

Package ‘AICcmodavg’

February 6, 2012

Type Package

Title Model selection and multimodel inference based on (Q)AIC(c)

Version 1.24

Date 2012-02-06

Author Marc J. Mazerolle <marc.mazerolle@uqat.ca>. Special thanks to T. Ergon for the original idea of storing candidate models in a list.

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

Depends methods, stats

Suggests lme4, MASS, Matrix, nlme, nnet, unmarked, stats4

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). Tables are printed with delta AIC and Akaike weights. 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 linear models (`‘lm’` class), generalized linear models (`‘glm’` class), linear models fit by generalized least squares (`‘gls’` class), linear mixed models (`‘lme’` class), generalized linear mixed models (`‘mer’` class), multinomial and ordinal logistic regressions (`‘multinom’` and `‘polr’` classes), and nonlinear models (`‘nls’` class). The package also supports various models incorporating detection probabilities such as single-season occupancy models (`‘unmarkedFitOccu’` class), multiple-season occupancy models (`‘unmarkedFitColExt’` class), single-season heterogeneity models (`‘unmarkedFitOccuRN’` class), single-season and multiple-season N-mixture models for repeated counts (`‘unmarkedFitPCount’` and

'unmarkedFitPCO' classes, respectively), and distance sampling models ('unmarkedFitDS' and 'unmarkedFitGDS' classes).

License GPL (>= 2)

LazyLoad yes

Repository CRAN

Date/Publication 2012-02-06 15:16:45

R topics documented:

AICcmodavg-package	2
AICc	7
aictab	10
beetle	15
cement	16
confset	17
c_hat	21
dry.frog	23
evidence	24
extract.LL.unmarked	26
extractSE.mer	28
fam.link.mer	29
importance	30
min.trap	33
modavg	34
modavg.shrink	42
modavg.utility	47
modavgpred	48
pine	55
predictSE.gls	56
predictSE.lme	57
predictSE.mer	59
Index	63

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). Tables are printed with delta AIC and Akaike weights. 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 linear models ('lm' class), generalized linear models ('glm' class), linear models fit by generalized least squares ('gls' class), linear mixed models ('lme' class), generalized linear mixed models ('mer' class), multinomial and ordinal logistic regressions ('multinom' and 'polr' classes), and nonlinear models ('nls' class). The package also supports various models incorporating detection probabilities such as single-season occupancy models ('unmarkedFitOccu' class), multiple-season occupancy models ('unmarkedFitColExt' class), single-season heterogeneity models ('unmarkedFitOccuRN' class), single-season and multiple-season N-mixture models for repeated counts ('unmarkedFitPCount' and 'unmarkedFitPCO' classes, respectively), and distance sampling models ('unmarkedFitDS' and 'unmarkedFitGDS' classes).

Details

```
Package:    AICcmodavg
Type:       Package
Version:    1.24
Date:       2012-02-06
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.
- [confset](#) Determines the confidence set for the best model based on one of three criteria.
- [evidence](#) Computes the evidence ratio between the highest-ranked model based on the information criteria selected and a lower-ranked model.
- [extract.LL.unmarked](#) Extracts log-likelihood from models of class 'unmarkedFit'.
- [extractSE.mer](#) Extracts standard errors of the fixed effects of a generalized linear mixed model of class 'mer' and adds the labels.
- [fam.link.mer](#) Extracts the distribution family and link function from a generalized linear mixed model of class 'mer'.
- [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.

- `modavg.shrink` Computes shrinkage version of model-averaged estimate, unconditional standard error, and unconditional confidence interval of a parameter of interest among a set of candidate models.
- `modavgpred` Computes model-average predictions and unconditional SE's among entire set of candidate models.
- `c_hat` Computes an estimate of variance inflation factor for binomial or Poisson GLM's based on Pearson's chi-square.
- `predictSE.gls` Computes predictions and associated standard errors for 'glms' object based on linear predictor.
- `predictSE.lme` Computes predictions and associated standard errors for 'lme' object based on fixed effects (i.e., population predictions).
- `predictSE.mer` Computes predictions and associated standard errors for 'mer' object based on fixed effects (i.e., population predictions).

Author(s)

Marc J. Mazerolle <marc.mazerolle@uqat.ca>. Special thanks to T. Ergon for the original idea of storing candidate models in a list.

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]]) #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
modavg.shrink(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)

##single-season occupancy model example modified from ?occu
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.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
modavg.shrink(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)
```

AICc

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

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

Usage

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

```
AICc.glm(mod, return.K = FALSE, c.hat = 1, second.ord = TRUE,
         nobs = NULL)
```

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

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

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

```
AICc.mult(mod, return.K = FALSE, c.hat = 1, second.ord = TRUE,
          nobs = NULL)
```

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

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

```
AICc.unmarked(mod, return.K = FALSE, c.hat = 1, second.ord = TRUE,
              nobs = NULL)
```

Arguments

`mod` an object of class 'lm', 'glm', 'glsl', 'lme', 'mer', 'multinom', 'nls', 'polr', and various 'unmarkedFit' classes containing the output of a model.

return.K	logical. If FALSE, the function returns the information criteria specified. If TRUE, the function returns K (number of estimated parameters) for a given model. Using this argument to facilitate computation of tables was an original idea from T. Ergon.
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, or single-season occupancy models (MacKenzie et al. 2002). If c.hat > 1, 'AICc' 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.
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'.

Details

'AICc' is a function that calls either 'AICc.glm', 'AICc.gls', 'AICc.lme', 'AICc.mer', 'AICc.mult', 'AICc.nls', 'AICc.polr', or 'AICc.unmarked' depending on the class of the object. The current function is implemented for 'lm', 'glm', 'gls', 'lme', 'mer', 'multinom', 'nls', and 'polr' classes as well as 'unmarkedFitOccu', 'unmarkedFitColExt', 'unmarkedFitOccuRN', 'unmarkedFitPCount', 'unmarkedFitPCO' classes. The function 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' selects one of the functions below based on the class of the object:

- 'AICc.glm' returns the AIC, AICc, QAIC, or QAICc depending on the values of the arguments.
- 'AICc.gls' returns the AIC or AICc depending on the values of the arguments.
- 'AICc.lme' returns the AIC or AICc depending on the values of the arguments.
- 'AICc.mer' returns the AIC or AICc depending on the values of the arguments.
- 'AICc.mult' returns the AIC, AICc, QAIC, or QAICc depending on the values of the arguments.
- 'AICc.nls' returns the AIC or AICc depending on the values of the arguments.
- 'AICc.polr' returns the AIC or AICc depending on the values of the arguments.
- 'AICc.unmarked' returns the AIC, AICc, QAIC, or QAICc 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.
- 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.
- Pinheiro, J. C., Bates, D. M. (2000) *Mixed-effect models in S and S-PLUS*. Springer Verlag: New York.
- 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

[aictab](#), [confset](#), [importance](#), [evidence](#), [c_hat](#), [modavg](#), [modavg.shrink](#), [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)
require(nlme)
m1 <- lme(distance ~ age, random = ~1 | Subject, data = Orthodont,
          method= "ML")
AICc(m1, return.K = FALSE)
```

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 objects of 'lm', 'glm', 'gls', 'lme', 'mer', 'multinom', 'nls', 'polr' classes as well as various models of 'unmarkedFit' classes but does not yet allow mixing of different classes.

Usage

```
aictab(cand.set, modnames, sort = TRUE, c.hat = 1, second.ord = TRUE,
       nobs = NULL)

aictab.glm(cand.set, modnames, sort = TRUE, c.hat = 1,
          second.ord = TRUE, nobs = NULL)

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

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

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

aictab.mult(cand.set, modnames, sort = TRUE, c.hat = 1,
           second.ord = TRUE, nobs = NULL)

aictab.nls(cand.set, modnames, sort = TRUE, second.ord = TRUE,
```

```

nobs = NULL)

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

aictab.unmarked(cand.set, modnames, sort = TRUE, c.hat = 1,
second.ord = TRUE, nobs = NULL)

```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table.
<code>sort</code>	logical. If true, the model selection table is ranked according to the (Q)AIC(c) values.
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from <code>'c_hat'</code> . Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, or single-season occupancy models (MacKenzie et al. 2002). If <code>c.hat > 1</code> , <code>'AICc'</code> will return the quasi-likelihood analogue of the information criterion requested. This option is not supported for generalized linear mixed models of the <code>'mer'</code> class.
<code>second.ord</code>	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., <code>'nobs'</code> defaults to total number of observations). This is relevant only for mixed models or various models of <code>'unmarkedFit'</code> classes where sample size is not straightforward. In such cases, one might use total number of observations or number of independent clusters (e.g., sites) as the value of <code>'nobs'</code> .

Details

`'aictab'` is a function that calls either `'aictab.glm'`, `'aictab.gls'`, `'aictab.lme'`, `'aictab.mer'`, `'aictab.mult'`, `'aictab.nls'`, `'aictab.polr'`, or `'aictab.unmarked'` depending on the class of the object. The current function is implemented for `'lm'`, `'glm'`, `'gls'`, `'lme'`, `'mer'`, `'multinom'`, and `'polr'` classes as well as `'unmarkedFitOccu'`, `'unmarkedFitColExt'`, `'unmarkedFitOccuRN'`, `'unmarkedFitPCCount'`, `'unmarkedFitPCO'` 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 (i.e., identity vs log link).
- 8) When dealing with models with overdispersion, use the same value of \hat{c} 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 \hat{c} from the most complex model (global model). If $\hat{c} > 1$, 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 the parameters of interest appearing in the top 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', 'aictab.glm', 'aictab.lme', 'aictab.mer', 'aictab.mult', 'aictab.nls', 'aictab.polr', and 'aictab.unmarked' create 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 ($\exp(-0.5 \cdot \text{delta}[i])$). This is not to be confused with the likelihood of the parameters given the data. The relative likelihood can then be normalized across all models to get the model probabilities.
(Q)AIC(c)wt	the Akaike weights, also termed "model probabilities" sensu Burnham and Anderson (2002) and Anderson (2008). These measures indicate the level of support (i.e., weight of evidence) in favor of any given model being the most parsimonious among the candidate model set.
Cum.Wt	the cumulative Akaike weights. These are only meaningful if results in table are sorted in decreasing order of Akaike weights (i.e., <code>sort = TRUE</code>).
c.hat	if c.hat was specified as an argument, it is included in the table.
LL	if c.hat = 1 and parameters estimated by maximum likelihood, the log-likelihood of each model.
Quasi.LL	if c.hat > 1, the quasi log-likelihood of each model.
Res.LL	if parameters are estimated by restricted maximum-likelihood (REML), the restricted log-likelihood of each model.

Author(s)

Marc J. Mazerolle

References

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

##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

Modnames <- paste("Mod", 1:length(Cand.set), sep = " ")
res.table <- aictab(cand.set = Cand.set, modnames = Modnames,
        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

```

```

##mer class example modified from ?glmer
require(lme4)
##create proportion of incidence
cbpp$prop <- cbpp$incidence/cbpp$size
gm1 <- glmer(prop ~ period + (1 | herd), family = binomial, weights =
             size, data = cbpp)
gm2 <- glmer(prop ~ 1 + (1 | herd), family = binomial, weights =
             size, data = cbpp)

Cands <- list(gm1, gm2)
Model.names <- c("effect of period", "no effect of period")
aictab(cand.set = Cands, modnames = Model.names, sort = TRUE)
detach(package:lme4)

##single-season occupancy model example modified from ?occu
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 table
aictab(cand.set = Cand.mods, modnames = Modnames,
       second.ord = TRUE)

detach(package:unmarked)

```

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) ...
```

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

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.
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, or single-season occupancy models (MacKenzie et al. 2002). If c.hat > 1, 'AICc' 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.
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'.

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 ($\Delta(Q)AIC(c) \leq 2$), some support ($2 < \Delta(Q)AIC(c) \leq 7$), little support ($7 < \Delta(Q)AIC(c) \leq 10$), 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: $\text{cutoff} = \exp(-1 * \Delta(Q)AIC(c)/2)$.

Value

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

when method = 'raw':

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

when method = 'ordinal'

method	identifies the method of determining the confidence set on the best model.
substantial	a reduced table with the models included in the confidence set for which $\Delta(Q)AIC(c) \leq 2$.

some	a reduced table with the models included in the confidence set for which $2 < \text{delta (Q)AIC(c)} \leq 7$.
little	a reduced table with the models included in the confidence set for which $7 < \text{delta (Q)AIC(c)} \leq 10$.
none	a reduced table with the models included in the confidence set for which $\text{delta (Q)AIC(c)} > 10$.
when method = 'ordinal'	
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.
- 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.

See Also

[AICc](#), [aictab](#), [c_hat](#), [evidence](#), [importance](#), [modavg](#), [modavg.shrink](#), [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
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)

```

c_hat

Compute Estimate of Dispersion for Poisson and Binomial GLM's

Description

This function computes an estimate of c-hat for binomial or Poisson GLM's based on Pearson's chi-square divided by the residual degrees of freedom.

Usage

```
c_hat(mod)
```

Arguments

mod the model for which a c-hat estimate is required.

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, McCullagh and Nelder (1989) recommend using Pearson's chi-square divided by the residual degrees of freedom, which performs better. The latter is the method implemented by 'c_hat'.

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 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 create 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 $\ll 1$ can also indicate lack-of-fit and that an alternative model (and distribution) should be investigated.

Note that it is only possible to estimate c -hat for binomial models with trials > 1 (i.e., success/trial or cbind(success, failure) syntax) or with Poisson GLM's.

Value

'c_hat' returns the estimated c -hat value

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

See Also

[AICc](#), [confset](#), [evidence](#), [modavg](#), [importance](#), [modavgpred](#)

Examples

```
#binomial glm example
set.seed(seed = 10)
resp <- rbinom(n = 60, size = 1, prob = 0.5)
set.seed(seed = 10)
```

```

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

## End(Not run)

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

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

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

```

dry.frog

Frog dehydration experiment on three different 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 vector

Species a factor with levels Racla

Shade a numeric vector
 SVL a numeric vector
 Substrate a factor with levels PEAT, SOIL, and SPHAGNUM
 Initial_mass a numeric vector
 Mass_lost a numeric vector
 Airtemp a numeric vector
 Wind_cat a numeric vector
 Cloud a numeric vector
 cent_Initial_mass a numeric vector
 Initial_mass2 a numeric vector
 cent_Air a numeric vector
 Perc.cloud a numeric vector
 Wind a numeric vector
 log_Mass_lost a numeric vector

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' as the first argument.

Usage

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

Arguments

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

Details

The default compares the 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:

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

Author(s)

Marc J. Mazerolle

References

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

See Also

[AICc](#), [aictab](#), [c_hat](#), [confset](#), [importance](#), [modavg](#), [modavg.shrink](#), [modavgpred](#)

Examples

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

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

```

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

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

##run models for the Orthodont data set in nlme
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 <- aicctab(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 = aicctab(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")

```

extract.LL.unmarked *Extract Log-Likelihood of Model*

Description

This function extracts the log-likelihood from an object of various 'unmarkedFit' classes.

Usage

```
extract.LL.unmarked(mod)
```

Arguments

mod an object of 'unmarkedFit' class resulting from the fit of 'occu', 'occuRN', 'colect', 'pcount', or 'pcountOpen'.

Details

This utility function extracts the information from an 'unmarkedFit' object resulting from 'occu', 'occuRN', 'colect', 'pcount', or 'pcountOpen'. The function is called by 'AICc.unmarked' and 'aictab.unmarked'.

Value

'extract.LL.unmarked' returns the value of the log-likelihood of the model.

Author(s)

Marc J. Mazerolle

See Also

[AICc](#), [aictab](#), [occu](#), [occuRN](#), [colect](#), [pcount](#), [pcountOpen](#)

Examples

```
##modified example of single-season model with heterogeneity from ?occuRN
require(unmarked)
data(birds)
woodthrushUMF <- unmarkedFrameOccu(woodthrush.bin)

##survey occasion-specific detection probabilities
fm.wood.rn <- occuRN(~ obsNum ~ 1, woodthrushUMF)

##extract log-likelihood
extract.LL.unmarked(fm.wood.rn)

detach(package:unmarked)
```

 extractSE.mer

Extract SE of Fixed Effects of 'glmer' Fit

Description

This function extracts the standard errors (SE) of the fixed effects of a generalized linear mixed model fit with 'glmer' and adds the appropriate labels.

Usage

```
extractSE.mer(mod)
```

Arguments

mod an object of 'mer' class resulting from the fit of 'glmer'.

Details

This is an extractor function that uses 'vcov.mer' from the 'lme4' package and it is called by 'modavg.mer'.

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](#)

Examples

```
##modified example from ?glmer
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.mer(gm1) #with labels
detach(package:lme4)
```

`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' class resulting from the fit of 'glmer' or 'lmer'.

Details

This utility function extracts the information from an 'mer' object resulting from 'glmer' or 'lmer'. The function is called by 'modavg.mer', 'modavgpred.mer', and 'predictSE.mer'.

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.mer' and 'modavgpred.mer'.

Author(s)

Marc J. Mazerolle

See Also

[modavg](#), [modavgpred](#), [predictSE.mer](#), [glmer](#), [lmer](#)

Examples

```
##modified example from ?glmer
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)

##example with linear mixed model with Orthodont data from
##Pinheiro and Bates (2000)
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)

```

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, c.hat = 1, second.ord = TRUE,
          nobs = NULL, parm.type = 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.
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, or single-season occupancy models (MacKenzie et al. 2002). If c.hat > 1, 'AICc' 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.
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'.
parm.type	this argument specifies the parameter type of the estimate specified in 'parm' and is only relevant for models of 'unmarkedFitOccu', 'unmarkedFitColExt', 'unmarkedFitOccuRN', 'unmarkedFitPCount', 'unmarkedFitPCO' 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 initial year, colonization, extinction, and detectability, respectively. For 'unmarkedFitOccuRN' objects, either 'lambda' or 'detect' can be entered for abundance and detectability parameters, respectively. For 'unmarkedFitPCount' 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.

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.
- 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.

See Also

[AICc](#), [aictab](#), [c_hat](#), [confset](#), [evidence](#), [modavg](#), [modavg.shrink](#), [modavgpred](#)

Examples

```

##example on Orthodont data set in nlme
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)

##single-season occupancy model example modified from ?occu
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)
```

min.trap	<i>Anuran larvae counts in minnow traps across pond type.</i>
----------	---

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).

Usage

```
modavg(cand.set, parm, modnames, c.hat = 1, gamdisp = NULL,  
       conf.level = 0.95, second.ord = TRUE, nobs = NULL,  
       exclude = NULL, warn = TRUE, uncond.se = "revised",  
       parm.type = NULL)
```

```
modavg.glm(cand.set, parm, modnames, c.hat = 1, gamdisp = NULL,  
           conf.level = 0.95, second.ord = TRUE, nobs = NULL,  
           exclude = NULL, warn = TRUE, uncond.se = "revised")
```

```
modavg.gls(cand.set, parm, modnames, conf.level = 0.95,  
           second.ord = TRUE, nobs = NULL, exclude = NULL, warn = TRUE,  
           uncond.se = "revised")
```

```
modavg.lme(cand.set, parm, modnames, conf.level = 0.95,  
           second.ord = TRUE, nobs = NULL, exclude = NULL, warn = TRUE,  
           uncond.se = "revised")
```

```
modavg.mer(cand.set, parm, modnames, conf.level = 0.95,  
           second.ord = TRUE, nobs = NULL, exclude = NULL, warn = TRUE,  
           uncond.se = "revised")
```

```
modavg.mult(cand.set, parm, modnames, c.hat = 1, conf.level = 0.95,  
            second.ord = TRUE, nobs = NULL, exclude = NULL, warn = TRUE,  
            uncond.se = "revised")
```

```
modavg.polr(cand.set, parm, modnames, conf.level = 0.95,  
            second.ord = TRUE, nobs = NULL, exclude = NULL, warn = TRUE,  
            uncond.se = "revised")
```

```
modavg.unmarked(cand.set, parm, modnames, c.hat = 1,  
                conf.level = 0.95, second.ord = TRUE, nobs = NULL,  
                exclude = NULL, warn = TRUE, uncond.se = "revised",  
                parm.type = NULL)
```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>parm</code>	the parameter of interest, enclosed between quotes, for which a model-averaged estimate is required. For a categorical variable, the label of the estimate must be included as it appears in the output (see 'Details' below).
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table.
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from 'c_hat'. Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, or single-season occupancy models (MacKenzie et al. 2002). If <code>c.hat</code> > 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 $\sqrt{c.hat}$). This option is not supported for generalized linear mixed models of the 'mer' class.
<code>gamdisp</code>	if gamma GLM is used, the dispersion parameter should be specified here to apply the same value to each model.
<code>conf.level</code>	the confidence level requested for the computation of unconditional confidence intervals.
<code>second.ord</code>	logical. If TRUE, the function returns the second-order Akaike information criterion (i.e., AICc).
<code>nobs</code>	this argument allows to specify a numeric value other than total sample size to compute the AICc (i.e., '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'.
<code>exclude</code>	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., <code>exclude = list("sex:mass", "mass2")</code>). See 'Details' and 'Examples' below.
<code>warn</code>	logical. If TRUE, <code>modavg()</code> performs a check and issues 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., <code>I()</code> for polynomial terms, <code>'</code> ' for interaction terms).
<code>uncond.se</code>	either, "old", or "revised", specifying the equation used to compute the unconditional standard error of a model-averaged estimate. With <code>uncond.se = "old"</code> , computations are based on equation 4.9 of Burnham and Anderson (2002), which was the former way to compute unconditional standard errors. With <code>uncond.se</code>

= "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.

parm.type this argument specifies the parameter type of the estimate specified in 'parm' and is only relevant for models of 'unmarkedFitOccu', 'unmarkedFitColExt', 'unmarkedFitOccuRN', 'unmarkedFitPCount', 'unmarkedFitPCO', 'unmarkedFitDS' and 'unmarkedFitGDS' 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 initial year, colonization, extinction, and detectability, respectively. For 'unmarkedFitOccuRN' objects, either 'lambda' or 'detect' can be entered for abundance and detectability parameters, respectively. For 'unmarkedFitPCount' 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' and 'unmarkedFitGDS' objects, only 'lambda' is supported for the moment.

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.

Care must be taken 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 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'.

'modavg' is a function that calls 'modavg.glm', 'modavg.gls', 'modavg.lme', 'modavg.mer', 'modavg.mult', 'modavg.polr', or 'modavg.unmarked' depending on the class of the object. The current function is implemented for a list containing objects of 'lm', 'glm', 'gl', 'lme', 'mer', 'multinom',

and 'polr' classes as well as 'unmarkedFitOccu', 'unmarkedFitColExt', 'unmarkedFitOccuRN', 'unmarkedFitPCCount', 'unmarkedFitPCO', 'unmarkedFitDS', and 'unmarkedFitGDS' classes.

Value

'modavg', 'modavg.glm', 'modavg.gls', 'modavg.lme', 'modavg.mer', 'modavg.mult', 'modavg.polr', and 'modavg.unmarked' create 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.
- 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. (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](#), [aictab](#), [c_hat](#), [confset](#), [evidence](#), [importance](#), [modavg.shrink](#), [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 model-averaged estimate of TypeBOG
modavg(parm = "TypeBOG", cand.set = Cand.mod, modnames = Modnames)
##round to 4 digits after decimal point
print(modavg(parm = "TypeBOG", cand.set = Cand.mod,
            modnames = Modnames), digits = 4)

##compute c-hat estimate based on residual deviance as in Mazerolle
##(2006)
Cand.mod[[1]]$deviance/Cand.mod[[1]]$df.residual

##compute model-averaged estimate of TypeBOG as in Table 4 of
##Mazerolle (2006)
modavg(parm = "TypeBOG", cand.set = Cand.mod, modnames = Modnames,

```

```

c.hat = 1.11)

##example with similarly-named variables and interaction terms
set.seed(seed = 4)
resp <- rnorm(n = 40, mean = 3, sd = 1)
size <- rep(c("small", "medsmall", "high", "medhigh"), times = 10)
set.seed(seed = 4)
mass <- rnorm(n = 40, mean = 2, sd = 0.1)
mass2 <- mass^2
age <- rpois(n = 40, lambda = 3.2)
agecorr <- rpois(n = 40, lambda = 2)
sizecat <- rep(c("a", "ab"), times = 20)
data1 <- data.frame(resp = resp, size = size, sizecat = sizecat,
                    mass = mass, mass2 = mass2, age = age,
                    agecorr = agecorr)

##set up models in list
Cand <- list( )
Cand[[1]] <- lm(resp ~ size + agecorr, data = data1)
Cand[[2]] <- lm(resp ~ size + mass + agecorr, data = data1)
Cand[[3]] <- lm(resp ~ age + mass, data = data1)
Cand[[4]] <- lm(resp ~ age + mass + mass2, data = data1)
Cand[[5]] <- lm(resp ~ mass + mass2 + size, data = data1)
Cand[[6]] <- lm(resp ~ mass + mass2 + sizecat, data = data1)
Cand[[7]] <- lm(resp ~ sizecat, data = data1)
Cand[[8]] <- lm(resp ~ sizecat + mass + sizecat:mass, data = data1)
Cand[[9]] <- lm(resp ~ agecorr + sizecat + mass + sizecat:mass,
                data = data1)

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

aictab(cand.set = Cand, modnames = Modnames, sort = TRUE) #correct

##as expected, issues warning as mass occurs sometimes with "mass2" or
##"sizecat:mass" in some of the models
## Not run: modavg(cand.set = Cand, parm = "mass", modnames = Modnames)

##no warning issued, because "age" and "agecorr" never appear in same model
modavg(cand.set = Cand, parm = "age", modnames = Modnames)

##as expected, issues warning because warn=FALSE, but it is a very bad
##idea in this example since "mass" occurs with "mass2" and "sizecat:mass"
##in some of the models - results are INCORRECT
## Not run: modavg(cand.set = Cand, parm = "mass", modnames = Modnames,
                  warn = FALSE)
## End(Not run)

##correctly excludes models with quadratic term and interaction term
##results are CORRECT
modavg(cand.set = Cand, parm = "mass", modnames = Modnames,

```



```

##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")

##example of model-averaged estimate of area from distance model
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)

```

modavg.shrink	<i>Compute Model-averaged Parameter Estimate with Shrinkage (Multi-model Inference)</i>
---------------	---

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) and Anderson (2008, pp. 130-132). Specifically, models without the parameter of interest have an estimate and variance of 0. 'modavg.shrink' also returns unconditional standard errors and unconditional confidence intervals as described in Buckland et al. (1997) and Burnham and Anderson (2002).

Usage

```
modavg.shrink(cand.set, parm, modnames, c.hat = 1, gamdisp = NULL,
              conf.level = 0.95, second.ord = TRUE, nobs = NULL,
              uncond.se = "revised", parm.type = NULL)

modavg.shrink.glm(cand.set, parm, modnames, c.hat = 1,
                  gamdisp = NULL, conf.level = 0.95, second.ord = TRUE,
                  nobs = NULL, uncond.se = "revised")

modavg.shrink.gls(cand.set, parm, modnames, conf.level = 0.95,
                  second.ord = TRUE, nobs = NULL, uncond.se = "revised")

modavg.shrink.lme(cand.set, parm, modnames, conf.level = 0.95,
                  second.ord = TRUE, nobs = NULL, uncond.se = "revised")

modavg.shrink.mer(cand.set, parm, modnames, conf.level = 0.95,
                  second.ord = TRUE, nobs = NULL, uncond.se = "revised")

modavg.shrink.mult(cand.set, parm, modnames, c.hat = 1,
                   conf.level = 0.95, second.ord = TRUE, nobs = NULL,
                   uncond.se = "revised")

modavg.shrink.polr(cand.set, parm, modnames, conf.level = 0.95,
                   second.ord = TRUE, nobs = NULL,
                   uncond.se = "revised")

modavg.shrink.unmarked(cand.set, parm, modnames, c.hat = 1,
                       conf.level = 0.95, second.ord = TRUE,
                       nobs = NULL, uncond.se = "revised",
                       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.
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, or single-season occupancy models (MacKenzie et al. 2002). If c.hat > 1, 'modavg.shrink' 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.
conf.level	the confidence level requested for the computation of unconditional confidence intervals.
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.
parm.type	this argument specifies the parameter type of the estimate specified in 'parm' and is only relevant for models of 'unmarkedFitOccu', 'unmarkedFitColExt', 'unmarkedFitOccuRN', 'unmarkedFitPCount', 'unmarkedFitPCO', 'unmarkedFitDS' and 'unmarkedFitGDS' 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 initial year,

colonization, extinction, and detectability, respectively. For 'unmarkedFitOccuRN' objects, either 'lambda' or 'detect' can be entered for abundance and detectability parameters, respectively. For 'unmarkedFitPCount' 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' and 'unmarkedFitGDS' objects, only 'lambda' is supported for the moment.

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 'modavg.shrink'.

'modavg.shrink' is a function that calls 'modavg.shrink.glm', 'modavg.shrink.gls', 'modavg.shrink.lme', 'modavg.shrink.mer', 'modavg.shrink.mult', 'modavg.shrink.polr', or 'modavg.shrink.unmarked' depending on the class of the object. The current function is implemented for a list containing objects of 'lm', 'glm', 'gls', 'lme', 'mer', 'multinom', and 'polr' classes as well as 'unmarkedFitOccu', 'unmarkedFitColExt', 'unmarkedFitOccuRN', 'unmarkedFitPCount', 'unmarkedFitPCO', 'unmarkedFitDS', and 'unmarkedFitGDS' classes.

Value

'modavg.shrink', 'modavg.shrink.glm', 'modavg.shrink.gls', 'modavg.shrink.lme', 'modavg.shrink.mer', 'modavg.shrink.mult', 'modavg.shrink.polr', and 'modavg.shrink.unmarked' create an object of class 'modavg' 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.
- 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. (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](#), [aictab](#), [c_hat](#), [importance](#), [confset](#), [evidence](#), [modavg](#), [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
```

```

modavg.shrink(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: modavg.shrink(cand.set = Cand.unbalanced,
                        modnames = Modnames, parm = "x1")
## End(Not run)

##example on Orthodont data set in nlme
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 <- modavg.shrink(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)

##example of N-mixture model modified from ?pcount
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
modavg.shrink(cand.set = Cands, modnames = Modnames, parm = "elev",
             parm.type = "lambda")
detach(package:unmarked)

```

modavg.utility

Accomodate Different Specifications of Interaction Terms

Description

These utility functions enable the user to specify differently interaction terms (e.g., A:B, B:A) across models for model averaging. Both functions are called internally by 'modavg'.

Usage

```

reverse.parm(parm)
reverse.exclude(exclude)

```

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.

Details

Both functions have been added to avoid problems when users are not consistent in the specification of interaction terms across models.

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.

Author(s)

Marc J. Mazerolle

See Also

[modavg](#), [modavg.shrink](#), [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
```

modavgpred

Compute Model-averaged Predictions

Description

This function computes the model-averaged predictions and unconditional standard errors based on the entire candidate model set. The function is currently implemented for `'lm'`, `'glm'`, `'gls'`, `'lme'`, and `'mer'` object classes that are stored in a list as well as various models of `'unmarkedFit'` classes.

Usage

```

modavgpred(cand.set, modnames, newdata, type = "response", c.hat = 1,
           gamdisp = NULL, second.ord = TRUE, nobs = NULL,
           uncond.se = "revised", parm.type = NULL)

modavgpred.glm(cand.set, modnames, newdata, type = "response",
              c.hat = 1, gamdisp = NULL, second.ord = TRUE,
              nobs = NULL, uncond.se = "revised")

modavgpred.gls(cand.set, modnames, newdata, second.ord = TRUE,
              nobs = NULL, uncond.se = "revised")

modavgpred.lme(cand.set, modnames, newdata, second.ord = TRUE,
              nobs = NULL, uncond.se = "revised")

modavgpred.mer(cand.set, modnames, newdata, type = "response",
              c.hat = 1, second.ord = TRUE, nobs = NULL,
              uncond.se = "revised")

modavgpred.unmarked(cand.set, modnames, newdata, second.ord = TRUE,
                   type = "response", c.hat = 1, nobs = NULL,
                   uncond.se = "revised", parm.type = NULL)

```

Arguments

<code>cand.set</code>	a list storing each of the models in the candidate model set.
<code>modnames</code>	a character vector of model names to facilitate the identification of each model in the model selection table.
<code>newdata</code>	a data frame with the same structure as that of the original data frame for which we want to make predictions.
<code>type</code>	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 'modavgpred').
<code>c.hat</code>	value of overdispersion parameter (i.e., variance inflation factor) such as that obtained from 'c_hat'. Note that values of <code>c.hat</code> different from 1 are only appropriate for binomial GLM's with trials > 1 (i.e., success/trial or <code>cbind(success, failure)</code> syntax), with Poisson GLM's, or single-season occupancy models (MacKenzie et al. 2002). If <code>c.hat > 1</code> , 'aictab' will return the quasi-likelihood analogue of the information criteria requested and multiply the variance-covariance matrix of the estimates by this value (i.e., SE's are multiplied by <code>sqrt(c.hat)</code>). This option is not supported for generalized linear mixed models of the 'mer' class.
<code>gamdisp</code>	the value of the gamma dispersion parameter.
<code>second.ord</code>	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.
parm.type	this argument specifies the parameter type of the estimate specified in 'parm' and is only relevant for models of 'unmarkedFitOccu', 'unmarkedFitColExt', 'unmarkedFitOccuRN', 'unmarkedFitPCount', 'unmarkedFitPCO', 'unmarkedFitDS' and 'unmarkedFitGDS' 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 initial year, colonization, extinction, and detectability, respectively. For 'unmarkedFitOccuRN' objects, either 'lambda' or 'detect' can be entered for abundance and detectability parameters, respectively. For 'unmarkedFitPCount' 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' and 'unmarkedFitGDS' objects, only 'lambda' is supported for the moment.

Details

'modavgpred' is a function that calls 'modavgpred.gls', 'modavgpred.glm', 'modavgpred.lme', 'modavgpred.mer', or 'modavgpred.unmarked' depending on the class of the object. 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.

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 of beta estimates (i.e., estimates on the linear predictor scale). For predictions of some types of response variables (e.g., discrete values 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. Burnham and Anderson (2002, p. 164) suggest alternative methods of computing unconditional confidence intervals for small degrees of freedom with profile likelihood intervals or bootstrapping.

Value

'modavgpred' returns an object of class 'modavgpred' with the following components:

type	the scale of predicted values (response or link) for 'glm', 'mer', or 'unmarked-Fit' 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.

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.
- 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.

See Also

[AICc](#), [aictab](#), [importance](#), [c_hat](#), [confset](#), [evidence](#), [modavg](#), [modavg.shrink](#), [predict](#), [predict.glm](#), [predictSE.gls](#), [predictSE.lme](#), [predictSE.mer](#)

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
##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)

library(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
## Not run:
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'

## End(Not run)
## Not run:
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
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")

##example of model-averaged abundance from distance model
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)

##example using Orthodont data set from Pinheiro and Bates (2000)
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)

```

pine	<i>Strength of pine wood based on the density adjusted for resin content.</i>
------	---

Description

This data set consists of the strength of pine wood as a function of density or density adjusted for resin content.

Usage

```
data(pine)
```

Format

A data frame with 42 observations on the following 3 variables.

y pine wood strength

x pine wood density

z pine wood density adjusted for resin content

Details

Burnham and Anderson (2002, p. 183) use this data set originally from Carlin and Chib (1995) to illustrate model selection for two competing and non-nested models.

Source

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

Carlin, B. P., Chib, S. (1995) Bayesian model choice via Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society, Series B* **57**, 473–484.

Examples

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

Description

Function to compute predicted values based on linear predictor and associated standard errors from a generalized least squares model.

Usage

```
predictSE.gls(mod, newdata, se.fit = TRUE, print.matrix = FALSE)
```

Arguments

<code>mod</code>	an object of class 'gls' containing the output of a model.
<code>newdata</code>	a data frame with the same structure as that of the original data frame for which we want to make predictions.
<code>se.fit</code>	logical. If TRUE, compute standard errors on predictions.
<code>print.matrix</code>	logical. If TRUE, the output is returned as a matrix, with predicted values and standard errors in columns. If FALSE, the output is returned as a list.

Details

'predictSE.gls' computes predicted values based on the linear predictor and associated standard errors, excluding the correlation or variance structure of the model. Standard errors are approximated using the delta method (Oehlert 1992).

Value

'predictSE.gls' returns requested values either as a matrix ('print.matrix = TRUE') or list ('print.matrix = FALSE') with components:

<code>fit</code>	the predicted values.
<code>se.fit</code>	the standard errors of the predicted values (if 'se.fit = TRUE').

Note

For standard errors with better properties, especially for small samples, one can opt for simulations (see Gelman and Hill 2007).

Author(s)

Marc J. Mazerolle

References

- 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](#), [predict.gls](#)

Examples

```
##Orthodont data from Pinheiro and Bates (2000) revisited
require(nlme)
m1 <- gls(distance ~ age, correlation = corCompSymm(value = 0.5, form = ~ 1 | Subject), data = Orthodont,
          method= "ML")
## Not run:
##compare against lme fit
identical(logLik(m1),
          logLik(lme(distance ~ age, random = ~1 | Subject, data = Orthodont,
                    method= "ML")))
##both are identical

## End(Not run)

##compute predictions and SE's for different ages
predictSE.gls(m1, newdata = data.frame(age = c(8, 10, 12, 14)))
```

predictSE.lme

Computing Predicted Values and Standard Errors

Description

Function to compute predicted values based on fixed effects (population predictions) and associated standard errors from a linear mixed effect model.

Usage

```
predictSE.lme(mod, newdata, se.fit = TRUE, level = 0,
              print.matrix = FALSE)
```

Arguments

mod	an object of class 'lme' 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.

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).
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.

Details

'predictSE.lme' computes predicted values based on fixed effects and associated standard errors. Standard errors are approximated using the delta method (Oehlert 1992).

Value

'predictSE.lme' 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

To get the standard errors on predictions that include random effects, one can opt for simulations (see Gelman and Hill 2007).

Author(s)

Marc J. Mazerolle

References

- 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

[lme](#), [predict.lme](#)

Examples

```
##Orthodont data set in Pinheiro and Bates (2000)
require(nlme)
m1 <- lme(distance ~ age, random = ~1 | Subject, data = Orthodont,
          method= "ML")
##create a new data frame to make predictions
newOrth <- data.frame(Sex = c("Male", "Male", "Female", "Female"),
                     age = c(15, 20, 15, 20),
                     Subject = c("M01", "M01", "F30", "F30"))
predictSE.lme(m1, newdata = newOrth, level = 0)
predict(m1, newdata = newOrth, level = 0)
```

```
##compare against predict.lme( )

##because only 'level = 0' is supported, the grouping levels are not
##necessary for prediction
newd <- data.frame(Sex = c("Male", "Male", "Female", "Female"),
                  age = c(15, 20, 15, 20))
predictSE.lme(m1, newdata = newd, level = 0)
predictSE.lme(m1, newdata = newd, level = 0, print.matrix = TRUE)

## Not run:
  predictSE.lme(m1, newdata = newd, level = 1, print.matrix = TRUE)
##generates an error

## End(Not run)
```

 predictSE.mer

Computing Predicted Values and Standard Errors

Description

Function to compute predicted values based on fixed effects (population predictions) and associated standard errors from a generalized linear mixed effect model.

Usage

```
predictSE.mer(mod, newdata, se.fit = TRUE, type = "response",
              level = 0, print.matrix = FALSE)
```

Arguments

mod	an object of class 'mer' 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.
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.
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).
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.

Details

'predictSE.mer' computes predicted values based on fixed effects and associated standard errors. Standard errors are approximated using the delta method (Oehlert 1992). The current version supports the use of offsets only for the Poisson distribution.

Value

'predictSE.mer' 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

To get the standard errors on predictions that include random effects, one can opt for simulations (see Gelman and Hill 2007).

Author(s)

Marc J. Mazerolle

References

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

[predictSE.lme](#), [mer-class](#), [lmer](#), [glmer](#)

Examples

```
##contagious bovine pleuropneumonia example modified from lme4
require(lme4)
data(cbpp)
##create proportion of incidence
cbpp$prop <- cbpp$incidence/cbpp$size
gm1 <- glmer(prop ~ period + (1 | herd), family = binomial,
             weights = size, data = cbpp)

##create a data set to make predictions
newherd<- data.frame(period = as.factor(c("1", "2", "3", "4")))

##predictions on logit link scale
predictSE.mer(mod = gm1, newdata = newherd, se.fit = TRUE,
             type = "link", level = 0, print.matrix = FALSE)

##predictions on scale of original response variable
predictSE.mer(mod = gm1, newdata = newherd, se.fit = TRUE,
             type = "response", level = 0, print.matrix = TRUE)

##example with linear mixed model with Orthodont data from
##Pinheiro and Bates (2000)
```

```

data(Orthodont, package = "nlme")
m2 <- lmer(distance ~ Sex + (1 | Subject), data = Orthodont,
           REML = FALSE)

##create data set for predictions for all combinations of Sex and 2 ages
neworth <- expand.grid(Sex = c("Male", "Female"), age = c(8, 10))

##compute predicted values
predictSE.mer(m2, newdata = neworth)
##the following yields the same answer because the model
##uses an identity link
predictSE.mer(m2, newdata = neworth, type = "link")

## Not run:
predictSE.mer(m2, newdata = neworth, level = 1, print.matrix = TRUE)
##generates an error

## End(Not run)

##compare with the following:
m3 <- glmer(distance ~ Sex + (1 | Subject),
            family = gaussian(link = log), data = Orthodont, nAGQ = 1)

##predictions on original scale of response variable
predictSE.mer(m3, newdata = neworth, type = "response")

##predictions on log scale
predictSE.mer(m3, newdata = neworth, type = "link")

##example of Poisson mixed model with offset term
##assign values
set.seed(seed = 222)
beta0 <- -3.45
beta.CWD <- 0.01
beta.basal.area <- 0.1
block.var <- 0.28
n.blocks <- 10
rep.per.block <- 9
n.obs <- n.blocks * rep.per.block
CWD <- rnorm(n = n.obs, mean = 190, sd = 50)
effort <- sample(10:20, size = n.obs, replace = TRUE)
basal.area <- rnorm(n = n.obs, mean = 60, sd = 20)
block.id <- sort(rep(1:n.blocks, rep.per.block))

##generate data
##random intercept (block)
block.int <- rnorm(n = 10, mean = 0, sd = sqrt(block.var))
lin.pred <- beta0 + beta.CWD * CWD + beta.basal.area * basal.area + log(effort)

y.val <- rep(NA, n.obs)
for (i in 1:n.obs) {

```

```
y.val[i] <- rpois(n = 1, lambda = exp(lin.pred[i] + block.int[block.id[i]]))
}
sim.data <- data.frame(Y.val = y.val, CWD = CWD, Basal.area = basal.area, Block = as.factor(block.id), Effort = effort)
sim.data$log.Effort <- log(sim.data$Effort)

##run model with log transformation of offset variable within call
m1 <- glmer(Y.val ~ CWD + Basal.area + (1 | Block) + offset(log(Effort)), data = sim.data, family = poisson)
##predictions
pred.data <- expand.grid(CWD = mean(sim.data$CWD), Basal.area = seq(from = 20, to = 50, by = 10), Effort = 20)
predictSE.mer(m1, newdata = pred.data, type = "response")

##run model with offset already on log scale
m1 <- glmer(Y.val ~ CWD + Basal.area + (1 | Block) + offset(log.Effort), data = sim.data, family = poisson)
##predictions
pred.data <- expand.grid(CWD = mean(sim.data$CWD), Basal.area = seq(from = 20, to = 50, by = 10), log.Effort = log(20))
predictSE.mer(m1, newdata = pred.data, type = "response")

##both are identical

detach(package:lme4)
```

Index

*Topic **datasets**

beetle, [15](#)
cement, [16](#)
dry.frog, [23](#)
min.trap, [33](#)
pine, [55](#)

*Topic **models**

AICc, [7](#)
AICcmodavg-package, [2](#)
aictab, [10](#)
c_hat, [21](#)
confset, [17](#)
evidence, [24](#)
extract.LL.unmarked, [26](#)
extractSE.mer, [28](#)
fam.link.mer, [29](#)
importance, [30](#)
modavg, [34](#)
modavg.shrink, [42](#)
modavg.utility, [47](#)
modavgpred, [48](#)
predictSE.gls, [56](#)
predictSE.lme, [57](#)
predictSE.mer, [59](#)

[AICc](#), [3](#), [7](#), [13](#), [19](#), [22](#), [25](#), [27](#), [31](#), [37](#), [45](#), [51](#)

[AICcmodavg](#) ([AICcmodavg-package](#)), [2](#)

[AICcmodavg-package](#), [2](#)

[aictab](#), [3](#), [9](#), [10](#), [19](#), [25](#), [27](#), [31](#), [37](#), [45](#), [51](#)

[beetle](#), [15](#)

[c_hat](#), [4](#), [9](#), [13](#), [19](#), [21](#), [25](#), [31](#), [37](#), [45](#), [51](#)

[cement](#), [16](#)

[colext](#), [27](#)

[confset](#), [3](#), [9](#), [13](#), [17](#), [22](#), [25](#), [31](#), [37](#), [45](#), [51](#)

[dry.frog](#), [23](#)

[evidence](#), [3](#), [9](#), [13](#), [19](#), [22](#), [24](#), [31](#), [37](#), [45](#), [51](#)

[extract.LL.unmarked](#), [3](#), [26](#)

[extractSE.mer](#), [3](#), [28](#)

[fam.link.mer](#), [3](#), [29](#)

[glmer](#), [28](#), [29](#), [60](#)

[gls](#), [57](#)

[importance](#), [3](#), [9](#), [13](#), [19](#), [22](#), [25](#), [30](#), [37](#), [45](#), [51](#)

[lme](#), [58](#)

[lmer](#), [28](#), [29](#), [60](#)

[mer-class](#), [60](#)

[min.trap](#), [33](#)

[modavg](#), [3](#), [9](#), [13](#), [19](#), [22](#), [25](#), [28](#), [29](#), [31](#), [34](#), [45](#), [48](#), [51](#)

[modavg.shrink](#), [4](#), [9](#), [13](#), [19](#), [25](#), [31](#), [37](#), [42](#), [48](#), [51](#)

[modavg.utility](#), [47](#)

[modavgpred](#), [4](#), [9](#), [13](#), [19](#), [22](#), [25](#), [29](#), [31](#), [37](#), [45](#), [48](#), [48](#)

[occu](#), [27](#)

[occuRN](#), [27](#)

[pcount](#), [27](#)

[pcountOpen](#), [27](#)

[pine](#), [55](#)

[predict](#), [51](#)

[predict.glm](#), [51](#)

[predict.gls](#), [57](#)

[predict.lme](#), [58](#)

[predictSE.gls](#), [4](#), [51](#), [56](#)

[predictSE.lme](#), [4](#), [51](#), [57](#), [60](#)

[predictSE.mer](#), [4](#), [29](#), [51](#), [59](#)

[print.aictab](#) ([aictab](#)), [10](#)

[print.confset](#) ([confset](#)), [17](#)

[print.evidence](#) ([evidence](#)), [24](#)

[print.importance](#) ([importance](#)), [30](#)

[print.modavg](#) ([modavg](#)), [34](#)

[print.modavg.shrink](#) ([modavg.shrink](#)), [42](#)

`print.modavgpred (modavgpred)`, [48](#)

`reverse.exclude (modavg.utility)`, [47](#)

`reverse.parm (modavg.utility)`, [47](#)