.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.
- Williams, B. K., Nichols, J. D., Conroy, M. J. (2002) *Analysis and Management of Animal Populations*. Academic Press: New York.

See Also

[AICc](#), [aictab](#), [importance](#), [c_hat](#), [confset](#), [evidence](#), [modavg](#), [modavgCustom](#), [modavgEffect](#), [modavgShrink](#), [predict](#), [predictSE](#)

Examples

```
##example from subset of models in Table 1 in Mazerolle (2006)
data(dry.frog)

Cand.models <- list( )
Cand.models[[1]] <- lm(log_Mass_lost ~ Shade + Substrate +
                      cent_Initial_mass + Initial_mass2,
                      data = dry.frog)
Cand.models[[2]] <- lm(log_Mass_lost ~ Shade + Substrate +
```

```

      cent_Initial_mass + Initial_mass2 +
      Shade:Substrate, data = dry.frog)
Cand.models[[3]] <- lm(log_Mass_lost ~ cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[4]] <- lm(log_Mass_lost ~ Shade + cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[4]] <- lm(log_Mass_lost ~ Shade + cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Cand.models[[5]] <- lm(log_Mass_lost ~ Substrate + cent_Initial_mass +
  Initial_mass2, data = dry.frog)

##setup model names
Modnames <- paste("mod", 1:length(Cand.models), sep = "")

##compute model-averaged value and unconditional SE of predicted log of
##mass lost for frogs of average mass in shade for each substrate type

##first create data set to use for predictions
new.dat <- data.frame(Shade = c(1, 1, 1),
  cent_Initial_mass = c(0, 0, 0),
  Initial_mass2 = c(0, 0, 0),
  Substrate = c("SOIL", "SPHAGNUM", "PEAT"))

##compare unconditional SE's using both methods
modavgPred(cand.set = Cand.models, modnames = Modnames,
  newdata = new.dat, type = "response", uncond.se = "old")
modavgPred(cand.set = Cand.models, modnames = Modnames,
  newdata = new.dat, type = "response", uncond.se = "revised")
##round to 4 digits after decimal point
print(modavgPred(cand.set = Cand.models, modnames = Modnames,
  newdata = new.dat, type = "response",
  uncond.se = "revised"), digits = 4)

##Gamma glm
## Not run:
##clotting data example from 'gamma.shape' in MASS package of
##Venables and Ripley (2002, Modern applied statistics with
##S. Springer-Verlag: New York.)
clotting <- data.frame(u = c(5, 10, 15, 20, 30, 40, 60, 80, 100),
  lot1 = c(118, 58, 42, 35, 27, 25, 21, 19, 18),
  lot2 = c(69, 35, 26, 21, 18, 16, 13, 12, 12))
clot1 <- glm(lot1 ~ log(u), data = clotting, family = Gamma)

require(MASS)
gamma.dispersion(clot1) #dispersion parameter
gamma.shape(clot1) #reciprocal of dispersion parameter ==
##shape parameter
summary(clot1, dispersion = gamma.dispersion(clot1)) #better

##create list with models
Cand <- list( )

```

```

Cand[[1]] <- glm(lot1 ~ log(u), data = clotting, family = Gamma)
Cand[[2]] <- glm(lot1 ~ 1, data = clotting, family = Gamma)

##create vector of model names
Modnames <- paste("mod", 1:length(Cand), sep = "")

##compute model-averaged predictions on scale of response variable for
##all observations
modavgPred(cand.set = Cand, modnames = Modnames, newdata = clotting,
           gamdisp = gamma.dispersion(clot1), type = "response")

##compute model-averaged predictions on scale of linear predictor
modavgPred(cand.set = Cand, modnames = Modnames, newdata = clotting,
           gamdisp = gamma.dispersion(clot1), type = "link")

##compute model-averaged predictions on scale of linear predictor
modavgPred(cand.set = Cand, modnames = Modnames, newdata = clotting,
           gamdisp = gamma.dispersion(clot1), type = "terms") #returns an error
##because type = "terms" is not defined for 'modavgPred'

modavgPred(cand.set = Cand, modnames = Modnames, newdata = clotting,
           type = "terms") #returns an error because
##no gamma dispersion parameter was specified (i.e., 'gamdisp' missing)

## End(Not run)

##example of model-averaged predictions from N-mixture model
##each variable appears twice in the models - this is a bit longer
## Not run:
require(unmarked)
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
                                obsCovs = mallard.obs)
##set up models so that each variable on abundance appears twice
fm.mall.one <- pcount(~ ivel + date ~ length + forest, mallardUMF,
                    K = 30)
fm.mall.two <- pcount(~ ivel + date ~ elev + forest, mallardUMF,
                    K = 30)
fm.mall.three <- pcount(~ ivel + date ~ length + elev, mallardUMF,
                      K = 30)
fm.mall.four <- pcount(~ ivel + date ~ 1, mallardUMF, K = 30)

##model list
Cands <- list(fm.mall.one, fm.mall.two, fm.mall.three, fm.mall.four)
Modnames <- c("length + forest", "elev + forest", "length + elev",
            "null")

##compute model-averaged predictions of abundance for values of elev
modavgPred(cand.set = Cands, modnames = Modnames, newdata =
           data.frame(elev = seq(from = -1.4, to = 2.4, by = 0.1),
                     length = 0, forest = 0), parm.type = "lambda",
           type = "response")

```

```

##compute model-averaged predictions of detection for values of ivel
modavgPred(cand.set = Cands, modnames = Modnames, newdata =
            data.frame(ivel = seq(from = -1.75, to = 5.9, by = 0.5),
                        date = 0), parm.type = "detect",
            type = "response")
detach(package:unmarked)

## End(Not run)

##example of model-averaged abundance from distance model
## Not run:
##this is a bit longer
data(linetran) #example from ?distsamp

ltUMF <- with(linetran, {
  unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
                  siteCovs = data.frame(Length, area, habitat),
                  dist.breaks = c(0, 5, 10, 15, 20),
                  tlength = linetran$Length * 1000, survey = "line",
                  unitsIn = "m")
})

## Half-normal detection function. Density output (log scale). No covariates.
fm1 <- distsamp(~ 1 ~ 1, ltUMF)

## Halfnormal. Covariates affecting both density and and detection.
fm2 <- distsamp(~area + habitat ~ habitat, ltUMF)

## Hazard function. Covariates affecting both density and and detection.
fm3 <- distsamp(~area + habitat ~ habitat, ltUMF, keyfun="hazard")

##assemble model list
Cands <- list(fm1, fm2, fm3)
Modnames <- paste("mod", 1:length(Cands), sep = "")

##model-average predictions on abundance
modavgPred(cand.set = Cands, modnames = Modnames, parm.type = "lambda", type = "link",
            newdata = data.frame(area = mean(linetran$area), habitat = c("A", "B")))
detach(package:unmarked)

## End(Not run)

##example of model-averaged occupancy in community occupancy model
## Not run:
##example adapted from ?occuComm in unmarked package
##simulate multispecies data
M <- 300
J <- 5
S <- 30

```

```

set.seed(123)
x <- rnorm(M)

mu_0 <- 0
sd_0 <- 0.4
beta0 <- rnorm(S, mu_0, sd_0)

mu_x <- 1
sd_x <- 0.3
beta_x <- rnorm(S, mu_x, sd_x)

mu_a <- 0
sd_a <- 0.2
alpha0 <- rnorm(S, mu_a, sd_a)

ylist <- list()
z <- matrix(NA, M, S)
for (s in 1:S){
  psi <- plogis(beta0[s] + beta_x[s] * x)
  z[,s] <- rbinom(M, 1, psi)

  p <- plogis(alpha0[s])

  y <- matrix(NA, M, J)
  for (m in 1:M){
    y[m,] <- rbinom(J, 1, p * z[m,s])
  }
  ylist[[s]] <- y
}
##add species names
names(ylist) <- paste0("sp", sprintf("%02d", 1:S))

##create unmarkedFrame
sc <- data.frame(x=x, a=factor(sample(letters[1:5], M, replace=TRUE)))
umf <- unmarkedFrameOccuComm(ylist, siteCovs=sc)

##fit model with one covariate on occupancy
##structure of the model psi ~ (x | species), p ~ (1 | species)
fit1 <- occuComm(~ 1 ~ x, umf)

##fit alternative model without covariate
fit2 <- occuComm(~ 1 ~ 1, umf)

##assemble models in list
occModels <- list("withX" = fit1, "withoutX" = fit2)

##check names of species in unmarkedFrameOccuComm object
checkNames <- names(umf@ylist)
checkNames

##create data frame for predictions on sp06
newData <- data.frame(x = seq(min(sc$x), max(sc$x), length.out = 30),

```

```

      species = factor("sp06", levels = checkNames))
resultsPred <- modavgPred(occModels, newdata = newData,
      parm.type = "psi")
resultsPred
##example with multiple species
newData2 <- expand.grid(x = seq(min(sc$x), max(sc$x), length.out = 30),
      species = factor(c("sp06", "sp10", "sp22"),
      levels = checkNames))
resultsPred2 <- modavgPred(occModels, newdata = newData2,
      parm.type = "psi")
resultsPred2

## End(Not run)

##example using Orthodont data set from Pinheiro and Bates (2000)
## Not run:
require(nlme)

##set up candidate models
m1 <- gls(distance ~ age, correlation = corCompSymm(value = 0.5, form = ~ 1 | Subject),
      data = Orthodont, method = "ML")

m2 <- gls(distance ~ 1, correlation = corCompSymm(value = 0.5, form = ~ 1 | Subject),
      data = Orthodont, method = "ML")

##assemble in list
Cand.models <- list(m1, m2)
##model names
Modnames <- c("age effect", "null model")

##model selection table
aictab(cand.set = Cand.models, modnames = Modnames)

##model-averaged predictions
modavgPred(cand.set = Cand.models, modnames = Modnames, newdata =
data.frame(age = c(8, 10, 12, 14)))
detach(package:nlme)

## End(Not run)

```

modavgShrink

Compute Model-averaged Parameter Estimate with Shrinkage (Multi-model Inference)

Description

This function computes an alternative version of model-averaging parameter estimates that consists in shrinking estimates toward 0 to reduce model selection bias as in Burnham and Anderson (2002, p. 152), Anderson (2008, pp. 130-132) and Lukacs et al. (2010). Specifically, models without the parameter of interest have an estimate and variance of 0. `modavgShrink` also returns unconditional

standard errors and unconditional confidence intervals as described in Buckland et al. (1997) and Burnham and Anderson (2002).

Usage

```

modavgShrink(cand.set, parm, modnames = NULL, second.ord = TRUE,
             nobs = NULL, uncond.se = "revised", conf.level = 0.95,
             ...)
## S3 method for class 'AICaov.lm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICbetareg'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICsclm.clm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICclm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICclmm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICcoxme'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICcoxph'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICglm.lm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, gamdisp = NULL, ...)

## S3 method for class 'AICgls'

```

```
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AICglmmTMB'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, ...)
```

```
## S3 method for class 'AICHurdle'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AIClm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AIClme'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AIClmekin'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AICmer'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AICglmerMod'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AIClmerMod'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AIClmerModLmerTest'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)
```

```
## S3 method for class 'AICmaxlikeFit.lm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, ...)

## S3 method for class 'AICmultinom.nnet'
modavgShrink(cand.set, parm, modnames =
             NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, ...)

## S3 method for class 'AICnegbin.glm.lm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICpolr'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICrlm.lm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICsurvreg'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICvglm'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, ...)

## S3 method for class 'AICzeroinfl'
modavgShrink(cand.set, parm, modnames = NULL,
             second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, ...)

## S3 method for class 'AICunmarkedFitOccu'
modavgShrink(cand.set, parm, modnames =
             NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitColExt'
modavgShrink(cand.set, parm, modnames
```

```
        = NULL, second.ord = TRUE, nobs = NULL, uncond.se =
        "revised", conf.level = 0.95, c.hat = 1, parm.type = NULL,
        ...)

## S3 method for class 'AICunmarkedFitOccuRN'
modavgShrink(cand.set, parm, modnames
             = NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCount'
modavgShrink(cand.set, parm, modnames
             = NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitPCO'
modavgShrink(cand.set, parm, modnames =
             NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDS'
modavgShrink(cand.set, parm, modnames =
             NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGDS'
modavgShrink(cand.set, parm, modnames =
             NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuFP'
modavgShrink(cand.set, parm, modnames
             = NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMPois'
modavgShrink(cand.set, parm, modnames
             = NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGMM'
modavgShrink(cand.set, parm, modnames
             = NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGPC'
modavgShrink(cand.set, parm, modnames
             = NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
             conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)
```

```

## S3 method for class 'AICunmarkedFitOccuMulti'
modavgShrink(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuMS'
modavgShrink(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuTTD'
modavgShrink(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitMMO'
modavgShrink(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitDSO'
modavgShrink(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitGOccu'
modavgShrink(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

## S3 method for class 'AICunmarkedFitOccuComm'
modavgShrink(cand.set, parm, modnames =
  NULL, second.ord = TRUE, nobs = NULL, uncond.se = "revised",
  conf.level = 0.95, c.hat = 1, parm.type = NULL, ...)

```

Arguments

cand.set	a list storing each of the models in the candidate model set.
parm	the parameter of interest, enclosed between quotes, for which a model-averaged estimate is required. For a categorical variable, the label of the estimate must be included as it appears in the output (see 'Details' below).
modnames	a character vector of model names to facilitate the identification of each model in the model selection table. If NULL, the function uses the names in the cand.set list of candidate models. If no names appear in the list, generic names (e.g., Mod1, Mod2) are supplied in the table in the same order as in the list of candidate models.

second.ord	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
nobs	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., nobs defaults to total number of observations). This is relevant only for mixed models or various models of unmarkedFit classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of nobs.
uncond.se	either, "old", or "revised", specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With uncond.se = "old", computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With uncond.se = "revised", equation 6.12 of Burnham and Anderson (2002) is used. Anderson (2008, p. 111) recommends use of the revised version for the computation of unconditional standard errors and it is now the default. Note that versions of package AICcmodavg < 1.04 used the old method to compute unconditional standard errors.
conf.level	the confidence level ($1 - \alpha$) requested for the computation of unconditional confidence intervals.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from c_hat. Note that values of c.hat different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or cbind(success, failure) syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011). If c.hat > 1, modavgShrink will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by sqrt(c.hat)). This option is not supported for generalized linear mixed models of the mer or merMod classes.
gamdisp	if gamma GLM is used, the dispersion parameter should be specified here to apply the same value to each model.
parm.type	this argument specifies the parameter type on which the model-averaged estimate of a predictor will be computed and is only relevant for models of unmarkedFit classes. The character strings supported vary with the type of model fitted. For unmarkedFitOccu, unmarkedFitOccuMulti, and unmarkedFitOccuComm objects, either psi or detect can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For unmarkedFitColExt objects, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the initial year, colonization, extinction, and detectability, respectively. For unmarkedFitOccuTTD objects, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the initial year, colonization, extinction, and time-to-detection (lambda rate parameter), respectively. For unmarkedFitOccuFP objects, one can specify psi, detect, falsepos, and certain, for occupancy, detectability, probability of assigning false-positives, and probability detections are certain, respectively. For unmarkedFitOccuRN objects, either lambda or detect can be entered for abundance and detectability parameters, respectively. For unmarkedFitPCount and unmarkedFitMPois

objects, lambda or detect denote parameters on abundance and detectability, respectively. For unmarkedFitPCO, unmarkedFitMMO, and unmarkedFitDSO objects, one can enter lambda, gamma, omega, iota, or detect, to specify parameters on abundance, recruitment, apparent survival, immigration, and detectability, respectively. For unmarkedFitDS objects, lambda and detect are supported. For unmarkedFitGDS, lambda, phi, and detect denote abundance, availability, and detection probability, respectively. For unmarkedFitGMM and unmarkedFitGPC objects, lambda, phi, and detect denote abundance, availability, and detectability, respectively. For unmarkedFitGOccu objects, possible values are psi, phi, or detect, denoting occupancy, availability, and detection probabilities, respectively.

... additional arguments passed to the function.

Details

The parameter for which a model-averaged estimate is requested must be specified with the parm argument and must be identical to its label in the model output (e.g., from summary). For factors, one must specify the name of the variable and the level of interest. The shrinkage version of model averaging is only appropriate for cases where each parameter is given an equal weighting in the model (i.e., each parameter must appear the same number of times in the models) and has the same interpretation across all models. As a result, models with interaction terms or polynomial terms are not supported by modavgShrink.

modavgShrink is implemented for a list containing objects of aov, betareg, clm, clmm, clogit, coxme, coxph, glm, glmmTMB, gls, hurdle, lm, lme, lmekin, maxlikeFit, mer, glmerMod, lmerMod, lmerModLmerTest, multinom, polr, rlm, survreg, vglm, zeroinfl classes as well as various models of unmarkedFit classes.

Value

modavgShrink creates an object of class modavgShrink with the following components:

Parameter	the parameter for which a model-averaged estimate with shrinkage was obtained.
Mod.avg.table	the model selection table based on models including the parameter of interest.
Mod.avg.beta	the model-averaged estimate based on all models.
Uncond.SE	the unconditional standard error for the model-averaged estimate (as opposed to the conditional SE based on a single model).
Conf.level	the confidence level used to compute the confidence interval.
Lower.CL	the lower confidence limit.
Upper.CL	the upper confidence limit.

Author(s)

Marc J. Mazerolle

References

- Anderson, D. R. (2008) *Model-based Inference in the Life Sciences: a primer on evidence*. Springer: New York.
- Buckland, S. T., Burnham, K. P., Augustin, N. H. (1997) Model selection: an integral part of inference. *Biometrics* **53**, 603–618.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Burnham, K. P., Anderson, D. R. (2004) Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods and Research* **33**, 261–304.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Lukacs, P. M., Burnham, K. P., Anderson, D. R. (2010) Model selection bias and Freedman’s paradox. *Annals of the Institute of Statistical Mathematics* **62**, 117–125.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike’s Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[AICc](#), [aictab](#), [c_hat](#), [importance](#), [confset](#), [evidence](#), [modavg](#), [modavgCustom](#), [modavgPred](#)

Examples

```
##cement example in Burnham and Anderson 2002
data(cement)
##setup same model set as in Table 3.2, p. 102
Cand.models <- list( )
Cand.models[[1]] <- lm(y ~ x1 + x2, data = cement)
Cand.models[[2]] <- lm(y ~ x1 + x2 + x4, data = cement)
Cand.models[[3]] <- lm(y ~ x1 + x2 + x3, data = cement)
Cand.models[[4]] <- lm(y ~ x1 + x4, data = cement)
Cand.models[[5]] <- lm(y ~ x1 + x3 + x4, data = cement)
Cand.models[[6]] <- lm(y ~ x2 + x3 + x4, data = cement)
Cand.models[[7]] <- lm(y ~ x1 + x2 + x3 + x4, data = cement)
Cand.models[[8]] <- lm(y ~ x3 + x4, data = cement)
Cand.models[[9]] <- lm(y ~ x2 + x3, data = cement)
Cand.models[[10]] <- lm(y ~ x4, data = cement)
Cand.models[[11]] <- lm(y ~ x2, data = cement)
Cand.models[[12]] <- lm(y ~ x2 + x4, data = cement)
Cand.models[[13]] <- lm(y ~ x1, data = cement)
```

```

Cand.models[[14]] <- lm(y ~ x1 + x3, data = cement)
Cand.models[[15]] <- lm(y ~ x3, data = cement)

##vector of model names
Modnames <- paste("mod", 1:15, sep="")

##AICc
aictab(cand.set = Cand.models, modnames = Modnames)

##compute model-averaged estimate with shrinkage - each parameter
##appears 8 times in the models
modavgShrink(cand.set = Cand.models, modnames = Modnames, parm = "x1")

##compare against classic model-averaging
modavg(cand.set = Cand.models, modnames = Modnames, parm = "x1")
##note that model-averaged estimate with shrinkage is closer to 0 than
##with the classic version

##remove a few models from the set and run again
Cand.unbalanced <- Cand.models[-c(3, 14, 15)]

##set up model names
Modnames <- paste("mod", 1:length(Cand.unbalanced), sep="")

##issues an error because some parameters appear more often than others
## Not run: modavgShrink(cand.set = Cand.unbalanced,
                        modnames = Modnames, parm = "x1")
## End(Not run)

##example on Orthodont data set in nlme
## Not run:
require(nlme)

##set up candidate model list
##age and sex parameters appear in the same number of models
##same number of models with and without these parameters
Cand.models <- list( )
Cand.models[[1]] <- lme(distance ~ age, data = Orthodont, method = "ML")
##random is ~ age | Subject as it is a grouped data frame
Cand.models[[2]] <- lme(distance ~ age + Sex, data = Orthodont,
                        random = ~ 1, method = "ML")
Cand.models[[3]] <- lme(distance ~ 1, data = Orthodont, random = ~ 1,
                        method = "ML")
Cand.models[[4]] <- lme(distance ~ Sex, data = Orthodont, random = ~ 1,
                        method = "ML")

##create a vector of model names
Modnames <- paste("mod", 1:length(Cand.models), sep = "")

##compute importance values for age
imp.age <- importance(cand.set = Cand.models, parm = "age",

```

```

modnames = Modnames, second.ord = TRUE,
nobs = NULL)

##compute shrinkage version of model averaging on age
mod.avg.age.shrink <- modavgShrink(cand.set = Cand.models,
                                parm = "age", modnames = Modnames,
                                second.ord = TRUE, nobs = NULL)

##compute classic version of model averaging on age
mod.avg.age.classic <- modavg(cand.set = Cand.models, parm = "age",
                              modnames = Modnames, second.ord = TRUE,
                              nobs = NULL)

##correspondence between shrinkage version and classic version of
##model averaging
mod.avg.age.shrink$Mod.avg.beta/imp.age$w.plus
mod.avg.age.classic$Mod.avg.beta
detach(package:nlme)

## End(Not run)

##example of N-mixture model modified from ?pcount
## Not run:
require(unmarked)
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
                                obsCovs = mallard.obs)
##set up models so that each variable on abundance appears twice
fm.mall.one <- pcount(~ ivel + date ~ length + forest, mallardUMF,
                    K = 30)
fm.mall.two <- pcount(~ ivel + date ~ elev + forest, mallardUMF,
                    K = 30)
fm.mall.three <- pcount(~ ivel + date ~ length + elev, mallardUMF,
                      K = 30)

##model list and names
Cands <- list(fm.mall.one, fm.mall.two, fm.mall.three)
Modnames <- c("length + forest", "elev + forest", "length + elev")

##compute model-averaged estimate with shrinkage for elev on abundance
modavgShrink(cand.set = Cands, modnames = Modnames, parm = "elev",
             parm.type = "lambda")
detach(package:unmarked)

## End(Not run)

```

Description

This function is an alternative to traditional multiple comparison tests in designed experiments. It creates a model selection table based on different grouping patterns of a factor and computes model-averaged predictions for each of the factor levels. The current version works with objects of `aov`, `glm`, `gls`, `lm`, `lme`, `mer`, `merMod`, `lmerModLmerTest`, `negbin`, and `rlm`, `survreg` classes.

Usage

```
multComp(mod, factor.id, letter.labels = TRUE, second.ord = TRUE,
         nobs = NULL, sort = TRUE, newdata = NULL, uncond.se = "revised",
         conf.level = 0.95, correction = "none", ...)

## S3 method for class 'aov'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         ...)

## S3 method for class 'lm'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         ...)

## S3 method for class 'gls'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         ...)

## S3 method for class 'glm'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         type = "response", c.hat = 1, gamdisp = NULL, ...)

## S3 method for class 'lme'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         ...)

## S3 method for class 'negbin'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         type = "response", ...)
```

```

## S3 method for class 'rlm'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         ...)

## S3 method for class 'survreg'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         type = "response", ...)

## S3 method for class 'mer'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95, correction = "none",
         type = "response", ...)

## S3 method for class 'merMod'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95,
         correction = "none", type = "response", ...)

## S3 method for class 'lmerModLmerTest'
multComp(mod, factor.id, letter.labels = TRUE,
         second.ord = TRUE, nobs = NULL, sort = TRUE, newdata = NULL,
         uncond.se = "revised", conf.level = 0.95,
         correction = "none", ...)

```

Arguments

<code>mod</code>	a model of one of the above-mentioned classes that includes at least one factor as an explanatory variable.
<code>factor.id</code>	the factor of interest, on which the groupings (multiple comparisons) are based. The user must supply the name of the categorical variable between quotes as it appears in the model formula.
<code>letter.labels</code>	logical. If TRUE, letters are used as labels to denote the grouping structure. If FALSE, numbers are used as group labels.
<code>second.ord</code>	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc), otherwise returns Akaike's Information Criterion (AIC).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., <code>nobs</code> defaults to total number of observations). This is relevant only for certain types of models such as mixed models where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>nobs</code> .
<code>sort</code>	logical. If TRUE, the model selection table is ranked according to the (Q)AIC(c) values.

<code>newdata</code>	a data frame with the same structure as that of the original data frame for which we want to make predictions. This data frame should hold all variables constant other than the <code>factor.id</code> variable. All levels of the <code>factor.id</code> variables should be included in the <code>newdata</code> data frame to get model-averaged predictions for each level. If <code>NULL</code> , model-averaged predictions are computed for each level of the <code>factor.id</code> variable while the values of the other explanatory variables are taken from the first row of the original data set.
<code>uncond.se</code>	either, "old", or "revised", specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With <code>uncond.se = "old"</code> , computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With <code>uncond.se = "revised"</code> , equation 6.12 of Burnham and Anderson (2002) is used. Anderson (2008, p. 111) recommends use of the revised version for the computation of unconditional standard errors and it is now the default. Note that versions of package <code>AICcmodavg < 1.04</code> used the old method to compute unconditional standard errors.
<code>conf.level</code>	the confidence level ($1 - \alpha$) requested for the computation of unconditional confidence intervals around predicted values for each level of <code>factor.id</code> .
<code>correction</code>	the type of correction applied to obtain confidence intervals for simultaneous inference (i.e., corrected for multiple comparisons). Current corrections include "none" for uncorrected unconditional confidence intervals, "bonferroni" for Bonferroni-adjusted confidence intervals (Dunn 1961), and "sidak" for Sidak-adjusted confidence intervals (Sidak 1967).
<code>type</code>	the scale of prediction requested, one of "response" or "link". The latter is only relevant for <code>glm</code> and <code>mer</code> classes. Note that the value "terms" is not defined for <code>multComp</code> .
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c_hat</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with <code>trials > 1</code> (i.e., <code>success/trial</code> or <code>cbind(success, failure)</code> syntax) or with Poisson GLM's. If <code>c.hat > 1</code> , <code>multComp</code> will return the quasi-likelihood analogue of the information criterion requested. This option is not supported for generalized linear mixed models of the <code>mer</code> class.
<code>gamdisp</code>	the value of the gamma dispersion parameter in a gamma GLM.
<code>...</code>	additional arguments passed to the function.

Details

A number of pairwise comparison tests are available for traditional experimental designs, some controlling for the experiment-wise error and others for comparison-wise errors (Day and Quinn 1991). With the advent of information-theoretic approaches, there has been a need for methods analogous to multiple comparison tests in a model selection framework. Dayton (1998) and Burnham et al. (2011) suggested using different parameterizations or grouping patterns of a factor to perform multiple comparisons with model selection. As such, it is possible to assess the support in favor of certain grouping patterns based on a factor.

For example, a factor with three levels has four possible grouping patterns: `{abc}` (all groups are different), `{abb}` (the first group differs from the other two), `{aab}` (the first two groups differ from

the third), and {aaa} (all groups are equal). `multComp` implements such an approach by pooling groups of the factor variable in a model and updating the model, for each grouping pattern possible. The models are ranked according to one of four information criteria (AIC, AICc, QAIC, and QAICc), and the labels in the table correspond to the grouping pattern. Note that the factor levels are sorted according to their means for the response variable before being assigned to a group. The function also returns model-averaged predictions and unconditional standard errors for each level of the factor `.id` variable based on the support in favor of each model (i.e., grouping pattern).

The number of grouping patterns increases substantially with the number of factor levels, as 2^{k-1} , where k is the number of factor levels. `multComp` supports factors with a maximum of 6 levels. Also note that `multComp` does not handle models where the factor `.id` variable is involved in an interaction. In such cases, one should create the interaction variable manually before fitting the model (see Examples).

`multComp` currently implements three methods of computing confidence intervals. The default unconditional confidence intervals do not account for multiple comparisons (`correction = "none"`). With a large number m of potential pairwise comparisons among levels of factor `.id`, there is an increased risk of type I error. For m pairwise comparisons and a given α level, `correction = "bonferroni"` computes the unconditional confidence intervals based on $\alpha_{corr} = \frac{\alpha}{m}$ (Dunn 1961). When `correction = "sidak"`, `multComp` reports Sidak-adjusted confidence intervals, i.e., $\alpha_{corr} = 1 - (1 - \alpha)^{\frac{1}{m}}$.

Value

`multComp` creates a list of class `multComp` with the following components:

<code>factor.id</code>	the factor for which grouping patterns are investigated.
<code>models</code>	a list with the output of each model representing a different grouping pattern for the factor of interest.
<code>model.names</code>	a vector of model names denoting the grouping pattern for each level of the factor.
<code>model.table</code>	the model selection table for the models corresponding to each grouping pattern for the factor of interest.
<code>ordered.levels</code>	the levels of the factor ordered according to the mean of the response variable. The grouping patterns (and model names) in the model selection table are based on the same order.
<code>model.avg.est</code>	a matrix with the model-averaged prediction, unconditional standard error, and confidence intervals for each level of the factor.
<code>conf.level</code>	the confidence level used for the confidence intervals.
<code>correction</code>	the type of correction applied to the confidence intervals to account for potential pairwise comparisons.

Author(s)

Marc J. Mazerolle

References

- Burnham, K. P., Anderson, D. R., Huyvaert, K. P. (2011) AIC model selection and multimodel inference in behavioral ecology: some background, observations and comparisons. *Behavioral Ecology and Sociobiology* **65**, 23–25.
- Day, R. W., Quinn, G. P. (1989) Comparisons of treatments after an analysis of variance in ecology. *Ecological Monographs* **59**, 433–463.
- Dayton, C. M. (1998) Information criteria for the paired-comparisons problem. *American Statistician*, **52** 144–151.
- Dunn, O. J. (1961) Multiple comparisons among means. *Journal of the American Statistical Association* **56**, 52–64.
- Sidak, Z. (1967) Rectangular confidence regions for the means of multivariate normal distributions. *Journal of the American Statistical Association* **62**, 626–633.

See Also

[aictab](#), [confset](#), [c_hat](#), [evidence](#), [glht](#), [fit.contrast](#)

Examples

```
##one-way ANOVA example
data(turkey)

##convert diet to factor
turkey$Diet <- as.factor(turkey$Diet)
##run one-way ANOVA
m.aov <- lm(Weight.gain ~ Diet, data = turkey)

##compute models with different grouping patterns
##and also compute model-averaged group means
out <- multComp(m.aov, factor.id = "Diet", correction = "none")
##look at results
out

##look at grouping structure of a given model
##and compare with original variable
cbind(model.frame(out$models[[2]]), turkey$Diet)

##evidence ratio
evidence(out$model.table)

##compute Bonferroni-adjusted confidence intervals
multComp(m.aov, factor.id = "Diet", correction = "bonferroni")

##two-way ANOVA with interaction
## Not run:
data(calcium)

m.aov2 <- lm(Calcium ~ Hormone + Sex + Hormone:Sex, data = calcium)
```

```

##multiple comparisons
multComp(m.aov2, factor.id = "Hormone")
##returns an error because 'Hormone' factor is
##involved in an interaction

##create interaction variable
calcium$inter <- interaction(calcium$Hormone, calcium$Sex)

##run model with interaction
m.aov.inter <- lm(Calcium ~ inter, data = calcium)

##compare both
logLik(m.aov2)
logLik(m.aov.inter)
##both are identical

##multiple comparisons
multComp(m.aov.inter, factor.id = "inter")

## End(Not run)

##Poisson regression
## Not run:
##example from ?glm
##Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
glm.D93 <- glm(counts ~ outcome + treatment, data = d.AD, family = poisson)

multComp(mod = glm.D93, factor.id = "outcome")

## End(Not run)

##example specifying 'newdata'
## Not run:
data(dry.frog)
m1 <- lm(log_Mass_lost ~ Shade + Substrate +
         cent_Initial_mass + Initial_mass2,
         data = dry.frog)

multComp(m1, factor.id = "Substrate",
         newdata = data.frame(
           Substrate = c("PEAT", "SOIL", "SPHAGNUM"),
           Shade = 0, cent_Initial_mass = 0,
           Initial_mass2 = 0))

## End(Not run)

```

newt

*Newt Capture-mark-recapture Data***Description**

This is a capture-mark-recapture data set on adult male and female Red-spotted Newts (*Notophthalmus viridescens*) recorded by Gill (1985). A total of 1079 unique individuals were captured in pitfall traps at a breeding site (White Oak Flat pond, Virginia) between 1975 and 1983.

Usage

```
data(newt)
```

Format

A data frame with 78 observations on the following 11 variables.

T1975 a binary variable, either 1 (captured) or 0 (not captured) during the 1975 breeding season.

T1976 a binary variable, either 1 (captured) or 0 (not captured) during the 1976 breeding season.

T1977 a binary variable, either 1 (captured) or 0 (not captured) during the 1977 breeding season.

T1978 a binary variable, either 1 (captured) or 0 (not captured) during the 1978 breeding season.

T1979 a binary variable, either 1 (captured) or 0 (not captured) during the 1979 breeding season.

T1980 a binary variable, either 1 (captured) or 0 (not captured) during the 1980 breeding season.

T1981 a binary variable, either 1 (captured) or 0 (not captured) during the 1981 breeding season.

T1982 a binary variable, either 1 (captured) or 0 (not captured) during the 1982 breeding season.

T1983 a binary variable, either 1 (captured) or 0 (not captured) during the 1983 breeding season.

Males a numeric variable indicating the total number of males with a given capture history.

Females a numeric variable indicating the total number of females with a given capture history.

Details

A single cohort of individuals was followed throughout the study, as all individuals were marked in 1975 and no new individuals were added during the subsequent years. This data set is used to illustrate classic Cormack-Jolly-Seber and related models (Cormack 1964, Jolly 1965, Seber 1965, Lebreton et al. 1992, Mazerolle 2015).

Source

Cormack, R. M. (1964) Estimates of survival from the sighting of marked animals. *Biometrika* **51**, 429–438.

Gill, D. E. (1985) Interpreting breeding patterns from census data: a solution to the Husting dilemma. *Ecology* **66**, 344–354.

Jolly, G. M. (1965) Explicit estimates from capture-recapture data with both death and immigration: stochastic model. *Biometrika* **52**, 225–247.

Laake, J. L. (2013) *RMark: an R interface for analysis of capture-recapture data with MARK*. Alaska Fisheries Science Center (AFSC), National Oceanic and Atmospheric Administration, National Marine Fisheries Service, AFSC Report 2013-01.

Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67-118.

Mazerolle, M. J. (2015) Estimating detectability and biological parameters of interest with the use of the R environment. *Journal of Herpetology* **49**, 541–559.

Seber, G. A. F. (1965) A note on the multiple-recapture census. *Biometrika* **52**, 249–259.

Examples

```
data(newt)
str(newt)

##convert raw capture data to capture histories
captures <- newt[, c("T1975", "T1976", "T1977", "T1978", "T1979",
                    "T1980", "T1981", "T1982", "T1983")]
newt.ch <- apply(captures, MARGIN = 1, FUN = function(i)
                paste(i, collapse = ""))

##organize as a data frame readable by RMark package (Laake 2013)
##RMark requires at least one column called "ch"
##and another "freq" if summarized captures are provided
newt.full <- data.frame(ch = rep(newt.ch, 2),
                       freq = c(newt$Males, newt$Females),
                       Sex = c(rep("male", length(newt.ch)),
                               rep("female", length(newt.ch))))

str(newt.full)
newt.full$ch <- as.character(newt.full$ch)

##delete rows with 0 freqs
newt.full.orig <- newt.full[which(newt.full$freq != 0), ]
```

Nmix.gof.test

Compute Chi-square Goodness-of-fit Test for N-mixture Models

Description

These functions compute a goodness-of-fit test for N-mixture models based on Pearson's chi-square.

Usage

```
##methods for 'unmarkedFitPCount', 'unmarkedFitPCO',
##'unmarkedFitDS', 'unmarkedFitGDS', 'unmarkedFitGMM',
##'unmarkedFitGPC', and 'unmarkedFitMPois' classes
Nmix.chisq(mod, ...)
```

```
Nmix.gof.test(mod, nsim = 5, plot.hist = TRUE, report = NULL,
parallel = TRUE, ncores, cex.axis = 1, cex.lab = 1, cex.main = 1,
lwd = 1, ...)
```

Arguments

<code>mod</code>	the N -mixture model of <code>unmarkedFitPCount</code> , <code>unmarkedFitPC0</code> , <code>unmarkedFitDS</code> , <code>unmarkedFitGDS</code> , <code>unmarkedFitGMM</code> , <code>unmarkedFitGPC</code> , <code>unmarkedFitMPois</code> , <code>unmarkedFitMMO</code> , or <code>unmarkedFitDSO</code> classes for which a goodness-of-fit test is required.
<code>nsim</code>	the number of bootstrapped samples.
<code>plot.hist</code>	logical. Specifies that a histogram of the bootstrapped test statistic is to be included in the output.
<code>report</code>	If <code>NULL</code> , the test statistic for each iteration is not printed in the terminal. Otherwise, an integer indicating the number of values of the test statistic that should be printed on the same line. For example, if <code>report = 3</code> , the values of the test statistic for three iterations are reported on each line.
<code>parallel</code>	logical. If <code>TRUE</code> , requests that <code>parboot</code> use multiple cores to accelerate computations of the bootstrap.
<code>ncores</code>	integer indicating the number of cores to use when bootstrapping in parallel during the analysis of simulated data sets. If <code>ncores</code> is not specified, one less than the number of available cores on the computer is used.
<code>cex.axis</code>	expansion factor influencing the size of axis annotations on plots produced by the function.
<code>cex.lab</code>	expansion factor influencing the size of axis labels on plots produced by the function.
<code>cex.main</code>	expansion factor influencing the size of the main title above plots produced by the function.
<code>lwd</code>	expansion factor of line width on plots produced by the function.
<code>...</code>	additional arguments passed to the function.

Details

The Pearson chi-square can be used to assess the fit of N -mixture models. Instead of relying on the theoretical distribution of the chi-square, a parametric bootstrap approach is implemented to obtain P -values with the `parboot` function of the `unmarked` package. `Nmix.chisq` computes the observed chi-square statistic based on the observed and expected counts from the model. `Nmix.gof.test` calls internally `Nmix.chisq` and `parboot` to generate simulated data sets based on the model and compute the chi-square test statistic.

It is also possible to obtain an estimate of the overdispersion parameter (\hat{c}) for the model at hand by dividing the observed chi-square statistic by the mean of the statistics obtained from simulation (MacKenzie and Bailey 2004, McKenny et al. 2006). This method of estimating \hat{c} is similar to the one implemented for capture-mark-recapture models in program MARK (White and Burnham 1999).

Note that values of $\hat{c} > 1$ indicate overdispersion (variance $>$ mean). Values much higher than 1 (i.e., > 4) probably indicate lack-of-fit. In cases of moderate overdispersion, one can multiply

the variance-covariance matrix of the estimates by c -hat. As a result, the SE's of the estimates are inflated (c -hat is also known as a variance inflation factor).

In model selection, c -hat should be estimated from the global model and the same value of c -hat applied to the entire model set. Specifically, a global model is the most complex model which can be simplified to yield all the other (nested) models of the set. When no single global model exists in the set of models considered, such as when sample size does not allow a complex model, one can estimate c -hat from 'subglobal' models. Here, 'subglobal' models denote models from which only a subset of the models of the candidate set can be derived. In such cases, one can use the smallest value of c -hat for model selection (Burnham and Anderson 2002).

Note that c -hat counts as an additional parameter estimated and should be added to K . All functions in package AICcmoavg automatically add 1 when the `c.hat` argument > 1 and apply the same value of c -hat for the entire model set. When c -hat > 1 , functions compute quasi-likelihood information criteria (either QAICc or QAIC, depending on the value of the second `ord` argument) by scaling the log-likelihood of the model by c -hat. The value of c -hat can influence the ranking of the models: as c -hat increases, QAIC or QAICc will favor models with fewer parameters. As an additional check against this potential problem, one can generate several model selection tables by incrementing values of c -hat to assess the model selection uncertainty. If ranking changes only slightly up to the c -hat value observed, one can be confident in making inference.

In cases of underdispersion (c -hat < 1), it is recommended to keep the value of c -hat to 1. However, note that values of c -hat $\ll 1$ can also indicate lack-of-fit and that an alternative model should be investigated.

Value

`Nmix.chisq` returns two value:

<code>chi.square</code>	the Pearson chi-square statistic.
<code>model.type</code>	the class of the fitted model.

`Nmix.gof.test` returns the following components:

<code>model.type</code>	the class of the fitted model.
<code>chi.square</code>	the Pearson chi-square statistic.
<code>t.star</code>	the bootstrapped chi-square test statistics (i.e., obtained for each of the simulated data sets).
<code>p.value</code>	the P -value assessed from the parametric bootstrap, computed as the proportion of the simulated test statistics greater than or equal to the observed test statistic.
<code>c.hat.est</code>	the estimate of the overdispersion parameter, c -hat, computed as the observed test statistic divided by the mean of the simulated test statistics.
<code>nsim</code>	the number of bootstrap samples. The recommended number of samples varies with the data set, but should be on the order of 1000 or 5000, and in cases with a large number of visits, even 10 000 samples, namely to reduce the effect of unusually small values of the test statistics.

Author(s)

Marc J. Mazerolle

References

- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- MacKenzie, D. I., Bailey, L. L. (2004) Assessing the fit of site-occupancy models. *Journal of Agricultural, Biological, and Environmental Statistics* **9**, 300–318.
- McKenny, H. C., Keeton, W. S., Donovan, T. M. (2006). Effects of structural complexity enhancement on eastern red-backed salamander (*Plethodon cinereus*) populations in northern hardwood forests. *Forest Ecology and Management* **230**, 186–196.
- White, G. C., Burnham, K. P. (1999). Program MARK: Survival estimation from populations of marked animals. *Bird Study* **46 (Supplement)**, 120–138.

See Also

[AICc](#), [c_hat](#), [evidence](#), [modavg](#), [importance](#), [mb.gof.test](#), [modavgPred](#), [pcount](#), [pcountOpen](#), [parboot](#)

Examples

```
##N-mixture model example modified from ?pcount
## Not run:
require(unmarked)
##single season
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
                                  obsCovs = mallard.obs)

##run model
fm.mallard <- pcount(~ ivel+ date + I(date^2) ~ length + elev + forest,
                    mallardUMF, K=30)

##compute observed chi-square
obs <- Nmix.chisq(fm.mallard)
obs

##round to 4 digits after decimal point
print(obs, digits.vals = 4)

##compute observed chi-square, assess significance, and estimate c-hat
obs.boot <- Nmix.gof.test(fm.mallard, nsim = 10)
##note that more bootstrap samples are recommended
##(e.g., 1000, 5000, or 10 000)
obs.boot
print(obs.boot, digits.vals = 4, digits.chisq = 4)
detach(package:unmarked)

## End(Not run)
```

pine	<i>Strength of Pine Wood Based on the Density Adjusted for Resin Content</i>
------	--

Description

This data set consists of the strength of pine wood as a function of density or density adjusted for resin content.

Usage

```
data(pine)
```

Format

A data frame with 42 observations on the following 3 variables.

y pine wood strength.

x pine wood density.

z pine wood density adjusted for resin content.

Details

Burnham and Anderson (2002, p. 183) use this data set originally from Carlin and Chib (1995) to illustrate model selection for two competing and non-nested models.

Source

Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.

Carlin, B. P., Chib, S. (1995) Bayesian model choice via Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society, Series B* **57**, 473–484.

Examples

```
data(pine)
## maybe str(pine) ; plot(pine) ...
```

Description

Function to compute predicted values based on linear predictor and associated standard errors from various fitted models.

Usage

```

predictSE(mod, newdata, se.fit = TRUE, print.matrix = FALSE, ...)

## S3 method for class 'gls'
predictSE(mod, newdata, se.fit = TRUE, print.matrix =
  FALSE, ...)

## S3 method for class 'lme'
predictSE(mod, newdata, se.fit = TRUE, print.matrix =
  FALSE, level = 0, ...)

## S3 method for class 'mer'
predictSE(mod, newdata, se.fit = TRUE, print.matrix =
  FALSE, level = 0, type = "response", ...)

## S3 method for class 'merMod'
predictSE(mod, newdata, se.fit = TRUE, print.matrix =
  FALSE, level = 0, type = "response", ...)

## S3 method for class 'lmerModLmerTest'
predictSE(mod, newdata, se.fit = TRUE, print.matrix =
  FALSE, level = 0, ...)

## S3 method for class 'unmarkedFitPCCount'
predictSE(mod, newdata, se.fit = TRUE,
  print.matrix = FALSE, type = "response", c.hat = 1, parm.type =
  "lambda", ...)

## S3 method for class 'unmarkedFitPCO'
predictSE(mod, newdata, se.fit = TRUE,
  print.matrix = FALSE, type = "response", c.hat = 1,
  parm.type = "lambda", ...)

```

Arguments

`mod` an object of class `gls`, `lme`, `mer`, `merMod`, `lmerModLmerTest`, `unmarkedFitPCCount`, or `unmarkedFitPCO` containing the output of a model.

<code>newdata</code>	a data frame with the same structure as that of the original data frame for which we want to make predictions.
<code>se.fit</code>	logical. If TRUE, compute standard errors on predictions.
<code>print.matrix</code>	logical. If TRUE, the output is returned as a matrix, with predicted values and standard errors in columns. If FALSE, the output is returned as a list.
<code>level</code>	the level for which predicted values and standard errors are to be computed. The current version of the function only supports predictions for the populations excluding random effects (i.e., <code>level = 0</code>).
<code>type</code>	specifies the type of prediction requested. This argument can take the value <code>response</code> or <code>link</code> , for predictions on the scale of the response variable or on the scale of the linear predictor, respectively.
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>Nmix.gof.test</code> . If <code>c.hat > 1</code> , <code>predictSE</code> will multiply the variance-covariance matrix of the predictions by this value (i.e., SE's are multiplied by $\sqrt{c.hat}$). High values of <code>c.hat</code> (e.g., <code>c.hat > 4</code>) may indicate that model structure is inappropriate.
<code>parm.type</code>	the parameter for which predictions are made based on the <i>N</i> -mixture model of class <code>unmarkedFitPCount</code> or <code>unmarkedFitPCO</code> classes.
<code>...</code>	additional arguments passed to the function.

Details

`predictSE` computes predicted values and associated standard errors. Standard errors are approximated using the delta method (Oehlert 1992). Predictions and standard errors for objects of `gls` class and mixed models of `lme`, `mer`, `merMod`, `lmerMod`, `lmerTest` classes exclude the correlation or variance structure of the model.

`predictSE` computes predicted values on abundance and standard errors based on the estimates from an `unmarkedFitPCount` or `unmarkedFitPCO` object. Currently, only predictions on abundance (i.e., `parm.type = "lambda"`) with the zero-inflated Poisson distribution is supported. For other parameters or distributions for models of `unmarkedFit` classes, use `predict` from the `unmarked` package.

Value

`predictSE` returns requested values either as a matrix (`print.matrix = TRUE`) or list (`print.matrix = FALSE`) with components:

<code>fit</code>	the predicted values.
<code>se.fit</code>	the standard errors of the predicted values (if <code>se.fit = TRUE</code>).

Note

For standard errors with better properties, especially for small samples, one can opt for simulations (see Gelman and Hill 2007), or nonparametric bootstrap (Efron and Tibshirani 1998).

Author(s)

Marc J. Mazerolle

References

- Efron, B., Tibshirani, R. J. (1998) *An Introduction to the Bootstrap*. Chapman & Hall/CRC: New York.
- Gelman, A., Hill, J. (2007) *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press: New York.
- Oehlert, G. W. (1992) A note on the delta method. *American Statistician* **46**, 27–29.

See Also

[gls](#), [lme](#), [glmer](#), [simulate.merMod](#), [boot](#), [parboot](#), [nonparboot](#), [pcount](#), [pcountOpen](#), [unmarkedFit-class](#)

Examples

```
##Orthodont data from Pinheiro and Bates (2000) revisited
## Not run:
require(nlme)
m1 <- gls(distance ~ age, correlation = corCompSymm(value = 0.5, form = ~ 1 | Subject),
          data = Orthodont, method= "ML")

##compare against lme fit
logLik(m1)
logLik(lme(distance ~ age, random = ~1 | Subject, data = Orthodont,
           method= "ML"))
##both are identical

##compute predictions and SE's for different ages
predictSE(m1, newdata = data.frame(age = c(8, 10, 12, 14)))
detach(package:nlme)

## End(Not run)

##example with mallard data set from unmarked package
## Not run:
require(unmarked)
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
                                 obsCovs = mallard.obs)
##run model with zero-inflated Poisson abundance
fm.mall.one <- pcount(~ ivel + date ~ length + forest, mallardUMF, K=30,
                    mixture = "ZIP")
##make prediction
predictSE(fm.mall.one, type = "response", parm.type = "lambda",
          newdata = data.frame(length = 0, forest = 0, elev = 0))
##compare against predict
predict(fm.mall.one, type = "state", backTransform = TRUE,
       newdata = data.frame(length = 0, forest = 0, elev = 0))

##add offset in model to scale abundance per transect length
fm.mall.off <- pcount(~ ivel + date ~ forest + offset(length), mallardUMF, K=30,
```

```

                                mixture = "ZIP")
##make prediction
predictSE(fm.mall.off, type = "response", parm.type = "lambda",
          newdata = data.frame(length = 10, forest = 0, elev = 0))
##compare against predict
predict(fm.mall.off, type = "state", backTransform = TRUE,
        newdata = data.frame(length = 10, forest = 0, elev = 0))
detach(package:unmarked)

## End(Not run)

```

salamander

Salamander Capture-mark-recapture Data

Description

This is a capture-mark-recapture data set on male and female Spotted Salamanders (*Ambystoma maculatum*) recorded by Husting (1965). A total of 1244 unique individuals were captured in pitfall traps at a breeding site between 1959 and 1963.

Usage

```
data(salamander)
```

Format

A data frame with 36 observations on the following 7 variables.

T1959 a binary variable, either 1 (captured) or 0 (not captured) during the 1959 breeding season.

T1960 a binary variable, either 1 (captured) or 0 (not captured) during the 1960 breeding season.

T1961 a binary variable, either 1 (captured) or 0 (not captured) during the 1961 breeding season.

T1962 a binary variable, either 1 (captured) or 0 (not captured) during the 1962 breeding season.

T1963 a binary variable, either 1 (captured) or 0 (not captured) during the 1963 breeding season.

Males a numeric variable indicating the total number of males with a given capture history. Negative values indicate losses on capture (animals not released on last capture).

Females a numeric variable indicating the total number of females with a given capture history. Negative values indicate losses on capture (animals not released on last capture).

Details

This data set is used to illustrate classic Cormack-Jolly-Seber and related models (Cormack 1964, Jolly 1965, Seber 1965, Lebreton et al. 1992).

Source

Cormack, R. M. (1964) Estimates of survival from the sighting of marked animals. *Biometrika* **51**, 429–438.

Husting, E. L. (1965) Survival and breeding structure in a population of *Ambystoma maculatum*. *Copeia* **1965**, 352–362.

Jolly, G. M. (1965) Explicit estimates from capture-recapture data with both death and immigration: stochastic model. *Biometrika* **52**, 225–247.

Laake, J. L. (2013) *RMark: an R interface for analysis of capture-recapture data with MARK*. Alaska Fisheries Science Center (AFSC), National Oceanic and Atmospheric Administration, National Marine Fisheries Service, AFSC Report 2013-01.

Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67–118.

Seber, G. A. F. (1965) A note on the multiple-recapture census. *Biometrika* **52**, 249–259.

Examples

```
data(salamander)
str(salamander)

##convert raw capture data to capture histories
captures <- salamander[, c("T1959", "T1960", "T1961", "T1962", "T1963")]
salam.ch <- apply(captures, MARGIN = 1, FUN = function(i)
  paste(i, collapse = ""))

##organize as a data frame readable by RMark package (Laake 2013)
##RMark requires at least one column called "ch"
##and another "freq" if summarized captures are provided
salam.full <- data.frame(ch = rep(salam.ch, 2),
  freq = c(salamander$Males, salamander$Females),
  Sex = c(rep("male", length(salam.ch)),
  rep("female", length(salam.ch))))

str(salam.full)
salam.full$ch <- as.character(salam.full$ch)

##delete rows with 0 freqs
salam.full.orig <- salam.full[which(salam.full$freq != 0), ]
```

Description

This function displays the estimates of a model with standard errors corrected for overdispersion for a variety of model classes. The output includes either confidence intervals based on the normal approximation or Wald hypothesis tests corrected for overdispersion.

Usage

```
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'glm'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitOccu'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitColExt'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitOccuRN'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitPCount'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitPCO'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitDS'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitGDS'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitOccuFP'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitMPois'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)

## S3 method for class 'unmarkedFitGMM'
summaryOD(mod, c.hat = 1, conf.level = 0.95,
           out.type = "confint", ...)
```

```
## S3 method for class 'unmarkedFitGPC'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'unmarkedFitOccuMulti'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'unmarkedFitOccuMS'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'unmarkedFitOccuTTD'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'unmarkedFitMMO'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'unmarkedFitDSO'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'unmarkedFitGOccu'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'unmarkedFitOccuComm'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'glmerMod'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'maxlikeFit'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'multinom'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)  
  
## S3 method for class 'vglm'  
summaryOD(mod, c.hat = 1, conf.level = 0.95,  
  out.type = "confint", ...)
```

Arguments

<code>mod</code>	an object of class <code>glm</code> , <code>glmmTMB</code> , <code>maxlikeFit</code> , <code>mer</code> , <code>merMod</code> , <code>multinom</code> , <code>vglm</code> , and various <code>unmarkedFit</code> classes containing the output of a model.
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>c.hat</code> , <code>mb.gof.test</code> , or <code>Nmix.gof.test</code> .
<code>conf.level</code>	the confidence level ($1 - \alpha$) requested for the computation of confidence intervals.
<code>out.type</code>	the type of summary requested for each parameter estimate. If <code>out.type = "confint"</code> , computes confidence intervals corrected for overdispersion, whereas <code>out.type = "nhst"</code> conducts null-hypothesis statistical testing corrected for overdispersion.
<code>...</code>	additional arguments passed to the function.

Details

Overdispersion occurs when the variance in the data exceeds that expected from a theoretical distribution such as the Poisson or binomial (McCullagh and Nelder 1989, Burnham and Anderson 2002). When the model is correct, small values of `c.hat` ($1 < \text{c.hat} < 4$) can reflect minor deviations from model assumptions (Burnham and Anderson 2002). In such cases, it is possible to adjust standard errors of parameter estimates by multiplying with `sqrt(c.hat)` (McCullagh and Nelder 1989). This is the correction applied by `summaryOD`.

Depending on the type of summary requested, i.e., `out.type = "confint"` or `out.type = "nhst"`, `summaryOD` will return either confidence intervals based on the normal approximation or Wald tests for each parameter estimate (Agresti 1990).

For binomial distributions, note that values of `c.hat` > 1 are only appropriate with trials > 1 (i.e., `success/trial` or `cbind(success, failure)` syntax). The function supports different model types such as Poisson GLM's and GLMM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or N -mixture models (Royle 2004, Dail and Madsen 2011).

Value

`summaryOD` returns an object of class `summaryOD` as a list with the following components:

<code>out.type</code>	the type of output requested by the user.
<code>c.hat</code>	the <code>c.hat</code> estimate used to adjust standard errors.
<code>conf.level</code>	the confidence level used to compute confidence intervals around the estimates.
<code>outMat</code>	the output of the model corrected for overdispersion organized in a matrix.

Author(s)

Marc J. Mazerolle

References

- Agresti, A. (2002) *Categorical Data Analysis*. Second edition. John Wiley and Sons: New Jersey.
- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.
- Mazerolle, M. J. (2006) Improving data analysis in herpetology: using Akaike's Information Criterion (AIC) to assess the strength of biological hypotheses. *Amphibia-Reptilia* **27**, 169–180.
- McCullagh, P., Nelder, J. A. (1989) *Generalized Linear Models*. Second edition. Chapman and Hall: New York.
- Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

See Also

[c_hat](#), [mb.gof.test](#), [Nmix.gof.test](#), [anovaOD](#)

Examples

```
##anuran larvae example from Mazerolle (2006)
data(min.trap)
##assign "UPLAND" as the reference level as in Mazerolle (2006)
min.trap$type <- relevel(min.trap$type, ref = "UPLAND")

##run model
m1 <- glm(Num_anura ~ Type + log.Perimeter + Num_ranatra,
          family = poisson, offset = log(Effort),
          data = min.trap)

##check c-hat for global model
c_hat(m1) #uses Pearson's chi-square/df

##display results corrected for overdispersion
summaryOD(m1, c_hat(m1))
summaryOD(m1, c_hat(m1), out.type = "nhst")

##example with occupancy model
## Not run:
##load unmarked package
if(require(unmarked)){
  data(bullfrog)
```

```

##detection data
detections <- bullfrog[, 3:9]

##assemble in unmarkedFrameOccu
bfrog <- unmarkedFrameOccu(y = detections)

##run model
fm <- occu(~ 1 ~ 1, data = bfrog)

##check GOF
##GOF <- mb.gof.test(fm, nsim = 1000)
##estimate of c-hat: 1.89

##display results after overdispersion adjustment
summaryOD(fm, c.hat = 1.89)
summaryOD(fm, c.hat = 1.89, out.type = "nhst")

detach(package:unmarked)
}

## End(Not run)

```

tortoise

Gopher Tortoise Distance Sampling Data

Description

This simulated data set by Mazerolle (2015) is based on the biological parameters for the Gopher Tortoise (*Gopherus polyphemus*) reported by Smith et al. (2009). A half-normal distribution with a scale of 10 and without an adjustment factor was used to simulate the distance data for a study area of 120 km^2 . An effort of 500 m in 300 line transects was deployed. A density of 72 individuals per km^2 was used in the simulation using the approach outlined in Buckland et al. (2001).

Usage

```
data(tortoise)
```

Format

A data frame with 410 observations on the following 5 variables.

Region.Label a numeric identifier for the study area.

Area a numeric variable for the surface area of the study area in square meters.

Sample.Label a numeric identifier for each line transect relating each observation to its corresponding transect.

Effort Effort in meters expended in each line transect.

distance a numeric variable for the perpendicular distances in meters relative to the transect line for each of the individuals detected during the survey. Note that transects without detections have a value of NA for this variable.

Details

This data set is used to illustrate classic distance sampling (Buckland et al. 2001, Mazerolle 2015).

Source

Buckland, S. T., Anderson, D. R., Burnham, K. P., Laake, J. L., Borchers, D. L., Thomas, L. (2001) *Introduction to distance sampling: estimating abundance of biological populations*. Oxford University Press: Oxford.

Mazerolle, M. J. (2015) Estimating detectability and biological parameters of interest with the use of the R environment. *Journal of Herpetology* **49**, 541–559.

Smith, L. L., Linehan, J. M., Stober, J. M., Elliott, M. J., Jensen, J. B. (2009) An evaluation of distance sampling for large-scale gopher tortoise surveys in Georgia, USA. *Applied Herpetology* **6**, 355–368.

Examples

```
data(tortoise)
str(tortoise)

##plot distance data to determine if truncation is required
##(Buckland et al. 2001, pp. 15--17)
hist(tortoise$distance)
```

 turkey

Turkey Weight Gain

Description

This one-way ANOVA data set presents turkey weight gain in pounds across five diets.

Usage

```
data(turkey)
```

Format

A data frame with 30 rows and 2 variables.

Diet diet factor with 5 levels.

Weight.gain weight gain in pounds.

Details

Heiberger and Holland (2004) and Ott (1993) analyze this data set to illustrate one-way ANOVA.

Source

Heiberger, R. M., Holland, B. (2004) *Statistical Analysis and Data Display: an intermediate course with examples in S-Plus, R, and SAS*. Springer: New York.

Ott, R. L. (1993) *An Introduction to Statistical Methods and Data Analysis*. Fourth edition. Duxbury: Pacific Grove, CA.

Examples

```
data(turkey)
str(turkey)
```

```
useBIC
```

Computing BIC or QBIC

Description

Functions to compute the Bayesian information criterion (BIC) or a quasi-likelihood analogue (QBIC).

Usage

```
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'aov'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'betareg'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'clm'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'clmm'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'coxme'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'coxph'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'fitdist'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'fitdistr'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)
```

```
## S3 method for class 'glm'
useBIC(mod, return.K = FALSE, nobs = NULL, c.hat = 1,
...)

## S3 method for class 'glmmTMB'
useBIC(mod, return.K = FALSE, nobs = NULL, c.hat = 1,
...)

## S3 method for class 'gls'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'gnls'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'hurdle'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'lavaan'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'lm'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'lme'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'lmeKin'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'maxlikeFit'
useBIC(mod, return.K = FALSE, nobs = NULL, c.hat =
1, ...)

## S3 method for class 'mer'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'merMod'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'lmerModLmerTest'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'multinom'
useBIC(mod, return.K = FALSE, nobs = NULL, c.hat = 1,
...)

## S3 method for class 'nlme'
```

```

useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'nls'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'polr'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'rlm'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'survreg'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

## S3 method for class 'unmarkedFit'
useBIC(mod, return.K = FALSE, nobs = NULL, c.hat =
1, ...)

## S3 method for class 'vglm'
useBIC(mod, return.K = FALSE, nobs = NULL, c.hat = 1,
...)

## S3 method for class 'zeroinfl'
useBIC(mod, return.K = FALSE, nobs = NULL, ...)

```

Arguments

mod	an object of class aov, betareg, clm, clmm, clogit, coxme, coxph, fitdist, fitdistr, glm, glmmTMB, gls, gnls, hurdle, lavaan, lm, lme, lmekin, maxlikeFit, mer, merMod, lmerModLmerTest, multinom, nlme, nls, polr, rlm, survreg, vglm, zeroinfl, and various unmarkedFit classes containing the output of a model.
return.K	logical. If FALSE, the function returns the information criterion specified. If TRUE, the function returns K (number of estimated parameters) for a given model.
nobs	this argument allows to specify a numeric value other than total sample size to compute the BIC (i.e., nobs defaults to total number of observations). This is relevant only for mixed models or various models of unmarkedFit classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of nobs.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from c_hat. Note that values of c.hat different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or cbind(success, failure) syntax), with Poisson GLM's, single-season occupancy models (MacKenzie et al. 2002), dynamic occupancy models (MacKenzie et al. 2003), or N-mixture models (Royle 2004, Dail and Madsen 2011). If c.hat > 1, useBIC will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e.,

SE's are multiplied by `sqrt(c.hat)`). This option is not supported for generalized linear mixed models of the `mer` or `merMod` classes.

... additional arguments passed to the function.

Details

useBIC computes the Bayesian information criterion (BIC, Schwarz 1978):

$$BIC = -2 * \log - likelihood + K * \log(n),$$

where the log-likelihood is the maximum log-likelihood of the model, K corresponds to the number of estimated parameters, and n corresponds to the sample size of the data set.

In the presence of overdispersion, a quasi-likelihood analogue of the BIC (QBIC) will be computed, as

$$QBIC = \frac{-2 * \log - likelihood}{c - hat} + K * \log(n),$$

where *c-hat* is the overdispersion parameter specified by the user with the argument `c.hat`. Note that BIC or QBIC values are meaningful to select among `gls` or `lme` models fit by maximum likelihood. BIC or QBIC based on REML are valid to select among different models that only differ in their random effects (Pinheiro and Bates 2000).

Value

useBIC returns the BIC or the number of estimated parameters, depending on the values of the arguments.

Note

The actual (Q)BIC values are not really interesting in themselves, as they depend directly on the data, parameters estimated, and likelihood function. Furthermore, a single value does not tell much about model fit. Information criteria become relevant when compared to one another for a given data set and set of candidate models.

Author(s)

Marc J. Mazerolle

References

- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.
- MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.

Pinheiro, J. C., Bates, D. M. (2000) *Mixed-effect models in S and S-PLUS*. Springer Verlag: New York.

Royle, J. A. (2004) *N*-mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

Schwarz, G. (1978) Estimating the dimension of a model. *Annals of Statistics* **6**, 461–464.

See Also

[AICc](#), [bictab](#), [bictabCustom](#), [useBICCustom](#)

Examples

```
##cement data from Burnham and Anderson (2002, p. 101)
data(cement)
##run multiple regression - the global model in Table 3.2
glob.mod <- lm(y ~ x1 + x2 + x3 + x4, data = cement)

##compute BIC with full likelihood
useBIC(glob.mod, return.K = FALSE)

##compute BIC for mixed model on Orthodont data set in Pinheiro and
##Bates (2000)
## Not run:
require(nlme)
m1 <- lme(distance ~ age, random = ~1 | Subject, data = Orthodont,
          method= "ML")
useBIC(m1, return.K = FALSE)

## End(Not run)
```

useBICCustom

Custom Computation of BIC and QBIC from User-supplied Input

Description

This function computes the Bayesian information criterion (BIC) or a quasi-likelihood counterpart (QBIC) from user-supplied input instead of extracting the values automatically from a model object. This function is particularly useful for output imported from other software or for model classes that are not currently supported by useBIC.

Usage

```
useBICCustom(logL, K, return.K = FALSE, nobs = NULL, c.hat = 1)
```

Arguments

logL	the value of the model log-likelihood.
K	the number of estimated parameters in the model.
return.K	logical. If FALSE, the function returns the information criterion specified. If TRUE, the function returns K (number of estimated parameters) for a given model.
nobs	the sample size required to compute the BIC or QBIC.
c.hat	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from c_hat. Note that values of c.hat different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or cbind(success, failure) syntax), with Poisson GLM's, single-season or dynamic occupancy models (MacKenzie et al. 2002, 2003), <i>N</i> -mixture models (Royle 2004, Dail and Madsen 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If c.hat > 1, useBICCustom will return the quasi-likelihood analogue of the information criterion requested.

Details

useBICCustom computes one of the following two information criteria:

the Bayesian information criterion (BIC, Schwarz 1978) or the quasi-likelihood BIC (QBIC).

Value

useBICCustom returns the BIC or QBIC depending on the values of the c.hat argument.

Note

The actual (Q)BIC values are not really interesting in themselves, as they depend directly on the data, parameters estimated, and likelihood function. Furthermore, a single value does not tell much about model fit. Information criteria become relevant when compared to one another for a given data set and set of candidate models.

Author(s)

Marc J. Mazerolle

References

- Burnham, K. P., Anderson, D. R. (2002) *Model Selection and Multimodel Inference: a practical information-theoretic approach*. Second edition. Springer: New York.
- Dail, D., Madsen, L. (2011) Models for estimating abundance from repeated counts of an open population. *Biometrics* **67**, 577–587.
- Lebreton, J.-D., Burnham, K. P., Clobert, J., Anderson, D. R. (1992) Modeling survival and testing biological hypotheses using marked animals: a unified approach with case-studies. *Ecological Monographs* **62**, 67–118.
- MacKenzie, D. I., Nichols, J. D., Lachman, G. B., Droege, S., Royle, J. A., Langtimm, C. A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**, 2248–2255.

MacKenzie, D. I., Nichols, J. D., Hines, J. E., Knutson, M. G., Franklin, A. B. (2003) Estimating site occupancy, colonization, and local extinction when a species is detected imperfectly. *Ecology* **84**, 2200–2207.

Royle, J. A. (2004) N -mixture models for estimating population size from spatially replicated counts. *Biometrics* **60**, 108–115.

Schwarz, G. (1978) Estimating the dimension of a model. *Annals of Statistics* **6**, 461–464.

See Also

[AICc](#), [aictabCustom](#), [useBIC](#), [bictab](#), [evidence](#), [modavgCustom](#)

Examples

```
##cement data from Burnham and Anderson (2002, p. 101)
data(cement)
##run multiple regression - the global model in Table 3.2
glob.mod <- lm(y ~ x1 + x2 + x3 + x4, data = cement)

##extract log-likelihood
LL <- logLik(glob.mod)[1]

##extract number of parameters
##including residual variance
K.mod <- length(coef(glob.mod)) + 1

##compute BIC with full likelihood
useBICCustom(LL, K.mod, nobs = nrow(cement))
##compare against useBIC
useBIC(glob.mod)
```

xtable

Format Objects to LaTeX or HTML

Description

Functions to format various objects following model selection and multimodel inference to LaTeX or HTML tables. These functions extend the methods from the xtable package (Dahl 2014).

Usage

```
## S3 method for class 'aictab'
xtable(x, caption = NULL, label = NULL, align = NULL,
       digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, include.AICc = TRUE,
       include.LL = TRUE, include.Cum.Wt = FALSE, ...)

## S3 method for class 'anovaOD'
xtable(x, caption = NULL, label = NULL, align = NULL,
```

```
digits = NULL, display = NULL, auto = FALSE,
nice.names = TRUE, ...)
```

```
## S3 method for class 'bictab'
xtable(x, caption = NULL, label = NULL, align = NULL,
       digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, include.BIC = TRUE, include.LL = TRUE,
       include.Cum.Wt = FALSE, ...)
```

```
## S3 method for class 'boot.wt'
xtable(x, caption = NULL, label = NULL, align = NULL,
       digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, include.AICc = TRUE, include.AICcWt = FALSE, ...)
```

```
## S3 method for class 'countDist'
xtable(x, caption = NULL, label = NULL,
       align = NULL, digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, table.countDist = "distance", ...)
```

```
## S3 method for class 'checkParms'
xtable(x, caption = NULL, label = NULL,
       align = NULL, digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, include.variable = TRUE, include.max.se =
       TRUE, include.n.high.se = TRUE, ...)
```

```
## S3 method for class 'countHist'
xtable(x, caption = NULL, label = NULL,
       align = NULL, digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, table.countHist = "count", ...)
```

```
## S3 method for class 'detHist'
xtable(x, caption = NULL, label = NULL,
       align = NULL, digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, table.detHist = "freq", ...)
```

```
## S3 method for class 'detTime'
xtable(x, caption = NULL, label = NULL,
       align = NULL, digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, table.detTime = "freq", ...)
```

```
## S3 method for class 'dictab'
xtable(x, caption = NULL, label = NULL, align = NULL,
       digits = NULL, display = NULL, auto = FALSE,
       nice.names = TRUE, include.DIC = TRUE, include.Cum.Wt = FALSE, ...)
```

```
## S3 method for class 'ictab'
xtable(x, caption = NULL, label = NULL, align = NULL,
       digits = NULL, display = NULL, auto = FALSE, nice.names = TRUE,
```

```
include.IC = TRUE, include.Cum.Wt = FALSE, ...)  
  
## S3 method for class 'mb.chisq'  
xtable(x, caption = NULL, label = NULL, align = NULL,  
       digits = NULL, display = NULL, auto = FALSE, nice.names = TRUE,  
       include.detection.histories = TRUE, ...)  
  
## S3 method for class 'modavg'  
xtable(x, caption = NULL, label = NULL, align = NULL,  
       digits = NULL, display = NULL, auto = FALSE, nice.names = TRUE,  
       print.table = FALSE, ...)  
  
## S3 method for class 'modavgCustom'  
xtable(x, caption = NULL, label = NULL,  
       align = NULL, digits = NULL, display = NULL, auto = FALSE,  
       nice.names = TRUE, print.table = FALSE, ...)  
  
## S3 method for class 'modavgEffect'  
xtable(x, caption = NULL, label = NULL,  
       align = NULL, digits = NULL, display = NULL, auto = FALSE,  
       nice.names = TRUE, print.table = FALSE, ...)  
  
## S3 method for class 'modavgIC'  
xtable(x, caption = NULL, label = NULL, align = NULL,  
       digits = NULL, display = NULL, auto = FALSE, nice.names = TRUE,  
       print.table = FALSE, ...)  
  
## S3 method for class 'modavgPred'  
xtable(x, caption = NULL, label = NULL,  
       align = NULL, digits = NULL, display = NULL, auto = FALSE,  
       nice.names = TRUE, ...)  
  
## S3 method for class 'modavgShrink'  
xtable(x, caption = NULL, label = NULL,  
       align = NULL, digits = NULL, display = NULL, auto = FALSE,  
       nice.names = TRUE, print.table = FALSE, ...)  
  
## S3 method for class 'multComp'  
xtable(x, caption = NULL, label = NULL,  
       align = NULL, digits = NULL, display = NULL, auto = FALSE,  
       nice.names = TRUE, print.table = FALSE, ...)  
  
## S3 method for class 'summaryOD'  
xtable(x, caption = NULL, label = NULL, align = NULL,  
       digits = NULL, display = NULL, auto = FALSE,  
       nice.names = TRUE, ...)
```

Arguments

<code>x</code>	an object of class <code>aictab</code> , <code>anovaOD</code> , <code>bictab</code> , <code>boot.wt</code> , <code>checkParms</code> , <code>countDist</code> , <code>countHist</code> , <code>detHist</code> , <code>detTime</code> , <code>dictab</code> , <code>ictab</code> , <code>mb.chisq</code> , <code>modavg</code> , <code>modavgEffect</code> , <code>modavgCustom</code> , <code>modavgIC</code> , <code>modavgPred</code> , <code>modavgShrink</code> , <code>multComp</code> , or <code>summaryOD</code> .
<code>caption</code>	a character vector of length 1 or 2 storing the caption or title of the table. If the vector is of length 2, the second item is the short caption used when LaTeX generates a list of tables. The default value is <code>NULL</code> and suppresses the caption.
<code>label</code>	a character vector storing the LaTeX label or HTML anchor. The default value is <code>NULL</code> and suppresses the label.
<code>align</code>	a character vector of length equal to the number of columns of the table specifying the alignment of the elements. Note that the rownames are considered as an additional column and require an alignment value.
<code>digits</code>	a numeric vector of length one or equal to the number of columns in the table (including the rownames) specifying the number of digits to display in each column.
<code>display</code>	a character vector of length equal to the number of columns (including the rownames) specifying the format of each column. For example, use <code>s</code> for strings, <code>f</code> for numbers in the regular format, or <code>d</code> for integers. See <code>formatC</code> for additional possible values.
<code>auto</code>	Logical, indicating whether to apply automatic format when no value is passed to <code>align</code> , <code>digits</code> , or <code>display</code> . This <code>autoformat</code> (based on <code>xalign</code> , <code>xdigits</code> , and <code>xdisplay</code>) can be useful to quickly format a typical matrix or <code>data.frame</code> . Default value is <code>FALSE</code> .
<code>nice.names</code>	logical. If <code>TRUE</code> , column labels are modified to improve their appearance in the table. If <code>FALSE</code> , simpler labels are used, or the ones supplied directly by the user in the object storing the output.
<code>include.AICc</code>	logical. If <code>TRUE</code> , the column containing the information criterion (AIC, AICc, QAIC, or QAICc) of each model is printed in the table. If <code>FALSE</code> , the column is suppressed.
<code>include.BIC</code>	logical. If <code>TRUE</code> , the column containing the information criterion (BIC or QBIC) of each model is printed in the table. If <code>FALSE</code> , the column is suppressed.
<code>include.DIC</code>	logical. If <code>TRUE</code> , the column containing the deviance information criterion (DIC) of each model is printed in the table. If <code>FALSE</code> , the column is suppressed.
<code>include.IC</code>	logical. If <code>TRUE</code> , the column containing the information criterion of each model is printed in the table. If <code>FALSE</code> , the column is suppressed.
<code>include.LL</code>	logical. If <code>TRUE</code> , the column containing the log-likelihood of each model is printed in the table. If <code>FALSE</code> , the column is suppressed.
<code>include.Cum.Wt</code>	logical. If <code>TRUE</code> , the column containing the cumulative Akaike weights is printed in the table. If <code>FALSE</code> , the column is suppressed.
<code>include.AICcWt</code>	logical. If <code>TRUE</code> , the column containing the Akaike weight of each model is printed in the table. If <code>FALSE</code> , the column is suppressed.
<code>include.detection.histories</code>	logical. If <code>TRUE</code> , the column containing detection histories is printed in the table. If <code>FALSE</code> , the column is suppressed.

<code>include.variable</code>	logical. If TRUE, the column containing the variable name is printed in the table. If FALSE, the column is suppressed.
<code>include.max.se</code>	logical. If TRUE, the column containing the maximum SE in the model is printed in the table. If FALSE, the column is suppressed.
<code>include.n.high.se</code>	logical. If TRUE, the column containing the number of SE's greater than the threshold specified by the user is printed in the table. If FALSE, the column is suppressed.
<code>print.table</code>	logical. If TRUE, the model selection table is printed and other sections of the output are suppressed (e.g., model-averaged estimates). If FALSE, the model selection table is suppressed and only the other portion of the output is printed in the table.
<code>table.detHist</code>	character string specifying, either "freq", "prop", or "hist". If <code>table.type = "freq"</code> , the function returns a table of frequencies of sites sampled, of sites with at least one detection, and for data with multiple primary periods, the frequencies of sites with observed extinctions and colonizations. If <code>table.type = "prop"</code> , the table returns the proportion of sites with at least one detection, and for data with multiple periods, the proportion of sites with observed extinctions and colonizations. If <code>table.type = "hist"</code> , the function returns the frequencies of each observed detection history.
<code>table.detTime</code>	character string specifying, either "freq", "prop", or "dist". If <code>table.type = "freq"</code> , the function returns a table of frequencies of sites sampled, of sites with at least one detection, and for data with multiple primary periods, the frequencies of sites with observed extinctions and colonizations. If <code>table.type = "prop"</code> , the table returns the proportion of sites with at least one detection, and for data with multiple periods, the proportion of sites with observed extinctions and colonizations. If <code>table.type = "dist"</code> , the function returns the quantiles of detection times.
<code>table.countDist</code>	character string specifying, either "distance", "count", "freq", or "prop". If <code>table.type = "distance"</code> , the function returns a table of counts summarized for each distance class. If <code>table.type = "count"</code> , the function returns the table of frequencies of counts observed across sites. If <code>table.type = "freq"</code> , the function returns a table of frequencies of sites sampled, of sites with at least one detection, and for data with multiple primary periods, the frequencies of sites with observed extinctions and colonizations. If <code>table.type = "prop"</code> , the table returns the proportion of sites with at least one detection, and for data with multiple periods, the proportion of sites with observed extinctions and colonizations.
<code>table.countHist</code>	character string specifying, either "count", "freq", "prop", or "hist". If <code>table.type = "count"</code> , the function returns the table of frequencies of counts observed across sites. If <code>table.type = "freq"</code> , the function returns a table of frequencies of sites sampled, of sites with at least one detection, and for data with multiple primary periods, the frequencies of sites with observed extinctions and colonizations. If <code>table.type = "prop"</code> , the table returns the proportion of

sites with at least one detection, and for data with multiple periods, the proportion of sites with observed extinctions and colonizations. If `table.type = "hist"`, the function returns the frequencies of each observed count history.

... additional arguments passed to the function.

Details

`xtable` creates an object of the `xtable` class inheriting from the `data.frame` class. This object can then be used with `print.xtable` for added flexibility such as suppressing row names, modifying caption placement, and format tables in LaTeX or HTML format.

Author(s)

Marc J. Mazerolle

References

Dahl, D. B. (2014) `xtable`: Export tables to LaTeX or HTML. R package version 1.7-3. <https://cran.r-project.org/package=xtable>.

See Also

[aictab](#), [boot.wt](#), [dictab](#), [formatC](#), [ictab](#), [mb.chisq](#), [modavg](#), [modavgCustom](#), [modavgIC](#), [modavgEffect](#), [modavgPred](#), [modavgShrink](#), [multComp](#), [summaryOD](#), [anovaOD](#), [xtable](#), [print.xtable](#)

Examples

```
if(require(xtable)) {
##model selection example
data(dry.frog)
##setup candidate models
Cand.models <- list( )
Cand.models[[1]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2,
  data = dry.frog)
Cand.models[[2]] <- lm(log_Mass_lost ~ Shade + Substrate +
  cent_Initial_mass + Initial_mass2 +
  Shade:Substrate, data = dry.frog)
Cand.models[[3]] <- lm(log_Mass_lost ~ cent_Initial_mass +
  Initial_mass2, data = dry.frog)
Model.names <- c("additive", "interaction", "no shade")

##model selection table - AICc
out <- aictab(cand.set = Cand.models, modnames = Model.names)

xtable(out)
##exclude AICc and LL
xtable(out, include.AICc = FALSE, include.LL = FALSE)
##remove row names and add caption
print(xtable(out, caption = "Model selection based on AICc"),
  include.rownames = FALSE, caption.placement = "top")
}
```

```

##model selection table - BIC
out2 <- bictab(cand.set = Cand.models, modnames = Model.names)

xtable(out2)
##exclude AICc and LL
xtable(out2, include.BIC = FALSE, include.LL = FALSE)
##remove row names and add caption
print(xtable(out2, caption = "Model selection based on BIC"),
      include.rownames = FALSE, caption.placement = "top")

##model-averaged estimate of Initial_mass2
mavg.mass <- modavg(cand.set = Cand.models, parm = "Initial_mass2",
                  modnames = Model.names)
##model-averaged estimate
xtable(mavg.mass, print.table = FALSE)
##table with contribution of each model
xtable(mavg.mass, print.table = TRUE)

##model-averaged predictions for first 10 observations
preds <- modavgPred(cand.set = Cand.models, modnames = Model.names,
                  newdata = dry.frog[1:10, ])
xtable(preds)
}

##example of diagnostics
## Not run:
if(require(unmarked)){
##distance sampling example from ?distsamp
data(linetran)
ltUMF <- with(linetran, {
  unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
    siteCovs = data.frame(Length, area, habitat),
    dist.breaks = c(0, 5, 10, 15, 20),
    tlength = linetran$Length * 1000, survey = "line",
    unitsIn = "m")
  })

##summarize counts across distance classes
xtable(countDist(ltUMF), table.countDist = "distance")
##summarize counts across all sites
xtable(countDist(ltUMF), table.countDist = "count")

##Half-normal detection function
fm1 <- distsamp(~ 1 ~ 1, ltUMF)
##determine parameters with highest SE's
xtable(checkParms(fm1))
}

```

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

Index

* datasets

beetle, 40
bullfrog, 56
calcium, 57
cement, 58
dry.frog, 93
fat, 110
gpa, 112
iron, 123
lizards, 124
min.trap, 132
newt, 196
pine, 201
salamander, 205
tortoise, 211
turkey, 212

* models

AICc, 11
AICcCustom, 16
AICcmodavg-defunct, 18
AICcmodavg-package, 3
aictab, 23
aictabCustom, 32
anovaOD, 35
bictab, 41
bictabCustom, 49
boot.wt, 52
c_hat, 79
checkConv, 59
checkParms, 61
confset, 65
countDist, 69
countHist, 73
covDiag, 77
detHist, 82
detTime, 86
DIC, 89
dictab, 91
evidence, 95

extractCN, 97
extractLL, 99
extractSE, 101
extractX, 102
fam.link.mer, 108
ictab, 113
importance, 115
mb.gof.test, 127
modavg, 133
modavg.utility, 146
modavgCustom, 148
modavgEffect, 151
modavgIC, 165
modavgPred, 168
modavgShrink, 179
multComp, 189
Nmix.gof.test, 197
predictSE, 202
summaryOD, 206
useBIC, 213
useBICCustom, 217
xtable, 219

AICc, 4, 11, 18, 30, 55, 68, 81, 96, 100, 120,
131, 142, 159, 174, 187, 200, 217,
219

AICc.mult (AICcmodavg-defunct), 18
AICc.unmarked (AICcmodavg-defunct), 18
AICcCustom, 4, 16, 16, 34, 51, 90, 150
AICcmodavg (AICcmodavg-package), 3
AICcmodavg-defunct, 18
AICcmodavg-package, 3
aictab, 4, 16, 22, 23, 47, 68, 90, 92, 96, 100,
120, 142, 147, 159, 174, 187, 194,
224

aictab.clm (AICcmodavg-defunct), 18
aictab.climm (AICcmodavg-defunct), 18
aictab.coxph (AICcmodavg-defunct), 18
aictab.glm (AICcmodavg-defunct), 18
aictab.gls (AICcmodavg-defunct), 18

- aictab.lm (AICcmodavg-defunct), 18
- aictab.lme (AICcmodavg-defunct), 18
- aictab.mer (AICcmodavg-defunct), 18
- aictab.merMod (AICcmodavg-defunct), 18
- aictab.mult (AICcmodavg-defunct), 18
- aictab.nlme (AICcmodavg-defunct), 18
- aictab.nls (AICcmodavg-defunct), 18
- aictab.polr (AICcmodavg-defunct), 18
- aictab.rlm (AICcmodavg-defunct), 18
- aictab.unmarked (AICcmodavg-defunct), 18
- aictabCustom, 4, 18, 30, 32, 51, 92, 114, 150, 167, 219
- anova0D, 5, 35, 81, 147, 210, 224
- beetle, 40
- bictab, 4, 30, 41, 96, 217, 219
- bictabCustom, 5, 34, 47, 49, 150, 217
- boot, 204
- boot.wt, 4, 52, 224
- bullfrog, 56
- c_hat, 5, 16, 18, 22, 30, 34, 39, 51, 55, 64, 68, 79, 96, 98, 120, 131, 142, 159, 174, 187, 194, 200, 210
- calcium, 57
- cement, 58
- checkConv, 5, 59, 64
- checkParms, 5, 61, 61
- colext, 100, 131
- confset, 4, 16, 18, 22, 30, 34, 47, 51, 55, 65, 81, 90, 92, 96, 114, 120, 142, 159, 174, 187, 194
- countDist, 5, 64, 69, 76, 85, 88
- countHist, 5, 64, 72, 73, 85, 88
- covDiag, 5, 61, 72, 76, 77, 85
- coxme, 100, 101
- coxph, 100
- detHist, 5, 64, 72, 76, 82, 88
- detTime, 5, 72, 76, 85, 86
- DIC, 4, 89, 92
- dictab, 4, 22, 90, 91, 224
- dictab.bugs (AICcmodavg-defunct), 18
- dictab.rjags (AICcmodavg-defunct), 18
- distsamp, 100
- dry.frog, 93
- evidence, 4, 16, 18, 22, 30, 34, 47, 51, 55, 68, 81, 90, 92, 95, 114, 120, 131, 142, 159, 174, 187, 194, 200, 219
- extract.LL (AICcmodavg-defunct), 18
- extractCN, 5, 64, 97, 106
- extractLL, 5, 22, 99
- extractSE, 5, 101, 106
- extractX, 5, 102
- fam.link.mer, 5, 108
- fat, 110
- fit.contrast, 194
- formatC, 224
- formatCands (modavg.utility), 146
- formulaShort (modavg.utility), 146
- gdistsamp, 100
- glht, 194
- glmer, 101, 109, 204
- gls, 204
- gpa, 112
- ictab, 5, 34, 51, 113, 167, 224
- importance, 4, 16, 22, 30, 47, 55, 68, 81, 96, 115, 131, 142, 159, 174, 187, 200
- iron, 123
- kappa, 98
- lizards, 124
- lme, 204
- lmekin, 100, 101
- lmer, 101, 109
- maxlike, 100
- mb.chisq, 85, 224
- mb.chisq (mb.gof.test), 127
- mb.gof.test, 5, 39, 61, 64, 81, 85, 98, 127, 200, 210
- min.trap, 132
- modavg, 4, 16, 22, 30, 55, 68, 78, 81, 96, 101, 109, 120, 131, 133, 147, 150, 167, 174, 187, 200, 224
- modavg.clm (AICcmodavg-defunct), 18
- modavg.clm (AICcmodavg-defunct), 18
- modavg.coxph (AICcmodavg-defunct), 18
- modavg.effect (AICcmodavg-defunct), 18
- modavg.glm (AICcmodavg-defunct), 18
- modavg.gls (AICcmodavg-defunct), 18
- modavg.lme (AICcmodavg-defunct), 18
- modavg.mer (AICcmodavg-defunct), 18
- modavg.merMod (AICcmodavg-defunct), 18
- modavg.mult (AICcmodavg-defunct), 18

- modavg.polr (AICcmodavg-defunct), 18
- modavg.rlm (AICcmodavg-defunct), 18
- modavg.shrink (AICcmodavg-defunct), 18
- modavg.unmarked (AICcmodavg-defunct), 18
- modavg.utility, 146
- modavgCustom, 5, 18, 34, 51, 106, 114, 142, 148, 167, 174, 187, 219, 224
- modavgEffect, 4, 22, 30, 106, 142, 151, 174, 224
- modavgIC, 5, 114, 150, 165, 224
- modavgPred, 4, 16, 22, 30, 55, 68, 78, 81, 96, 106, 109, 120, 131, 142, 147, 150, 159, 167, 168, 187, 200, 224
- modavgpred (AICcmodavg-defunct), 18
- modavgShrink, 4, 16, 22, 30, 55, 68, 96, 120, 142, 147, 150, 159, 167, 174, 179, 224
- mult.comp (AICcmodavg-defunct), 18
- multComp, 4, 22, 189, 224
- newt, 196
- Nmix.chisq, 72, 76, 78, 88
- Nmix.chisq (Nmix.gof.test), 197
- Nmix.gof.test, 5, 39, 61, 64, 72, 76, 78, 81, 88, 98, 131, 197, 210
- nonparboot, 204
- occu, 100, 131
- occuRN, 100
- parboot, 64, 98, 131, 200, 204
- pcount, 78, 100, 200, 204
- pcountOpen, 100, 200, 204
- pine, 201
- predict, 106, 174
- predictSE, 5, 22, 78, 106, 109, 174, 202
- predictSE.zip (AICcmodavg-defunct), 18
- print.aictab (aictab), 23
- print.anovaOD (anovaOD), 35
- print.bictab (bictab), 41
- print.boot.wt (boot.wt), 52
- print.c_hat (c_hat), 79
- print.checkConv (checkConv), 59
- print.checkParms (checkParms), 61
- print.confset (confset), 65
- print.countDist (countDist), 69
- print.countHist (countHist), 73
- print.covDiag (covDiag), 77
- print.detHist (detHist), 82
- print.detTime (detTime), 86
- print.dictab (dictab), 91
- print.evidence (evidence), 95
- print.extractCN (extractCN), 97
- print.extractX (extractX), 102
- print.ictab (ictab), 113
- print.importance (importance), 115
- print.mb.chisq (mb.gof.test), 127
- print.modavg (modavg), 133
- print.modavgCustom (modavgCustom), 148
- print.modavgEffect (modavgEffect), 151
- print.modavgIC (modavgIC), 165
- print.modavgPred (modavgPred), 168
- print.modavgShrink (modavgShrink), 179
- print.multComp (multComp), 189
- print.Nmix.chisq (Nmix.gof.test), 197
- print.summaryOD (summaryOD), 206
- print.xtable, 224
- rcond, 98
- reverse.exclude (modavg.utility), 146
- reverse.parm (modavg.utility), 146
- salamander, 205
- simulate.merMod, 204
- summaryOD, 5, 39, 81, 206, 224
- tortoise, 211
- turkey, 212
- useBIC, 4, 16, 47, 213, 219
- useBICCustom, 5, 217, 217
- xtable, 5, 219, 224