# Package 'AICcmodavg'

February 11, 2016

**Type** Package

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

**Version** 2.0-4

**Date** 2016-02-10

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

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

**Depends** R (>= 3.0.0)

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

**Suggests** betareg, coxme, fitdistrplus, lavaan, lme4, maxlike, nnet, ordinal, pscl, R2jags, R2OpenBUGS, R2WinBUGS

**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 'unmarkedFit' 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.

**License** GPL (>= 2)

**LazyLoad** yes

**Repository** CRAN

**NeedsCompilation** no

**Date/Publication** 2016-02-11 16:52:31

## R topics documented:

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 works with Cox proportional hazards models and conditional logistic regression (`coxph` and `coxme` classes), linear models (`lm` class), generalized linear models (`glm`, `vglm`, `hurdle`, and `zeroinfl` classes), linear models fit by generalized least squares (`gls` class), linear mixed models (`lme` class), generalized linear mixed models (`mer` and `merMod` 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: | 2.0-4 |
| Date: | 2016-02-10 |
| License: | GPL (>=2 ) |
| LazyLoad: | yes |

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

- `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 other AICc, QAIC, and QAICc 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.

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.
- 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.
- 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:

- extractLL Extracts log-likelihood from models of certain classes.
- extractSE Extracts standard errors from models of certain classes and adds the labels.
- 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.
- xtable Formats 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

```
##anuran larvae example from Mazerolle (2006) - Poisson GLM with offset
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]], 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

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

```
##model selection table based on AICc
aictab(cand.set = Cand.mod, modnames = Modnames)

##compute evidence ratio
evidence(aictab(cand.set = Cand.mod, modnames = Modnames))

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

##compute importance value for "TypeBOG" - same number of models
##with vs without variable
importance(cand.set = Cand.mod, modnames = Modnames, parm = "TypeBOG")

##compute model-averaged estimate of "TypeBOG"
modavg(cand.set = Cand.mod, modnames = Modnames, parm = "TypeBOG")

##compute model-averaged estimate of "TypeBOG" with shrinkage
##same number of models with vs without variable
modavgShrink(cand.set = Cand.mod, modnames = Modnames,
             parm = "TypeBOG")

##compute model-average predictions for two types of ponds
##create a data set for predictions
dat.pred <- data.frame(Type = factor(c("BOG", "UPLAND")),
                       log.Perimeter = mean(min.trap$log.Perimeter),
                       Num_ranatra = mean(min.trap$Num_ranatra),
                       Effort = mean(min.trap$Effort))

##model-averaged predictions across entire model set
modavgPred(cand.set = Cand.mod, modnames = Modnames,
           newdata = dat.pred, type = "response")

##compute model-averaged effect size between two groups
##works when data set has two rows
modavgEffect(cand.set = Cand.mod, modnames = Modnames,
             newdata = dat.pred, type = "link")


##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)))
```

```
##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)
Modnames <- c("fm1", "fm2", "fm3", "fm4")

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

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

##compute confidence set based on 'raw' method
confset(cand.set = Cand.models, modnames = Modnames, 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, modnames = Modnames, parm = "sitevar1",
           parm.type = "psi")

##compute model-averaged estimate of "sitevar1" on occupancy
modavg(cand.set = Cand.models, modnames = Modnames, parm = "sitevar1",
       parm.type = "psi")

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

##compute model-average predictions for two types of ponds
##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, modnames = Modnames,
           newdata = dat.pred, parm.type = "psi")
detach(package:unmarked)
```

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

---

AICc *Computing AIC, AICc, QAIC, and QAICc*

---

## Description

Functions to computes 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 'gls'
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 'multinom'
AICc(mod, return.K = FALSE, second.ord = TRUE,
     nobs = NULL, c.hat = 1, ...)

## 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, gls, gnls, hurdle, lavaan, lm, lme, lmekin, maxlikeFit, mer, merMod, 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. |
| 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. |

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, 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.

... — additional arguments passed to the function.

## Details

AICc 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 1991), the quasi-likelihood AIC (QAIC, Burnham and Anderson 2002), and the quasi-likelihood AICc (QAICc, Burnham and Anderson 2002). 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. (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

## 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 *Custom Computation of 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.

## 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 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), *N*-mixture models (Royle 2004, Dail and Madsen 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If c.hat > 1, AICc will return the quasi-likelihood analogue of the information criterion requested. |

## Details

AICc 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 1991), the quasi-likelihood AIC (QAIC, Burnham and Anderson 2002), and the quasi-likelihood AICc (QAICc, Burnham and Anderson 2002).

## 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.

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. (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.AICsclm.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.AICgls.

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.AICunmarkedFitMP 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.AICsclm.clm.

modavg.clmm has been replaced by modavg.AICsclm.clm.

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.AICunmarkedFitMP 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.AICunmarkedFitC 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.AICsclm.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.AICgls.

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.AICunmarkedFitC modavgShrink.AICunmarkedFitOccuRN, modavgShrink.AICunmarkedFitPCount, 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.AICgls.

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.AICunmarkedFitColExt, modavgPred.AICunmarkedFitOccuRN, modavgPred.AICunmarkedFitPCount, 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, gls, gnls, hurdle, lavaan, lm, lme, lmekin, maxlikeFit, mer, merMod, 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

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

## S3 method for class 'AICnls'
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 'AICrlm.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 'AICunmarkedFitPCount'
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 '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

| | |
|---|---|
| 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 (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 | 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. |
| 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 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, 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. |
| ... | 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, gls, gnls, hurdle, lavaan, lm, lme, lmekin, maxlikeFit, mer, merMod, 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 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 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 (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*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. |
| 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, 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 *Custom Creation of Model Selection Tables from User-supplied Input*

---

## 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), *N*-mixture models (Royle 2004, Dail and Madsen 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If c.hat > 1, AICc 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 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*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, confset, c_hat, evidence, 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)
```

---

`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) ...
```

---

# 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`, `rlm`, `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 'AICsclm.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 'AICrlm.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, AICc 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*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).

## 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) ...
```

---

# 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

| | |
|---|---|
| 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 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. |
| method | a character value, either as raw, ordinal, or ratio, indicating the method for determining the confidence set for the best model (see 'Description' above for details). |
| level | the level of confidence (i.e., sum of model probabilities) used to determine the confidence set on the best model when using the raw method. Note that the argument is not used for the other methods of determining the confidence set on the best model. |
| delta | 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 method = ratio. |
| 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, 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. |

## 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 (delta (Q)AIC(c) <= 2), some support (2 < delta (Q)AIC(c) <= 7), little support (7 < delta (Q)AIC(c) <= 10), no support (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) <= 2. |
| `some` | a reduced table with the models included in the confidence set for which 2 < delta (Q)AIC(c) <= 7. |
| `little` | a reduced table with the models included in the confidence set for which 7 < delta (Q)AIC(c) <= 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)
```

---

countDist *Compute Summary Statistics from Distance Sampling Data*

---

## 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, ...)

## S3 method for class 'unmarkedFrameDS'
countDist(object, plot.freq = TRUE,
                  plot.distance = TRUE, ...)

## S3 method for class 'unmarkedFitDS'
countDist(object, plot.freq = TRUE,
                  plot.distance = TRUE, ...)

## S3 method for class 'unmarkedFrameGDS'
countDist(object, plot.freq = TRUE,
                  plot.distance = TRUE, ...)

## S3 method for class 'unmarkedFitGDS'
countDist(object, plot.freq = TRUE,
                  plot.distance = TRUE, ...)
```

## Arguments

| | |
|---|---|
| object | an object of various unmarkedFrame or unmarkedFit classes containing distance sampling data. |
| plot.freq | 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 should be plotted. |
| ... | 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) and Chandler et al. (2011).

countDist can take data frames of the unmarkedFrameDS and unmarkedFrameGDS classes as input. For convenience, the function can also extract the raw data from model objects of classes unmarkedFitDS and unmarkedFitGDS. 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.

`hist.table.seasons`
: a list of tables with the frequency of counts in each distance class for each primary period.

`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.

## 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.

Royle, J. A., Dawson, D. K., Bates, S. (2004) Modeling abundance effects in distance sampling. *Ecology* **85**, 1591–1597.

## See Also

`covDiag`, `detHist`, `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, ...)

## S3 method for class 'unmarkedFramePCount'
countHist(object, plot.freq = TRUE, ...)

## S3 method for class 'unmarkedFitPCount'
countHist(object, plot.freq = TRUE, ...)

## S3 method for class 'unmarkedFrameGPC'
countHist(object, plot.freq = TRUE, ...)

## S3 method for class 'unmarkedFitGPC'
countHist(object, plot.freq = TRUE, ...)
```

```
## S3 method for class 'unmarkedFrameMPois'
countHist(object, plot.freq = TRUE, ...)

## S3 method for class 'unmarkedFitMPois'
countHist(object, plot.freq = TRUE, ...)

## S3 method for class 'unmarkedFramePCO'
countHist(object, plot.freq = TRUE,
                  plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFitPCO'
countHist(object, plot.freq = TRUE,
                  plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFrameGMM'
countHist(object, plot.freq = TRUE,
                  plot.seasons = FALSE, ...)

## S3 method for class 'unmarkedFitGMM'
countHist(object, plot.freq = TRUE,
                  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. |
| `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` classes as input. For convenience, the function can also extract the raw data from model objects of classes `unmarkedFitPCount`, `unmarkedFitGPC`, `unmarkedFitMPois`, `unmarkedFitPCO`, and `unmarkedFitGMM`. 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.

## 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`, `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 spar` 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 'merMod'
c_hat(mod, method = "pearson", ...)

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

## Arguments

| | |
|---|---|
| mod | an object of class glm, 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 « 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

## 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 invidual 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 'unmarkedFrameOccu'
detHist(object, ...)

## S3 method for class 'unmarkedFrameOccuFP'
```

```
detHist(object, ...)

## S3 method for class 'unmarkedMultFrame'
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), and false-positive occupancy models (Royle and Link 2006, Miller et al. 2011).

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

## Value

detHist returns a list with the following components:

| | |
|---|---|
| hist.table.full | a table with the frequency of each observed detection history. |
| hist.table.seasons | a list of tables with the frequency of each season-specific detection history. |
| 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. |

## 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.

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.

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, 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)
```

---

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, ...)
```

## Arguments

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

## Details

DIC is implemented for bugs and rjags 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` and `rjags` 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, ...)
```

## 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. |
| `sort` | logical. If `TRUE`, the model selection table is ranked according to the DIC values. |
| `...` | 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` and `jags` 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 names 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*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 inital 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` *Compute Evidence Ratio Between Two Models*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| aic.table | a model selection table of class aictab such as that produced by aictab or of class boot.wt as produced by boot.wt. The table may be sorted or not, as the function sorts the table internally. |
| model.high | the top-ranked model (default), or alternatively, the name of another model as it appears in the model selection table. |
| model.low | 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 Akaike 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 name of the required model must be specified in the model.high argument.

## Value

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

| | |
|---|---|
| Model.high | the model specified in model.high. |
| Model.low | the model specified in model.low. |
| Ev.ratio | 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, 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 <- aictab(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)


##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

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

## Details

The condition number ($\kappa$) is a measure of the transfer of error to the solution in response to small changes in the input (Cheney and Kincaid 2008). In this implementation, the condition number is computed on the Hessian matrix of models of `unmarkedFit` classes from the `optim` results stored in the model object. The condition number is defined as the ratio of the largest to the smallest non-negative singular values of a given matrix (Cline et al. 1979, Dixon 1983). In the special case of positive semi-definite matrices, the singular values are equal to the eigenvalues (Ruhe 1975).

Large values of the condition number may indicate problems in estimating parameters or their variance (ill-conditioning), possibly due to a model having too many parameters for the given data set. Cheney and Ward (2008) suggest using the $\log_{10}(\kappa)$ of the condition number as a crude estimate of the number of digits of precision lost.

## Value

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

| | |
|---|---|
| `CN` | the condition number ($\kappa$) of the model. |
| `log10` | the log base 10 of the condition number. |
| `method` | the method used to extract the singular values or eigenvalues. |

## Author(s)

Marc J. Mazerolle

## References

Cheney, W., Kincaid, D. (2008) *Numerical mathematics and computing*. Sixth edition. Thomson Brooks/Cole: Belmont.

Cline, A. K., Moler, C. B., Stewart, G. W., Wilkinson, J. H. (1979) An estimate for the condition number of a matrix. *SIAM Journal on Numerical Analysis* **16**, 368–375.

Dixon, J. D. (1983) Estimating extremal eigenvalues and condition numbers of matrices. *SIAM Journal on Numerical Analysis* **20**, 812–814.

Ruhe, A. (1975) On the closeness of eigenvalues and singular values for almost normal matrices. *Linear Algebra and its Applications* **11**, 87–94.

## See Also

c_hat, mb.gof.test, Nmix.gof.test, parboot, kappa, rcond

## 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 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

| | |
|---|---|
| mod | an object of coxme, coxph, lmekin, maxlikeFit, vglm, or unmarkedFit class resulting from the fit of distsamp, gdistsamp, gmultmix, multinomPois, gpcount, occu, occuRN, colext, pcount, or pcountOpen. |
| ... | 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` *Extract SE of Fixed Effects of* `coxme`*,* `glmer`*, and* `lmekin` *Fit*

---

## Description

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

## Usage

```
extractSE(mod, ...)

## S3 method for class 'coxme'
extractSE(mod, ...)

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

## S3 method for class 'mer'
extractSE(mod, ...)

## S3 method for class 'merMod'
extractSE(mod, ...)
```

## Arguments

| | |
|---|---|
| mod | an object of coxme, lmekin, mer or merMod class. |
| ... | additional arguments passed to the function. |

## Details

These are extractor functions that use vcov.coxme, vcov.lmekin, vcov.mer, and vcov.merMod. Some of these functions are called by modavg, 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, lmekin

## 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)
```

`fam.link.mer` *Extract Distribution Family and Link Function*

## 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:

| | |
|---|---|
| `family` | the family of the distribution of the model. |
| `link` | the link function of the model. |
| `supp.link` | a character value indicating whether the link function used is supported by `predictSE` and `modavgPred`. |

## 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)
```

`importance` *Compute Importance Values of Variable*

## 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 'AICsclm.clm'
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 '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 '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 'AICrlm.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 '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

| | |
|---|---|
| cand.set | a list storing each of the models in the candidate model set. |
| parm | the parameter of interest for which a measure of relative importance is required. |
| 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. |
| 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, 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. |
| parm.type | this argument specifies the parameter type on which the effect size will be computed and is only relevant for models of unmarkedFitOccu, unmarkedFitColExt, unmarkedFitOccuFP, unmarkedFitOccuRN, unmarkedFitMPois, unmarkedFitGPC, unmarkedFitPCount, unmarkedFitPCO, unmarkedFitDS, unmarkedFitGDS, and unmarkedFitGMM classes. The character strings supported vary with the type of model fitted. For unmarkedFitOccu objects, either psi or detect can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For unmarkedFitColExt, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the inital year, colonization, extinction, and detectability, respectively. For unmarkedFitOccuFP objects, one can specify psi, detect, or fp, for occupancy, detectability, and probability of assigning false-positives, 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 objects, one can enter lambda, gamma, omega, or detect, to specify parameters on abundance, recruitment, apparent survival, and detectability, respectively. For unmarkedFitDS objects, only lambda is supported for the moment. For unmarkedFitGDS objects, lambda and phi denote abundance and availability, respectively. For unmarkedFitGMM and unmarkedFitGPC objects, |

| | |
|---|---|
| | `lambda`, `phi`, and `detect` denote abundance, availability, and detectability, respectively. |
| `...` | additional arguments passed to the function. |

## Value

`importance` returns an object of class `importance` consisting of the following components:

| | |
|---|---|
| `parm` | the parameter for which an importance value is required. |
| `w.plus` | the parameter for which an importance value is required. |
| `w.minus` | 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")
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 and Dynamic 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) 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, ...)

mb.gof.test(mod, nsim = 5, plot.hist = TRUE, ...)

## S3 method for class 'unmarkedFitOccu'
mb.gof.test(mod, nsim = 5, plot.hist = TRUE,
                    ...)

## S3 method for class 'unmarkedFitColExt'
mb.gof.test(mod, nsim = 5, plot.hist = TRUE,
                    plot.seasons = FALSE, ...)
```

## Arguments

| | |
|---|---|
| mod | the model for which a goodness-of-fit test is required. |
| print.table | logical. Specifies if the detailed table of observed and expected values is to be included in the output. |
| nsim | the number of bootstrapped samples. |
| plot.hist | 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. |
| plot.seasons | 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. |
| ... | 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 accomodated by creating cohorts for each pattern of missing values.

It is also possible to obtain an estimate of the overdispersion parameter (c-hat) 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 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 and the same value of c-hat 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 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 generate 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 « 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 occupancy models:

| | |
|---|---|
| `chisq.table` | the table of observed and expected values for each detection history and its chi-square component (if `print.table = TRUE`). 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. |
| `chi.square` | 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. |
| `model.type` | the model type, either `single-season` or `dynamic`. |

`mb.chisq` returns the following additional components for dynamic occupancy models:

| | |
|---|---|
| `tables` | a list containing the season-specific chi-square tables (if `print.table = TRUE`). |
| `all.chisq` | an element containing the season-specific chi-squares. |
| `n.seasons` | the number of primary periods (seasons). |

`mb.gof.test` returns the following components for single-season occupancy models:

| | |
|---|---|
| `chisq.table` | the table of observed and expected values for each detection history and its chi-square component. |
| `chi.square` | the Pearson chi-square statistic. |
| `t.star` | the bootstrapped chi-square test statistics (i.e., obtained for each of the simulated data sets). |
| `p.value` | 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. |
| `c.hat.est` | the estimate of the overdispersion parameter, c-hat, computed as the observed test statistic divided by the mean of the simulated test statistics. |
| `nsim` | 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:

| | |
|---|---|
| `chisq.table` | a list including the table of observed and expected values for each detection history and its chi-square component for each primary period (season). |

`chi.square` the chi-square test statistic, as the sum of the chi-squares across the primary periods.

`p.value` 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.

`p.global` 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.

## 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)
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 *Compute Model-averaged Parameter Estimate (Multimodel Inference)*

---

## 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 'AICsclm.clm'
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 '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 '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 '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 'AICrlm.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, ...)
```

## 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.

exclude — this argument excludes models based on the terms specified for the computation of a model-averaged estimate of parm. The exclude argument is set to NULL by default and does not exclude any models other than those without the parm. When parm 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., exclude = list("sex:mass", "mass2")). See 'Details' and 'Examples' below.

warn — logical. If TRUE, modavg performs a check and isssues a warning when the value in parm 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., I( ) for polynomial terms, : for interaction terms).

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, 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 effect size will be computed and is only relevant for models of unmarkedFitOccu, unmarkedFitColExt, unmarkedFitOccuFP, unmarkedFitOccuRN, unmarkedFitMPois, unmarkedFitPCount, unmarkedFitPCO, unmarkedFitDS, unmarkedFitGDS, unmarkedFitGMM, and unmarkedFitGPC classes. The character strings supported vary with the type of model fitted. For unmarkedFitOccu objects, either psi or detect can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For unmarkedFitColExt, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the inital year, colonization, extinction, and detectability, respectively. For unmarkedFitOccuFP objects, one can specify

psi, detect, or fp, for occupancy, detectability, and probability of assigning false-positives, 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 objects, one can enter lambda, gamma, omega, or detect, to specify parameters on abundance, recruitment, apparent survival, and detectability, respectively. For unmarkedFitDS objects, only lambda is supported for the moment. For unmarkedFitGDS, lambda and phi denote abundance and availability, respectively. For unmarkedFitGMM and unmarkedFitGPC objects, lambda, phi, and detect denote abundance, availability, and detectability, 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 terms(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, gls, hurdle, lm, lme, lmekin, maxlikeFit, mer, glmerMod, lmerMod, 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
##"sizecatab: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 "sizecatab" occurs sometimes in
##an interaction in some models
## Not run: modavg(cand.set = Cand, parm = "sizecatab",
                modnames = Modnames)
## End(Not run)

##exclude models with "sizecat:mass" interaction - results are CORRECT
modavg(cand.set = Cand, parm = "sizecatab", 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.

## Usage

```
reverse.parm(parm)
reverse.exclude(exclude)
formatCands(cand.set)
```

## 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. |

## 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.

## 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.

## Author(s)

Marc J. Mazerolle

## See Also

`aictab`, `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)
```

---

`modavgCustom` *Compute Model-averaged Parameter Estimate (Multimodel Inference) from User-supplied Input*

---

## 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)
```

## 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). |
| nobs | the sample size required to compute the AICc or QAICc. |
| 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 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 and dynamic occupancy models (MacKenzie et al. 2002, 2003), *N*-mixture models (Royle 2004, Dail and Madsen 2011), or capture-mark-recapture models (e.g., Lebreton et al. 1992). If c.hat > 1, modavg 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)). |

## 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` `modavg`, `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
modavgCustom(logL = LL, K = Ks, modnames = Modnames,
             estimate = mode.ests, se = model.se.ests, nobs = 121)



##model-averaging with shrinkage
##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
##extract
modavgCustom(logL = LLs, K = Ks, modnames = Model.names,
             nobs = nrow(min.trap), estimate = betas, se = ses)

## End(Not run)
```

---

modavgEffect *Compute Model-averaged Effect Sizes (Multimodel Inference on Group Differences)*

---

## 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 'AICrlm.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, ...)
```

## 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 explanatory 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'). |
| second.ord | logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc). |
| nobs | this argument allows the specification of 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. To obtain confidence intervals corrected for multiple comparisons between pairs of treatments, it is possible to adjust the $\alpha$ level according to various strategies such as the Bonferroni correction (Dunn 1961). |
| type | the scale of prediction requested, one of "response" or "link" (only relevant for glm, mer, and unmarkedFit classes). Note that the value "terms" is not defined for modavgEffect). |
| 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 and dynamic occupancy models (MacKenzie et al. 2002, 2003), or *N*-mixture models (Royle 2004, Dail and |

Madsen 2011). If `c.hat > 1`, `modavgEffect` 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` 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 `unmarkedFitOccu`, `unmarkedFitColExt`, `unmarkedFitOccuFP`, `unmarkedFitOccuRN`, `unmarkedFitMPois`, `unmarkedFitPCount`, `unmarkedFitPCO`, `unmarkedFitDS`, `unmarkedFitGDS`, `unmarkedFitGMM`, and `unmarkedFitGPC` classes. The character strings supported vary with the type of model fitted. For `unmarkedFitOccu` objects, either `psi` or `detect` can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For `unmarkedFitColExt`, possible values are `psi`, `gamma`, `epsilon`, and `detect`, for parameters on occupancy in the inital year, colonization, extinction, and detectability, respectively. For `unmarkedFitOccuFP` objects, one can specify `psi`, `detect`, or `fp`, for occupancy, detectability, and probability of assigning false-positives, 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` objects, one can enter `lambda`, `gamma`, `omega`, or `detect`, to specify parameters on abundance, recruitment, apparent survival, and detectability, respectively. For `unmarkedFitDS` objects, only `lambda` is supported for the moment. For `unmarkedFitGDS`, `lambda` and `phi` denote abundance and availability, respectively. For `unmarkedFitGMM` and `unmarkedFitGPC` objects, `lambda`, `phi`, and `detect` denote abundance, availability, and detectability, 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 a 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, rlm, survreg, as well as unmarkedFitOccu, unmarkedFitColExt, unmarkedFitOccuFP, unmarkedFitOccuRN, unmarkedFitPCount, unmarkedFitPCO, unmarkedFitDS, unmarkedFitGDS, unmarkedFitMPois, unmarkedFitGMM, and unmarkedFitGPC 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 |

## 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 behaviorial 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)
```

---

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`, `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 'AICrlm.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, ...)
```

## 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 the same structure as that of the original data frame for which we want to make predictions. |
| 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. |
| type | the scale of prediction requested, one of response or link. The latter is only relevant for glm, mer, and unmarkedFit classes. Note that the value terms is not defined for modavgPred. |
| 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 and dynamic occupancy models (MacKenzie et al. 2002, 2003), or *N*-mixture models (Royle 2004, Dail and Madsen 2011). If c.hat > 1, modavgPred 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 class.

gamdisp — the value of the gamma dispersion parameter.

parm.type — this argument specifies the parameter type on which the effect size will be computed and is only relevant for models of unmarkedFitOccu, unmarkedFitColExt, unmarkedFitOccuFP, unmarkedFitOccuRN, unmarkedFitMPois, unmarkedFitPCount, unmarkedFitPCO, unmarkedFitDS, unmarkedFitGDS, unmarkedFitGMM, and unmarkedFitGPC classes. The character strings supported vary with the type of model fitted. For unmarkedFitOccu objects, either psi or detect can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For unmarkedFitColExt, possible values are psi, gamma, epsilon, and detect, for parameters on occupancy in the inital year, colonization, extinction, and detectability, respectively. For unmarkedFitOccuFP objects, one can specify psi, detect, or fp, for occupancy, detectability, and probability of assigning false-positives, 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 objects, one can enter lambda, gamma, omega, or detect, to specify parameters on abundance, recruitment, apparent survival, and detectability, respectively. For unmarkedFitDS objects, only lambda is supported for the moment. For unmarkedFitGDS, lambda and phi denote abundance and availability, respectively. For unmarkedFitGMM and unmarkedFitGPC objects, lambda, phi, and detect denote abundance, availability, and detectability, 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:

| | |
|---|---|
| `type` | the scale of predicted values (response or link) for `glm`, `mer`, `merMod`, or `unmarkedFit` classes. |
| `mod.avg.pred` | the model-averaged prediction over the entire candidate model set. |
| `uncond.se` | 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 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 (Multimodel 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 'AICclmm'
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 '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 'AICmaxlikeFit.list'
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 '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, ...)
```

## 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 appropri- |

ate 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, `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 effect size will be computed and is only relevant for models of `unmarkedFitOccu`, `unmarkedFitColExt`, `unmarkedFitOccuFP`, `unmarkedFitOccuRN`, `unmarkedFitMPois`, `unmarkedFitPCount`, `unmarkedFitPCO`, `unmarkedFitDS`, `unmarkedFitGDS`, `unmarkedFitGMM`, and `unmarkedFitGPC` classes. The character strings supported vary with the type of model fitted. For `unmarkedFitOccu` objects, either `psi` or `detect` can be supplied to indicate whether the parameter is on occupancy or detectability, respectively. For `unmarkedFitColExt`, possible values are `psi`, `gamma`, `epsilon`, and `detect`, for parameters on occupancy in the inital year, colonization, extinction, and detectability, respectively. For `unmarkedFitOccuFP` objects, one can specify `psi`, `detect`, or `fp`, for occupancy, detectability, and probability of assigning false-positives, 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` objects, one can enter `lambda`, `gamma`, `omega`, or `detect`, to specify parameters on abundance, recruitment, apparent survival, and detectability, respectively. For `unmarkedFitDS` objects, only `lambda` is supported for the moment. For `unmarkedFitGDS`, `lambda` and `phi` denote abundance and availability, respectively. For `unmarkedFitGMM` and `unmarkedFitGPC` objects, `lambda`, `phi`, and `detect` denote abundance, availability, and detectability, 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`, `gls`, `hurdle`, `lm`, `lme`, `lmekin`, `maxlikeFit`, `mer`, `glmerMod`, `lmerMod`, `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)
```

---

multComp *Create Model Selection Tables based on Multiple Comparisons*

---

## 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, 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 '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", ...)
```

## Arguments

| | |
|---|---|
| mod | a model of one of the above-mentioned classes that includes at least one factor as an explanatory variable. |
| factor.id | 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. |
| letter.labels | logical. If TRUE, letters are used as labels to denote the grouping structure. If FALSE, numbers are used as group labels. |
| second.ord | logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc), otherwise returns Akaike's Information Criterion (AIC). |
| 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. |

| | |
|---|---|
| newdata | 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 factor.id variable. All levels of the factor.id variables should be included in the newdata data frame to get model-averaged predictions for each level. If NULL, model-averaged predictions are computed for each level of the factor.id variable while the values of the other explanatory variables are taken from the first row of the original data set. |
| 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 around predicted values for each level of factor.id. |
| correction | 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). |
| type | the scale of prediction requested, one of "response" or "link". The latter is only relevant for glm and mer classes. Note that the value "terms" is not defined for multComp. |
| 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, multComp will return the quasi-likelihood analogue of the information criterion requested. This option is not supported for generalized linear mixed models of the mer class. |
| gamdisp | the value of the gamma dispersion parameter in a gamma GLM. |
| ... | 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 $\alpha$ 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:

| | |
|---|---|
| factor.id | the factor for which grouping patterns are investigated. |
| models | a list with the output of each model representing a different grouping pattern for the factor of interest. |
| model.names | a vector of model names denoting the grouping pattern for each level of the factor. |
| model.table | the model selection table for the models corresponding to each grouping pattern for the factor of interest. |
| ordered.levels | 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. |
| model.avg.est | a matrix with the model-averaged prediction, unconditional standard error, and confidence intervals for each level of the factor. |
| conf.level | the confidence level used for the confidence intervals. |
| correction | 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 behaviorial 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, ...)
```

## Arguments

| | |
|---|---|
| mod | the *N*-mixture model of 'unmarkedFitPCount', 'unmarkedFitPCO', 'unmarkedFitDS', 'unmarkedFitGDS', 'unmarkedFitGMM', 'unmarkedFitGPC', or 'unmarkedFitMPois' classes for which a goodness-of-fit test is required. |
| nsim | the number of bootstrapped samples. |
| plot.hist | logical. Specifies that a histogram of the bootstrapped test statistic is to be included in the output. |
| ... | 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 (c-hat) 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 c-hat is similar to the one implemented for capture-mark-recapture models in program MARK (White and Burnham 1999).

Note that values of c-hat > 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 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 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 « 1 can also indicate lack-of-fit and that an alternative model should be investigated.

## Value

`Nmix.chisq` returns two value:

| | |
|---|---|
| `chi.square` | the Pearson chi-square statistic. |
| `model.type` | the class of the fitted model. |

`Nmix.gof.test` returns the following components:

| | |
|---|---|
| `model.type` | the class of the fitted model. |
| `chi.square` | the Pearson chi-square statistic. |
| `t.star` | the bootstrapped chi-square test statistics (i.e., obtained for each of the simulated data sets). |
| `p.value` | 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. |
| `c.hat.est` | the estimate of the overdispersion parameter, c-hat, computed as the observed test statistic divided by the mean of the simulated test statistics. |
| `nsim` | 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 *Strength of Pine Wood Based on the Density Adjusted for Resin Content*

---

## 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) ...
```

---

# predictSE *Computing Predicted Values and Standard Errors*

---

## 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 'unmarkedFitPCount'
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, unmarkedFitPCount, or unmarkedFitPCO containing the output of a model. |
| newdata | a data frame with the same structure as that of the original data frame for which we want to make predictions. |
| se.fit | logical. If TRUE, compute standard errors on predictions. |
| print.matrix | 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. |
| level | 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., level = 0). |
| type | specifies the type of prediction requested. This argument can take the value response or link, for predictions on the scale of the response variable or on the scale of the linear predictor, respectively. |
| c.hat | value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from Nmix.gof.test. If c.hat > 1, predictSE 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 c.hat (e.g., c.hat > 4) may indicate that model structure is inappropriate. |
| parm.type | the parameter for which predictions are made based on the *N*-mixture model of class unmarkedFitPCount or unmarkedFitPCO classes. |
| ... | 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 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:

| | |
|---|---|
| `fit` | the predicted values. |
| `se.fit` | the standard errors of the predicted values (if `se.fit = TRUE`). |

## 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), ]
```

`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)
```

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, nice.names = TRUE,
        include.AICc = 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, 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,
        nice.names = TRUE, table.countDist = "distance", ...)

## S3 method for class 'countHist'
xtable(x, caption = NULL, label = NULL,
        align = NULL, digits = NULL, display = NULL,
        nice.names = TRUE, table.countHist = "count", ...)

## S3 method for class 'detHist'
xtable(x, caption = NULL, label = NULL,
        align = NULL, digits = NULL, display = NULL,
        nice.names = TRUE, table.detHist = "freq", ...)

## S3 method for class 'dictab'
xtable(x, caption = NULL, label = NULL, align = NULL,
        digits = NULL, display = NULL, nice.names = TRUE,
        include.DIC = TRUE, include.Cum.Wt = FALSE, ...)

## S3 method for class 'mb.chisq'
xtable(x, caption = NULL, label = NULL, align = NULL,
        digits = NULL, display = NULL, nice.names = TRUE,
        include.detection.histories = TRUE, ...)

## S3 method for class 'modavg'
```

```
xtable(x, caption = NULL, label = NULL, align = NULL,
        digits = NULL, display = NULL, nice.names = TRUE,
        print.table = FALSE, ...)

## S3 method for class 'modavgEffect'
xtable(x, caption = NULL, label = NULL,
        align = NULL, digits = NULL, display = NULL, nice.names = TRUE,
        print.table = FALSE, ...)

## S3 method for class 'modavgPred'
xtable(x, caption = NULL, label = NULL,
        align = NULL, digits = NULL, display = NULL, nice.names = TRUE,
        ...)

## S3 method for class 'modavgShrink'
xtable(x, caption = NULL, label = NULL,
        align = NULL, digits = NULL, display = NULL, nice.names = TRUE,
        print.table = FALSE, ...)

## S3 method for class 'multComp'
xtable(x, caption = NULL, label = NULL,
        align = NULL, digits = NULL, display = NULL, nice.names = TRUE,
        print.table = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | an object of class aictab, boot.wt, countDist, countHist, detHist, dictab, mb.chisq, modavg, modavgEffect, modavgPred, modavgShrink, or multComp. |
| caption | 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 NULL and suppresses the caption. |
| label | a character vector storing the LaTeX label or HTML anchor. The default value is NULL and suppresses the label. |
| align | 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. |
| digits | 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. |
| display | a character vector of length equal to the number of columns (including the rownames) specifying the format of each column. For example, use s for strings, f for numbers in the regular format, or d for integers. See formatC for additional possible values. |
| nice.names | logical. If TRUE, column labels are modified to improve their appearance in the table. If FALSE, simpler labels are used, or the ones supplied directly by the user in the object storing the output. |

include.AICc logical. If TRUE, the column containing the information criterion of each model is printed in the table. If FALSE, the column is suppressed.

include.DIC logical. If TRUE, the column containing the deviance information criterion (DIC) of each model is printed in the table. If FALSE, the column is suppressed.

include.LL logical. If TRUE, the column containing the log-likelihood of each model is printed in the table. If FALSE, the column is suppressed.

include.Cum.Wt logical. If TRUE, the column containing the cumulative Akaike weights is printed in the table. If FALSE, the column is suppressed.

include.AICcWt logical. If TRUE, the column containing the Akaike weight of each model is printed in the table. If FALSE, the column is suppressed.

include.detection.histories
logical. If TRUE, the column containing detection histories is printed in the table. If FALSE, the column is suppressed.

print.table 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.

table.detHist character string specifying, either "freq", "prop", or "hist". If table.type = "freq", 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 table.type = "prop", 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 detection history.

table.countDist
character string specifying, either "distance", "count", "freq", or "prop". If table.type = "distance", the function returns a table of counts summarized for each distance class. If table.type = "count", the function returns the table of frequencies of counts observed across sites. If table.type = "freq", 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 table.type = "prop", 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.

table.countHist
character string specifying, either "count", "freq", "prop", or "hist". If table.type = "count", the function returns the table of frequencies of counts observed across sites. If table.type = "freq", 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 table.type = "prop", 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. http://CRAN.R-project.org/package=xtable.

## See Also

aictab, boot.wt, dictab, formatC, mb.chisq, modavg, modavgEffect, modavgPred, modavgShrink, multComp, 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
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-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")
}

## End(Not run)
```

# Index