Package: mra (via r-universe)

October 14, 2024

Type Package

Title Mark-Recapture Analysis

Version 2.16.11

Date 2018-01-04

Author Trent McDonald [cre, aut], Eric Regehr [ctb], Bryan Manly [ctb], Jeff Bromaghin [ctb]

Maintainer Trent McDonald <tmcdonald@west-inc.com>

Description Accomplishes mark-recapture analysis with covariates. Models available include the Cormack-Jolly-Seber open population (Cormack (1972) <doi:10.2307/2556151>; Jolly (1965) <doi:10.2307/2333826>; Seber (1965) <doi:10.2307/2333827>) and Huggin's (1989) <doi:10.2307/2336377> closed population. Link functions include logit, sine, and hazard. Model selection, model averaging, plot, and simulation routines included. Open population size by the Horvitz-Thompson (1959) <doi:10.2307/2280784> estimator.

License GNU General Public License

BugReports https://github.com/tmcd82070/MRA/issues

Suggests knitr

VignetteBuilder knitr

Imports stats, graphics

RoxygenNote 6.0.1

NeedsCompilation yes

Date/Publication 2018-01-05 13:48:51 UTC

Additional_repositories https://cranhaven.r-universe.dev

Repository https://cranhaven.r-universe.dev

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/mra

RemoteSha feba4dee690f6222d9bcd676e1124be1d8df15e5

Contents

mra-package . . . 2
dipper.data . . . 4
F.3d.model.matrix . . . 5
F.cjs.covars . . . 6
F.cjs.estim . . . 7
F.cjs.gof . . . 15
F.cjs.simulate . . . 18
F.cr.model.avg . . . 22
F.cr.model.matrix . . . 25
F.fit.table . . . 27
F.huggins.estim . . . 29
F.sat.lik . . . 35
F.step.cjs . . . 36
F.update.df . . . 39
ivar . . . 40
lines.cjs . . . 42
mra.control . . . 43
plot.cjs . . . 45
plot.nhat . . . 46
predict.cjs . . . 47
print.cjs . . . 48
print.hug . . . 49
print.nhat . . . 50
residuals.cjs . . . 52
tvar . . . 53

Index **55**

`mra-package` *MRA - Mark Recapture Analysis*

Description

Description - This package contains analysis functions, and associated routines, to conduct analyses of mark-recapture (capture-recapture) data using individual, time, and individual-time varying covariates. In general, these routines relate vectors of capture histories to vectors of covariates using a regression approach (Amstrup et al. 2005, Ch 9). All capture, survival, transition, etc. parameters are functions of individual and time specific covariates, and the estimated parameters are coefficients in logistic-linear equations.

Relationship to MARK - For the most part, these routines perform a subset of the analyses available in program MARK or via the MARK front-end package, RMark. The most significant difference between this package and MARK is parameterization. The parameterization used here does not utilize triangular "parameter information matrices" (PIMs) as MARK (and RMark) does. Because of this, the "design" matrix utilized by this package is not parallel to the "design" matrix of program MARK. For those new to mark-recapture analysis, this parameterization difference will

be inconsequential. The approach taken here provides equivalent modeling flexibility, yet is easier to grasp and visualize, in our opinion. For those already familiar with the PIMs used by program MARK, it is helpful to view the "PIMs" of this package as rectangular matrices of the real parameters. I.e., the "PIMs" of this package are rectangular matrices where cell (i,j) contains the real parameter (capture or survival) for individual i at capture occasion j.

Analyses available here that are *not* included in program MARK include:

- Estimation of population size from open population CJS models via the Horvitz-Thompson estimator.
- Residuals, goodness of fit tests, and associated plots for assessing model fit in open CJS models.

Future Research - The author of MRA welcome interest in and routines that perform the following analyzes:

- Continuous time models. Especially those that allow inclusion of covariates.
- Band recovery models.
- Baysian models.
- Joint live-dead recovery models.
- MCMC methods or routines that can be applied to exiting models.
- Plotting methods for exiting models.
- Model selection methods for existing models.
- Simulation methods and routines.

Details

Package:	mra
Type:	Package
License:	GNU General Public License

Author(s)

Trent McDonald

Maintainer: Trent McDonald <tmcdonald@west-inc.com>

References

Amstrup, S.C., T.L. McDonald, and B.F.J. Manly. 2005. *Handbook of Capture-Recapture Analysis*, Princeton: Princeton University Press.

`dipper.data` *European Dipper data*

Description

Example capture-recapture data from a study of European dippers.

Usage

```
data(dipper.data)
```

Format

A data frame containing 294 capture histories and the sex designation of birds captured. Capture indicators are either 0 = not captured, 1 = captured, or 2 = captured but died and not released back into the population. Columns in the data frame are:

`h1` a numeric vector indicating capture at occasion 1

`h2` a numeric vector indicating capture at occasion 2

`h3` a numeric vector indicating capture at occasion 3

`h4` a numeric vector indicating capture at occasion 4

`h5` a numeric vector indicating capture at occasion 5

`h6` a numeric vector indicating capture at occasion 6

`h7` a numeric vector indicating capture at occasion 7

`males` a numeric vector indicating males. 1 = males, 0 = females

`females` a numeric vector indicating females. 0 = males, 1 = females

Details

This is a popular capture-recapture example data set. It has been analyzed by Lebreton et al. (1992) Amstrup et al. (2005) and others.

`dipper.males` is a vector indicating male birds. I.e., `dipper.males <- dipper.data\$males`

`dipper.histories` is a matrix of just the capture history columns `h1 - h7`, extracted from `dipper.data` and made into a matrix. This matrix can be fed directly into one of the estimation routines, such as `F.cjs.estim`.

To access: After loading the MRA library (with `library(mra)`) you must execute `data(dipper.data)`, `data(dipper.data)`, or `data(dipper.males)` to get access to these data frames. They are not attached when the library is loaded.

Source

Amstrup, S. C., McDonald, T. L., and Manly, B. F. J. 2005. Handbook of Capture-Recapture Analysis. Princeton University Press. [Chapter 9 has several examples that use this data.]

Examples

```
data(dipper.data)
```

`F.3d.model.matrix` *3-Dimensional capture-recapture model matrices*

Description

Returns a 3D model matrix for capture-recapture modeling in the form of a (giant) 2D matrix.

Usage

```
F.3d.model.matrix(formula, d1, d2)
```

Arguments

`formula`	A formula object specifying covariates in a capture-recapture model. Must not have a response, i.e., ~, followed by the names of 2-D arrays or 1-D vectors contained inside calls to `ivar` or `tvar`. See `help(F.cjs.estim)` for examples.
`d1`	Magnitude of dimension 1 of the returned matrix. This is always number of rows in the returned matrix. Usually, `d1` = number of individuals = number of rows in capture history matrix.
`d2`	Magnitude of dimension 2 of the returned matrix. This is number of columns in the capture history matrix.

Details

This routine is intended to be called internally by the routines of MRA. General users should never have to call this routine.

This routine uses a call to `eval` with a model frame, and calls the R internal `model.matrix` to resolve the matrices in the formula. All matrices specified in the models should be in the current scope and accessible to both `eval` and `model.matrix`.

See `help(F.cjs.estim)` for examples of ways to specify models.

Value

A (giant) 2-d matrix containing covariate values suitable for passing to the Fortran code that does the estimation for MRA. This matrix has all the 2-d matrices of the model `cbind`-ed together. It's dimension is NAN x NS*(number of coefficients). A convenient way to view the matrix is to assign a 3-d dimension. I.e., if `x` is the result of a call to this function and there are NX coefficients in the model, then `dim(x) <- c(NAN,NS,NX)` makes a 3-d matrix with NAN rows, NS columns, and NX pages. View the covariates for a single animal with `x[3,,]` or similar statement.

Names of variables in the model are returned as attribute "variables". Whether the model has an intercept is returned as attribute "intercept".

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cr.model.matrix, tvar, ivar, model.matrix, eval

Examples

```
# Synthetic example with 10 animals and 5 occasions
nan <- 10
ns <- 5
sex <- as.factor(as.numeric(runif( nan ) > 0.5))
attr(sex,"ns") <- ns
x <- matrix( runif( nan*ns ) , nrow=nan, ncol=ns )
F.3d.model.matrix( ~ ivar(sex) + x, nan, ns )
```

F.cjs.covars *F.cjs.covars*

Description

Return an x and y 3-D array for estimation of a traditional time-variant Cormack-Jolly-Seber capture-recapture model.

Usage

```
F.cjs.covars(nan, ns)
```

Arguments

nan	Number of individuals/animals.
ns	Number of trap/mark occasions.

Details

Pages from \$x are designed to useful for fitting classical CJS models with time-variant, but individual-invariant effects. To fit a CJS model using this function, the commands would be something like:

```
tmp<-F.cjs.covars(nan,ns);F.cjs.estim(capture=~tmp\$x[,,2]+tmp\$x[,,3]+ ..., survival=
~tmp\$x[,,1]+tmp\$x[,,2]+ ..., histories=my.histories)
```

Value

A list containing a single component, \$x, that can be used to estimate a classical CJS model when included in a subsequent call to F.cjs.estim. The returned component, \$x, is a 3-D array containing 0's everywhere, except for 1's in certain columns. \$x has dimension nan X ns X ns. Element [i,j,k] of \$x is 1 if j == k, and 0 otherwise. I.e., the k-th "page" of the 3-D array has 1's in the k-th column, 0's elsewhere.

Author(s)

Trent McDonald, WEST Inc., tmcdonald@west-inc.com

See Also

F.cjs.estim

Examples

```
##  Fit CJS model to dipper data, time-varying capture and survivals.
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
dipper.cjs <- F.cjs.estim( capture=~xy$x[,,2]+xy$x[,,3]+xy$x[,,4]+xy$x[,,5]+xy$x[,,6],
 survival=~xy$x[,,1]+xy$x[,,2]+xy$x[,,3]+xy$x[,,4]+xy$x[,,5], dipper.histories )
print(dipper.cjs)
```

F.cjs.estim *F.cjs.estim - Cormack-Jolly-Seber estimation*

Description

Estimates Cormack-Jolly-Seber (CJS) capture-recapture models with individual, time, and individual-time varying covariates using the "regression" parametrization of Amstrup et al (2005, Ch 9). For live recaptures only. Losses on capture allowed. Uses a logistic link function to relate probability of capture and survival to external covariates.

Usage

```
F.cjs.estim(capture, survival, histories, cap.init, sur.init, group,  nhat.v.meth = 1,
c.hat = -1, df = NA, intervals=rep(1,ncol(histories)-1), conf=0.95,
link="logit", control=mra.control())
```

Arguments

capture	Formula specifying the capture probability model. Must be a formula object with no response. I.e., "~" followed by the names of 2-D arrays of covariates to fit in the capture model. For example: 'capture = ~ age + sex', where age and sex are matrices of size NAN X NS containing the age and sex covariate values. NAN = number of animals = number of rows in histories matrix (see below).

NS = number of samples = number of columns in histories matrix (see below). Number of matrices specified in the capture model is assumed to be NX. Time varying and individual varying vectors are fitted using `ivar()` and `tvar()` (see Details). Factors are allowed within `ivar()` and `tvar()`.

`survival` Formula specifying the survival probability model. Must be a formula object with no response. I.e., "~" followed by the names of 2-D arrays of covariates to fit in the survival model. For example: 'survival = ~ year + ageclass' where year and ageclass are matrices of size NAN X NS containing year and ageclass covariate values. Number of matrices specified in the survival model is assumed to be NY. Time varying and individual varying vectors are fitted using `ivar()` and `tvar()` (see Details). Factors are allowed within `ivar()` and `tvar()`.

`histories` A NAN X NS = (number of animals) X (number of capture occasions) matrix containing capture histories. Capture histories are comprised of 0's, 1',s and 2's. 0 in cell (i,j) means animal i was not captured on occasion j, 1 in cell (i,j) means animal i was captured on occasion j and released live back into the population, 2 in cell (i,j) means animal i was captured on occasion j and was not released back into the population (e.g., it died). Animals with '2' as the last non-zero entry of their history are considered 'censored'. Their lack of capture information is removed from the likelihood after the occasion with the 2. Rows of all zeros (i.e., no captures) are allowed in the history matrix, but do not affect coefficient or population size estimates. A warning is thrown if rows of all zeros exist. Capture and survival probabilities are computed for animals with all zero histories. In this way, it is possible to have the routine compute capture or survival estimates for combinations of covariates that do not exist in the data by associating the covariate combinations with histories that have all zero entries.

`cap.init` (optional) Vector of initial values for coefficients in the capture model. One element per covariate in `capture`. The default value usually works.

`sur.init` (optional) Vector or initial values for coefficients in the survival model. One element per covariate in `survival`. The default value usually works.

`group` (optional) A vector of length NAN giving the (non-changing) group membership of every captured animal (e.g., sex). Group is used only for computing TEST 2 and TEST 3. TEST 2 and TEST 3 are computed separately for each group. E.g., if group=sex, TEST 2 and TEST 3 are computed for each sex. TEST 2 and TEST3 are used only to estimate C-hat. See `c.hat` for pooling rules for these test components to estimate C-hat.

`nhat.v.meth` Integer specifying method for computing variance estimates of population size estimates. `nhat.v.meth` = 1 uses the variance estimator of Taylor et al. 2002, Ursus, p. 188 which is the so-called Huggins variance estimator, and incorporates covariances. `nhat.v.meth` = 2 uses the variance estimator of Amstrup et al. 2005 (p. 244, Eqn. 9.10), which is the same variance estimator as `nhat.v.meth` = 1 with more 2nd order approximation terms included. Method 2 should provide better variances than method 1, especially if the coefficient of variation of capture probabilities are >1.0, but method 2 has not been studied as much as method 1. `nhat.v.meth` = 3 uses the variance estimator of McDonald and Amstrup, 1999, JABES, which is a 1st order approximation that does not incorporate covariances. Method 3 is much faster than methods 1 and 2 and

could be easily calculated by hand, but should only be used when there is little capture heterogeneity.

`c.hat` External (override) estimate of variance inflation factor (`c.hat`) to use during estimation. If input value of `c.hat` is <= 0, MRA computes an estimate of variance inflation based on TEST 2 and TEST 3 applied to groups (if called for, see `group` above) using Manly, McDonald, and McDonald, 1993, rules for pooling. I.e., all cells in each TEST 2 or TEST 3 Chi-square component table must be >= 5 before that component contributes to the estimate of C-hat. This rules is slightly different than program MARK's pooling rules, so MRA's and MARK's estimates of `c.hat` will generally be different. If the input `c.hat` > 0, MRA does not estimate C.hat, and uses the supplied value.

`df` External (override) model degrees of freedom to use during estimation. If `df` == NA, the number of parameters is estimated from the rank of the matrix of 2nd derivatives or Hessian, depending on `cov.meth` parameter. If `df` <= 0, the number of parameters will be set to NX+NY = the number of estimated coefficients. Otherwise, if `df` > 0, the supplied value is used. Only AIC, QAIC, AICc, and QAICc are dependent on this value (in their penalty terms).

`intervals` Time intervals. This is a vector of length `ncol(histories)-1` (i.e., number of capture occasions minus 1) specifying relative time intervals between occasions. For example, if capture occasions occurred in 1999, 2000, 2005, and 2007 `intervals` would be set to `c(1,5,2)`. Estimates of survival are adjusted for time intervals between occasions assuming an exponential lifetime model, i.e., probability of surviving from occasion `j` to occasion `j+1` is `Phi(j)^(jth interval length)`, and it is the `Phi(j)`'s that are related to covariates through the survival model. In other words, all survival estimates are for an interval of length 1. If an interval of 1 is one year, then all survival estimates will be annual survival, with probability of surviving 2 years equal to annual survival squared, probability of surviving 3 years equal to annual survival cubed, etc.

`conf` Confidence level for the confidence intervals placed around estimates of population size. Default 95% confidence.

`link` The link function to be used. The link function converts linear predictors in the range (-infinity, infinity) to probabilities in the range (0,1). Valid values for the link function are "logit" (default), "sine", and "hazard". (see Examples for a plot of the link functions)

- The "logit" link is $\eta = log(\frac{p}{1-p})$ with inverse $p = \frac{1}{1+exp(-\eta)}$.
- The "sine" link is $\eta = \frac{8asin(2p-1)}{\pi}$, which ranges from -4 to 4. The inverse "sine" link is $p = \frac{1+sin(\eta\pi/8)}{2}$ for values of η between -4 and 4. For values of η < -4, p = 0. For values of η > 4, p = 1. Scaling of the sine link was chosen to yield coefficients roughly the same magnitude as the logit link.
- The "hazard" link is $\eta = log(-log(1-p))$, with inverse $1-exp(-exp(\eta))$. The value of p from the inverse hazard link approaches 0 as η decreases. For values of η > 3, p = 1 for all intents and purposes.

`control` A list containing named control parameters for the minimization and estimation process. Control parameters include number of iterations, covariance estimation method, etc. Although the default values work in the vast majority

of cases, changes to these variables can effect speed and performance for ill-behaved models. See `mra.control()` for a description of the individual control parameters.

Details

This is the work-horse routine for estimating CJS models. It compiles all the covariate matrices, then calls a Fortran routine to maximize the CJS likelihood and perform goodness-of-fit tests. Horvitz-Thompson-type population size estimates are also computed by default.

If `control=mra.control(trace=1)`, a log file, named `mra.log`, is written to the current directory. This file contains additional details, such as individual Test 2 and Test 3 components, in a semi-friendly format. This file is overwritten each run. See `help(mra.control)` for more details.

Model Specification: Both the `capture` and `survival` model can be specified as any combination of 2-d matrices (time and individual varying covariates), 1-d time varying vectors, 1-d individual varying vectors, 1-d time varying factors, and 1-d individual varying factors.

Specification of time or individual varying effects uses the `tvar` (for 'time varying') and `ivar` (for 'individual varying') functions. These functions expand covariate vectors along the appropriate dimension to be 2-d matrices suitable for fitting in the model. `ivar` expands an individual varying vector to all occasions. `tvar` expands a time varying covariate to all individuals. To do the expansion, both `tvar` and `ivar` need to know the size of the 'other' dimension. Thus, `tvar(x,100)` specifies a 2-d matrix with size `100` by `length(x)`. `ivar(x,100)` specifies a 2-d matrix with size `length(x)` by `100`.

For convenience, the 'other' dimension of time or individual varying covariates can be specified as an attribute of the vector. Assuming `x` is a NS vector and the 'nan' attribute of `x` has been set as `attr(x,"nan") <- NAN`, `tvar(x,NAN)` and `tvar(x)` are equivalent. Same, but vise-versa, for individual varying covariates (i.e., assign the number of occasions using `attr(x,"ns")<-NS`). This saves some typing in model specification.

Factors are allowed in `ivar` and `tvar`. When a factor is specified, the `contr.treatment` coding is used. By default, an intercept is assumed and the first level of all factors are dropped from the model (i.e., first levels are the reference levels, the default R action). However, there are applications where more than one level will need to be dropped, and the user has control over this via the `drop.levels` argument to `ivar` and `tvar`. For example, `tvar(x,drop.levels=c(1,2))` drops the first 2 levels of factor `x`. `tvar(x,drop.levels=length(levels(x)))` does the SAS thing and drops the last level of factor `x`. If `drop.levels` is outside the range `[1,length(levels(x))]` (e.g., negative or 0), no levels of the factor are dropped. If no intercept is fitted in the model, this results in the so-called cell means coding for factors.

Example model specifications: Assume 'age' is a NAN x NS 2-d matrix of ages, 'effort' is a size NS 1-d vector of efforts, and 'sex' is a size NAN 1-d factor of sex designations ('M' and 'F').

1. capture= ~ 1 : constant effect over all individuals and time (intercept only model)
2. capture= ~ age : Intercept plus age
3. capture= ~ age + tvar(effort,NAN) : Intercept plus age plus effort
4. capture= ~ age + tvar(effort,NAN) + ivar(sex,NS) : Intercept plus age plus effort plus sex. Females (1st level) are the reference.
5. capture= ~ -1 + ivar(sex,NS,0) : sex as a factor, cell means coding
6. capture= ~ tvar(as.factor(1:ncol(histories)),nrow(histories),c(1,2)) : time varying effects

Values in 2-d Matrix Covariates: Even though covariate matrices are required to be NAN x NS (same size as capture histories), there are not that many parameters. The first capture probability cannot be estimated in CJS models, and the NS-th survival parameter does not exist. When a covariate matrix appears in the *capture* model, only values in columns 2:ncol(histories) are used. When a covariate matrix appears in the *survival* model, only values in columns 1:(ncol(histories)-1) are used. See examples for demonstration.

Value

An object (list) of class c("cjs","cr") with many components. Use `print.cr` to print it nicely. Use `names(fit)`, where the call was `fit <- F.cr.estim(...)`, to see names of all returned components. To see values of individual components, issue commands like fit$s.hat, fit$se.s.hat, fit$n.hat, etc.

Components of the returned object are as follows:

`histories` The input capture history matrix.

`aux` Auxiliary information about the fit, mostly stored input values. This is a list containing the following components:

- call = original call
- nan = number of animals
- ns = number of samples = number of capture occasions
- nx = number of coefficients in capture model
- ny = number of coefficients in survival model
- cov.name = names of all coefficients
- ic.name = name of capture history matrix.
- mra.version = version number of MRA package used to estimate the model
- R.version = R version used for during estimation
- run.date = date the model was estimated.

`loglik` Maximized CJS likelihood value for the model

`deviance` Model deviance = -2*`loglik`. This is relative deviance, see help for `F.sat.lik`.

`aic` AIC for the model = `deviance` + 2*(df). df is either the estimated number of independent parameters (by default), or NX+NY, or a specified value, depending on the input value of DF parameter.

`qaic` QAIC (quasi-AIC) = (`deviance` / `vif`) + 2(df)

`aicc` AIC with small sample correction = AIC + (2*`df`*(`df`+1)) / (`nan` - `df` - 1)

`qaicc` QAIC with small sample correction = QAIC + (2*`df`*(`df`+1))/(`nan` - `df` - 1)

`vif` Variance inflation factor used = estimate of c.hat = `chisq.vif` / `chisq.df`

`chisq.vif` Composite Chi-square statistic from Test 2 and Test 3 used to compute `vif`, based on pooling rules.

`vif.df` Degrees of freedom for composite chi-square statistic from Test 2 and Test 3, based on pooling rules.

`parameters` Vector of all coefficient estimates, NX capture probability coefficients first, then NY survival coefficients. This vector is length NX+NY regardless of estimated DF.

`se.param` Standard error estimates for all coefficients. Length NX+NY.

`capcoef` Vector of coefficients in the capture model. Length NX.

`se.capcoef` Vector of standard errors for coefficients in capture model. Length NX.

`surcoef` Vector of coefficients in the survival model. Length NY.

`se.surcoef` Vector of standard errors for coefficients in survival model. Length NY.

`covariance` Variance-covariance matrix for the estimated model coefficients. Size (NX+NY) X (NX+NY).

`p.hat` Matrix of estimated capture probabilities computed from the model. One for each animal each occasion. Cell (i,j) is estimated capture probability for animal i during capture occasion j. Size NAN X NS. First column corresponding to first capture probability is NA because cannot estimate P1 in a CJS model.

`se.p.hat` Matrix of standard errors for estimated capture probabilities. One for each animal each occasion. Size NAN X NS. First column is NA.

`s.hat` Matrix of estimated survival probabilities computed from the model. One for each animal each occasion. Size NAN X NS. Cell (i,j) is estimated probability animal i survives from occasion j to j+1. There are only NS-1 intervals between occasions. Last column corresponding to survival between occasion NS and NS+1 is NA.

`se.s.hat` Matrix of standard errors for estimated survival probabilities. Size NAN X NS. Last column is NA.

`df` The number of parameters assumed in the model. This value was used in the penalty term of AIC, AICc, QAIC, and QAICc. This value is either the number of independent parameters estimated from the rank of the variance-covariance matrix (by default), or NX+NY, or a specified value, depending on the input value of DF parameter. See `F.update.df` to update this value after the model is fitted.

`df.estimated` The number of parameters estimated from the rank of the variance-covariance matrix. This is stored so that `df` can be updated using `F.update.df`.

`control` A list containing the input maximization and estimation control parameters.

`message` A vector of strings interpreting various codes about the estimation. The messages interpret, in this order, the codes for (1) maximization algorithm used, (2) exit code from the maximization algorithm (interprets `exit.code`), and (3) covariance matrix code (interprets `cov.code`).

`exit.code` Exit code from the maximization routine. Interpretation for `exit.code` is in `message`. Exit codes are as follows:

- exit.code = 0: FAILURE: Initial Hessian not positive definite.
- exit.code = 1: SUCCESS: Convergence criterion met.
- exit.code = 2: FAILURE: G'dX > 0, rounding error.
- exit.code = 3: FAILURE: Likelihood evaluated too many times.
- exit.code =-1: FAILURE: Unknown optimization algorithm."

`cov.code` A code indicating the method used to compute the covariance matrix.

`fn.evals` The number of times the likelihood was evaluated prior to exit from the minimization routine. If `exit.code = 3`, `fn.evals` equals the maximum set in `mra.control`. This, in combination with the exit codes and execution time, can help detect non-convergence or bad behavior.

`ex.time` Execution time for the maximization routine, in *minutes*. This is returned for 2 reasons. First, this is useful for benchmarking. Second, in conjunction with `exit.code`, `cov.code`, and `fn.evals`, this could be used to detect ill-behaved or marginally unstable problems, if you know what you are doing. Assuming `maxfn` is set high in `mra.control()` (e.g., 1000), if `exit.code = 1` but the model takes a long time to execute relative to similarly sized problems, it could indicate unstable or marginally ill-behaved models.

`n.hat` Vector of Horvitz-Thompson estimates of population size. The Horvitz-Thompson estimator of size is,

$$\hat{N}_{ij} = \sum_{i=1}^{NAN} \frac{h_{ij}}{\hat{p}_{ij}}$$

Length of `n.hat` = NS. No estimate for first occasion.

`se.n.hat` Estimated standard errors for `n.hat` estimates. Computed using method specified in `nhat.v.meth`.

`n.hat.lower` Lower limit of `n.hat.conf` percent on `n.hat`. Length NS.

`n.hat.upper` Upper limit of `n.hat.conf` percent on `n.hat`. Length NS.

`n.hat.conf` Confidence level of intervals on `n.hat`

`nhat.v.meth` Code for method used to compute variance of `n.hat`

`num.caught` Vector of observed number of animals captured each occasion. Length NS.

`fitted` Matrix of fitted values for the capture histories. Size NAN X NS. Cell (i,j) is expected value of capture indicator in cell (i,j) of `histories` matrix.

`residuals` Matrix of Pearson residuals defined as,

$$r_{ij} = \frac{(h_{ij} - \Psi_{ij})^2}{\Psi_{ij}}$$

, where Ψ_{ij} is the expected (or fitted) value for cell (i,j) and h_{ij} is the capture indicator for animal i at occasion j. This matrix has size NAN X NS. See parts pertaining to the "overall test" in documentation for `F.cjs.gof` for a description of Ψ_{ij}.

`resid.type` String describing the type of residuals computed. Currently, only Pearson residuals are returned.

Note

MARK Users: Due to differences in the way MRA and MARK parameterize the sine link, *coefficient* estimates will differ between the two packages when this link is used to fit the same model in both packages. The fit (measured by deviance, AIC, etc.) will agree between the two packages. Capture and survival probability estimates will also agree between the two packages.

MARK does not contain a hazard rate link function.

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

References

Taylor, M. K., J. Laake, H. D. Cluff, M. Ramsay, and F. Messier. 2002. Managing the risk from hunting for the Viscount Melville Sound polar bear population. Ursus 13:185-202.

Manly, B. F. J., L. L. McDonald, and T. L. McDonald. 1999. The robustness of mark-recapture methods: a case study for the northern spotted owl. Journal of Agricultural, Biological, and Environmental Statistics 4:78-101.

Huggins, R. M. 1989. On the statistical analysis of capture experiments. Biometrika 76:133-140.

Amstrup, S. C., T. L. McDonald, and B. F. J. Manly (editors). 2005. Handbook of Capture-Recapture Analysis. Princeton University Press.

Peterson. 1986. Statistics and Probability Letters. p.227.

McDonald, T. L., and S. C. Amstrup. 2001. Estimation of population size using open capture-recapture models. Journal of Agricultural, Biological, and Environmental Statistics 6:206-220.

See Also

tvar, ivar, print.cjs, residuals.cjs, plot.cjs, F.cjs.covars, F.cjs.gof, mra.control, F.update.df

Examples

```
## Fit CJS model to dipper data, time-varying capture and survivals.
## Method 1 : using factors
data(dipper.histories)
ct <- as.factor( paste("T",1:ncol(dipper.histories), sep=""))
attr(ct,"nan")<-nrow(dipper.histories)
dipper.cjs <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(ct,drop=c(1,6,7)), dipper.histories )

## Method 2 : same thing using 2-d matrices
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
# The following extracts 2-D matrices of 0s and 1s
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x3+x4+x5+x6+x7, ~x2+x3+x4+x5, dipper.histories )

## Values in the 1st column of capture covariates do not matter
x3.a <- x3
x3.a[,1] <- 999
dipper.cjs2 <- F.cjs.estim( ~x3.a+x4+x5+x6+x7, ~x2+x3+x4+x5, dipper.histories )
# compare dipper.cjs2 to dipper.cjs

## Values in the last column of survival covariates do not matter
x3.a <- x3
x3.a[,ncol(dipper.histories)] <- 999
dipper.cjs2 <- F.cjs.estim( ~x3+x4+x5+x6+x7, ~x2+x3.a+x4+x5, dipper.histories )
```

```
# compare dipper.cjs2 to dipper.cjs


## A plot to compare the link functions
sine.link <- function(eta){ ifelse( eta < -4, 0, ifelse( eta > 4, 1, .5*(1+sin(eta*pi/8)))) }
eta <- seq(-5,5, length=40)
p1 <- 1 / (1 + exp(-eta))
p2 <- sine.link(eta)
p3 <- 1.0 - exp( -exp( eta ))
plot(eta, p1, type="l" )
lines(eta, p2, col="red" )
lines(eta, p3, col="blue" )
legend( "topleft", legend=c("logit", "sine", "hazard"), col=c("black", "red", "blue"), lty=1)
```

F.cjs.gof *F.cjs.gof*

Description

Goodness of fit measures for a CJS open-population capture recapture model.

Usage

```
F.cjs.gof( cjsobj, resid.type="pearson", rule.of.thumb = 2, HL.breaks = "deciles" )
```

Arguments

cjsobj	A CJS capture-recapture fitted object from a previous call to F.cjs.estim
resid.type	Type of residual to return. resid.type = 'pearson' produces Pearson residuals. resid.type = 'deviance' produces deviance residuals. Anything other than 'deviance' gives you Pearson residuals.
rule.of.thumb	Rule of thumb to include a cell in one of the chi-square statistics. For example, if rule.of.thumb = 2, the expected count in a cell has to be greater than 2 in order for the cell to be included in the overall Chi-square statistic for that table. No pooling of cells is done. Cells with expected values less than rule.of.thumb are dropped.
HL.breaks	vector of bin break points to use in the Hosmer-Lemeshow statistic. This must be a partition of the interval [0,1], with 0 as lowest break and 1 as max. E.g., if HL.breaks = c(.25,.75), the bins used are [0,.25),[.25,.75),[.75,1]. The default, "deciles", calculates breakpoints such that 10 values are in each. I.e., approximately 0.1 * n expected values are in each of 10 cells.

Details

The "overall" Chi-square test computes the sum of [(h(ij) - Psi(ij))*(h(ij) - Psi(ij))] / Psi(ij) over all "live" cells in the capture-recapture problem. "Live" cells are those following initial captures, prior to and including the occasion when an animal was censoring (died on capture and removed). If an animal was not censored, the "live" cells for it extend from occasion following initial capture to the end of the study. In the above, h(ij) is the 0-1 capture indicator for animal i at occasion j. Psi(ij) is the expected value of h(ij), and is computed as the produce of survival estimates from initial capture to occasion j, times probability of capture at occasion j. Assuming animal i was initially captured at the a-th occasion, Psi(ij) is computed as phi(ia) * phi(i(a+1)) * ... * phi(i(j-1)) * p(ij), where phi(ij) is the modeled estimate of survival for animal i from occasion j to occasion j+1, and p(ij) is the probability of capturing animal i during occasion j.

The other derived GOF tests computed here use h(ij) and its expected value Psi(ij). Test 4 sums observed and expected over individuals. Test 5 sums observed and expected over occasions. The other 3 tests were borrowed from logistic regression by viewing h(ij) as a binary response, and Psi(ij) as its expected value.

Value

A CJS object equivalent to the input crobj, with additional components for GOF testing. Additional components are a variety of goodness of fit statistics. Goodness of tests included are: (1) "Overall" = Chi-square test of overall goodness of fit based on all "live" cells in the capture histories, (2) "Osius and Rojek" = Osius and Rojeck correction to the overall chi-square test, (3) "Test 4" = Chi-square of observed and expected captures by occasion, (4) "Test 5" = Chi-square of observed and expected captures by individual, summed over animals, (5) "Hosmer-Lemeshow" = Hosmer-Lemeshow Chi-square GOF over all occasions and animals, and (6) "ROC" = area under the curve overall classification accuracy of expected values for capture histories. Tests (2), (5), and (6) are based on methods in chapter 5 of Hosmer and Lemeshow (2000).

Specifically, the output object has class c("cjsgof", "cjs", "cr"), contains all the components of the original CJS object, plus the following components:

`gof.chi`	Chi-square statistic for overall goodness of fit based on all "live" cells in the capture-recapture histories.
`gof.df`	Degrees of freedom for overall goodness of fit test.
`gof.pvalue`	P-value for overall goodness of fit.
`or.table`	Chi-square table for the Osius and Rojek correction to the overall GOF test (See p. 153 of Hosmer and Lemeshow (2000)).
`or.chi`	Chi-square statistic for the Osius and Rojek test.
`or.df`	Degrees of freedom for the Osius and Rojek test.
`or.correction`	Correction to the Osius and Rojek test. This is computed as number of unique expected values minus the sum of 1 over the individual cell counts.
`or.rss`	Root sum-of-squares for the Osius and Rojek test, obtained from weighted regression.
`or.z`	Osius and Rojek Z statistic. This is computed as (or.chi - or.df) / sqrt(or.correction + or.rss)
`or.pvalue`	2-tailed Osius and Rojek p-value computed from standard normal distribution and the Osius and Rojek Z statistic.

t4.table	Chi-square table for Test 4, which sums observed and expected captures over individuals. This table has one cell for each occasion.
t4.chi	Chi-square statistic for Test 4, computed from t4.table by summing the chi-square contributions over cells that meet the rule.of.thumb.
t4.df	Degrees of freedom for Test 4. Equal to number of cells meeting rule.of.thumb minus 1.
t4.pvalue	P-value for Test 4 computed from Chi-squared distribution.
t5.table	Chi-square table for Test 5, which sums observed and expected captures over occasions. This table has one cell for each individual.
t5.chi	Chi-square statistic for Test 5, compute from t5.table by summing the chi-square contributions over cells that meet the rule.of.thumb.
t5.df	Degrees of freedom for Test 5. Equal to number of cells meeting rule.of.thumb minus 1.
t5.pvalue	P-value for Test 5 computed from Chi-squared distribution.
HL.table	Chi-square table for the Hosmer-Lemeshow test.
HL.chi	Chi-square statistic for the Hosmer-Lemeshow test.
HL.df	Degrees of freedom for the Hosmer-Lemeshow test.
HL.pvalue	P-value for the Hosmer-Lemeshow test.
roc	Area under the curve statistic for the ability of the "live" cell expected values to classify captures.

Note

Future plans include adding the following: (1) Osius-Rojek = Overall z statistic for GOF over all occasions and animals; and (2) Stukel = Overall z test for appropriateness of the logistic link.

Future plans also include a plot method whereby all tests, especially the ROC, could be assessed graphically.

Print the GOF results in a nice format using print.cjs.

Author(s)

Trent McDonald, WEST Inc., tmcdonald@west-inc.com

References

Hosmer, D. W. and S. Lemeshow. 2000. Applied Logistic Regression, 2nd edition. New York: John Wiley and Sons.

See Also

F.cjs.estim, print.cjs

Examples

```
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x2+x3+x4+x5+x6, ~x1+x2+x3+x4+x5, dipper.histories )
dipper.cjs.gof <- F.cjs.gof( dipper.cjs )
print(dipper.cjs.gof)
```

`F.cjs.simulate` *F.cjs.simulate - Generation of capture histories that follow a CJS model.*

Description

This function generates capture history matrices that follow open Cormack-Jolly-Seber (CJS) models. A super-population approach is taken wherein individuals with unique capture and survival probabilities are randomly 'born' into the realized population and captured. Any CJS model, including those with heterogeneous survival or capture probabilities, can be simulated. Closed populations can also be simulated.

Usage

```
F.cjs.simulate(super.p, super.s, fit, N1 = 1000,
  births.per.indiv = "constant.popln", R = 100)
```

Arguments

`super.p` A matrix or vector of true capture probabilities in the super-population of individuals.

- If `super.p` is a VECTOR, all individuals in the realized population will have the same true capture probabilities, but capture probabilities can vary by occasion. In this case, `length(super.p)` capture occasions will be simulated.
- If `super.p` is a MATRIX, the rows of `super.p` will be randomly selected and used for the capture probabilities of individuals when they are 'born' into the population. Number of rows in `super.p` must be greater than or equal to 1, and does not need to match number of rows in `super.s`. When `super.p` is a matrix, `ncol(super.p)` capture occasions will be simulated.

`super.s` A matrix or vector of true survival probabilities in the super-population of individuals.

- If `super.s` is a VECTOR, all individuals in the realized population will have the same true survival probabilities after they are 'born' into the realized population. If the number of occasions to simulate is NS (see `super.p` above), `super.s` must be of length $NS - 1$.

- If super.p is a MATRIX, the rows of super.p will be randomly selected and used as survival probabilities for individuals when they are 'born' into the population. If the number of occasions to simulate is NS, super.s must have $NS - 1$ columns. The vector super.s[,j] is the set of true survival probabilities for animals alive just after occasion j until just before occasion j+1. Number of rows in super.s must be greater than or equal to 1, and does not need to match number of rows in super.p.

Number of survival probabilities in super.s is one less than NS because survival probabilities apply between capture occasions.

fit
: A previously estimated CJS object. Instead of specifying super.p and super.s, a fitted CJS model can be specified. If either one of super.p or super.s is missing, the (estimated) probabilities in fit will be used for their respective place. That is, if super.p is missing, fit must be present and fit$p.hat will be used for the matrix of true capture probabilities. If super.p is missing, fit must be present and fit$s.hat will be used for the matrix of true survival probabilities. Because capture probabilities for the first occasion are not usually estimable by CJS models, capture probabilities for the first occasion are set equal to 1.0. All members of the realized population will be observed on the first occasion in this case.

N1
: A scalar specifying the initial population size. I.e., N1 individuals will be 'born' into the realized population just before the first sampling occasion.

births.per.indiv
: Either a vector of births per individual in the realized population, or the string "constant.popln" (the default). If births.per.indiv = "constant.popln", the total number of births into the realized population between capture occasions will equal the number of deaths between occasions. In this case, true realized population size will be (exactly) constant through time. If births.per.indiv is a vector of length $NS-1$, then round(N_j*births.per.indiv[,j]) births will occur between occasions j and j+1, where N_j is the true number of individuals in the realized population at occasion j. Values in birth.rate must be 0 or greater. As an example, all animals in the realized population have one offspring between occasions if births.per.indiv = rep(1,NS). Assuming a sex ratio of 50%, all females alive in the population between occasions have one offspring if births.per.indiv = 0.5. All females in the population have two offspring if births.per.indiv = 1.

R
: A scalar specifying the number of replications for the simulation. A total of R independent capture history matrices will be generated.

Details

Some examples: A two-group heterogeneous population contains one group of individuals with one common set of capture probabilities, and another group of individuals with another set of common capture probabilities. A population with one group of individuals having capture probability equal to 0.25, and another group with capture probability equal to 0.75 can be simulated using

- F.cjs.simulate(rbind(rep(0.25,10),rep(0.75,10)), rep(s,9)).

, where s is some survival probability between 0 and 1. If s = 1, a closed (no births or deaths) two-group heterogeneous model is simulated. In this example, the realized population is sampled for 10 occasions.

Non-equal sized heterogeneous groups can be simulated using

- F.cjs.simulate(rbind(matrix(0.25,1,10),matrix(0.75,9,10)), rep(1,9)).

Using this call, approximately 10% of individuals in the realized population will have capture probabilities equal to 0.25, while 90% will have capture probabilities equal to 0.75. Additional groups can be included by including more rows with distinct probabilities in super.p.

A population with heterogeneous capture probabilities proportional to a vector w can be simulated using

- F.cjs.simulate(matrix(w/sum(x), length(w), 10), rep(s,9))

.

A stochastic population that varies around a specified size of N1 = 1000 can be simulated with a statement like

- F.cjs.simulate(rep(0.25,10), rep(s,9), N1=1000, births.per.indiv=rep((1-s)/s,9)).

In this simulation, N(j)*(1-s) individuals die between each occasion, but are replaced because the N(j)*s surviving individuals each have (1-s)/s offspring.

Because of the super-population approach taken here, it is not possible to specify which individuals have which survival or capture probabilities, nor to guarantee that a certain number of individuals in the realized population have capture probabilities equal to any particular value.

Value

A list of length R. Each component of this list is a list of length 2. Each of these R sublists contains the following components:

hists	The simulated capture histories for a particular iteration. This is a matrix with a random number of rows (due to the stochastic nature of captures) and NS columns.
popln.n	A vector of length NS containing the true population sizes at each sampling occasion.

Author(s)

Trent McDonald, WEST Inc. (tmcdonald@west-inc.com)

See Also

F.cjs.estim

Examples

```
## Not run:

## Don't run specified because these examples can take > 10 seconds.

## Simulate constant model, and analyze


ns <- 10
N <- 100
sim.list <- F.cjs.simulate( rep(0.3,ns), rep(0.9,ns-1), N1=N, R=100 )

f.analyze <- function(x){
    fit <- F.cjs.estim( ~1, ~1, x$hists, control=mra.control(maxfn=200, cov.meth=2) )
    if( fit$exit.code == 1 ){
        return( fit$n.hat )
    } else {
        return( rep(NA,ncol(x$hists)) )
    }
}
results <- t(sapply(sim.list, f.analyze))
plot( 1:10, colMeans(results, na.rm=TRUE), xlab="Occasion",
  ylab="Mean population estimate", col="red", type="b")
abline( h=N )

## Plot RMSE by occasion
std <- apply(results, 2, sd, na.rm=TRUE)
bias <- apply(results - N, 2, mean, na.rm=TRUE)
plot( std, bias, type="n" )
text( std, bias, 2:10 )
abline(h=0)
title(main="RMSE by Sample Occasion")

## Show bias when heterogeniety is present
sim.list <- F.cjs.simulate( matrix(c(0.3,.7,.7,.7),4,ns), rep(0.9,ns-1), N1=N, R=100 )
results <- t(sapply(sim.list, f.analyze))
mean.N <- colMeans(results, na.rm=TRUE)
plot( 1:length(mean.N), mean.N, ylim=range(c(mean.N,N),na.rm=TRUE),
  xlab="Occasion", ylab="Mean population estimate", col="red", type="b")
abline( h=N )
abline( h=mean(mean.N), col="red", lty=2)
title(main="Heterogeniety causes negative bias")


## Simulate CJS model, first estimate one
data(dipper.histories)
ct <- as.factor( paste("T",1:ncol(dipper.histories), sep=""))
attr(ct,"nan")<-nrow(dipper.histories)
dipper.cjs <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(ct,drop=c(1,6,7)), dipper.histories )

## Now generate histories from it.
sim.list <- F.cjs.simulate( fit=dipper.cjs, N1=100, birth.rate=rep(1,6), R=100 )
```

```
## Now analyze generated histories using lapply or sapply.  Can fit any model.
## Here we fit the correct model.
f.analyze <- function(x){
    #   write a counter to console, this is not necessary
    i <- get("i", env=.GlobalEnv) + 1
    cat(paste("Iteration", i, "\n"))
    assign("i",i,env=.GlobalEnv)

    ct <- as.factor( 1:ncol(x$hists) )
    fit <- F.cjs.estim( ~tvar(ct,nan=nrow(x$hists),drop=c(1,2)),
        ~tvar(ct,nan=nrow(x$hists),drop=c(1,6,7)),
        x$hists, control=mra.control(maxfn=200, cov.meth=2) )
    if( fit$exit.code == 1 ){
        return( fit$n.hat )
    } else {
        return( rep(NA,ncol(x$hists)) )
    }
}
i <- 0
results <- t(sapply(sim.list, f.analyze))
mean.N <- colMeans(results, na.rm=TRUE)
plot( 1:length(mean.N), mean.N, ylim=range(c(mean.N,N),na.rm=TRUE),
  xlab="Occasion", ylab="Mean population estimate", col="red", type="b")
abline( h=N )
title(main="Time varying CJS model")

## End(Not run)
```

`F.cr.model.avg` *F.cr.model.avg - Model averaging of mark-recapture parameters.*

Description

Computes model averaged estimates of survival, capture probability, or population size estimates from a set of previously fitted MRA objects.

Usage

```
F.cr.model.avg( fits=ls(pattern="^fit"), what="survival", fit.stat="qaicc" )
```

Arguments

`fits`	A character vector of MRA fitted object names. Each will be retrieved from the global environment (i.e., .GlobalEnv) using `get` and tested to make sure they are MRA fitted objects. If not, a warning is issued and the object is ignored. If the object is an MRA model, it is included in model averaging. The default value will use any object whose name starts with "fit". For example, if fitted objects are named "fit1.01", "fit1.02", and "fit1.03", `fits` equal to `c("fit1.01",`

"fit1.02", "fit1.03") or ls(pat="^fit1"), will average statistics in these three objects.

what — A text string naming the parameter to average. Choices are "survival" (the default), "capture", and "n.hat". Only the first character is inspected (e.g., "c" is equivalent to "capture").

fit.stat — A string (scalar) naming the model fit statistic to use when computing model weights. Possible values are: "qaicc" (the default), "qaic", "aicc", and "aic".

Details

Each model is checked for convergence prior to including in the model averaging process. The test for whether a model converged is (fit$exit.code == 1) & (fit$cov.code == 0) & (fit$df > 0), where fit is the fitted object. If the model did not converge, it is excluded from model averaging.

Conditional and unconditional variance estimates are computed following Burnham and Anderson 2002 (pages 150 and 162 and surrounding).

If what = "n.hat", the returned object is suitable for printing using print.nhat and plotting using plot.cjs. If what = "survival" or "capture", the returned object is unclassed and the user is responsible for printing and plotting.

Value

If what = "survival" or "capture", the return is a list object containing the following components:

fit.table — A data frame, sorted by fit.stat, containing model names, fit statistics, delta fit statistics, and model averaging weights.

s.hat or p.hat — A matrix of size nan X ns containing model averaged estimates of survival or capture probability.

se.s.hat or se.p.hat — A matrix of size nan X ns containing the unconditional (on model selection) estimate of standard error for the corresponding model averaged statistic in s.hat or p.hat. Unconditional variances are computed using formulas in Burnham and Anderson (2002, pages 150 and 162)

se.s.hat.conditional or se.p.hat.conditional — A matrix of size nan X ns containing the conditional estimate of standard error for the corresponding model averaged statistic in s.hat or p.hat. These estimates of variance are conditional on model selection.

mod.selection.proportion — A matrix of size nan X ns containing the proportion of variance due to model selection uncertainty. Values in this matrix are simply the difference between unconditional variance and conditional variance, divided by the unconditional variance.

If what = "n.hat", the return is a list of class "n.hat" containing the following components:

fit.table — A data frame, sorted by fit.stat, containing model names, fit statistics, delta fit statistics, and model averaging weights.

n.hat — A vector of length ns containing model averaged estimates of population size.

`se.n.hat` A vector of length `ns` containing the unconditional (on model selection) estimate of standard error for the corresponding model averaged population size.

`se.n.hat.conditional` A vector of length `ns` containing the conditional estimate of standard error for the corresponding model averaged population size.

`mod.selection.proportion` A vector of length`ns` containing the proportion of variance due to model selection uncertainty. Values in this matrix are simply the difference between unconditional variance and conditional variance, divided by the unconditional variance.

`n.hat.lower` A vector of length `ns` containing lower 95% confidence limits for the corresponding population size estimate.

`n.hat.upper` A vector of length `ns` containing upper 95% confidence limits for the corresponding population size estimate.

`nhat.v.meth` Scalar indicating the type of variance estimate used. Values are: `4` = "(Model averaged Huggins variance)", `5` = "(Model averaged Huggins variance + higher terms)", or `6` = "(Model averaged McDonald and Amstrup)". See help for `F.cjs.estim` for more explanation.

Author(s)

Original routine by Eric Regehr, US Fish and Wildlife. Modified for MRA by Trent McDonald, WEST-INC, tmcdonald@west-inc.com

References

Burnham, K. and D. Anderson (2002) "Model Selection: A practical guide". Cambridge University Press.

See Also

`F.cjs.estim`, `F.huggins.estim`, `F.fit.table`, `plot.cjs`

Examples

```
## Fit several CJS model to dipper data. Model average survival

## Time varying survival and capture (true CJS model)
data(dipper.histories)
ct <- as.factor( paste("T",1:ncol(dipper.histories), sep=""))
attr(ct,"nan")<-nrow(dipper.histories)
dipper.01 <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(ct,drop=c(1,6,7)), dipper.histories )

## Linear trend in survival
cT <- 1:ncol(dipper.histories)
dipper.02 <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(cT, nan=nrow(dipper.histories)),
  dipper.histories )
```

```
## No trend in survival
dipper.03 <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~1, dipper.histories )

## Model average
mod.avg.surv <- F.cr.model.avg( ls(pat="^dipper.[0-9]"), what="s", fit.stat="aicc" )

mod.avg.n <- F.cr.model.avg( ls(pat="^dipper.[0-9]"), what="n", fit.stat="aicc" )

## Plot
plot(mod.avg.n)
```

`F.cr.model.matrix` *Capture-Recapture model matrix*

Description

Returns two model matrices for capture-recapture modeling. Both are in the form of (giant) 2D matrices.

Usage

```
F.cr.model.matrix(capture, survival, nan, ns)
```

Arguments

`capture`	Formula for the capture model. Must be a formula object with no response, then ~, followed by the names of 2-D arrays of covariates to fit in the capture model. For example: capture = ~ age + sex, where age and sex are matrices.
`survival`	Formula for the survival model. Must be a formula object with no response, then ~, followed by the names of 2-D arrays of covariates to fit in the survival model. For example: capture = ~ age + sex, where age and sex are matrices.
`nan`	Number of individuals in the model. This is necessary for the `ivar` and `tvar` functions to work. Normally, `nan` = number of rows in capture history matrix. No default value.
`ns`	Number of sampling occasions. Normally, `ns` = number of columns in the capture history matrix.

Details

This routine is intended to be called internally by model fitting routines of MRA. General users should never have to call this routine.

This routine uses a call to `eval` with a model frame, and calls the R internal `model.matrix` to resolve the matrices in the formula. All matrices specified in the models should be in the current scope and accessible to both `eval` and `model.matrix`.

This routine calls `F.3d.model.matrix` twice. `F.3d.model.matrix` does all the work.

Value

A list containing the following components:

capX	A NAN by IX+(NX*NS) matrix containing covariate values for the capture model. Matrices specified in the model are column appended together. NAN = nrow(x) where x is a 2-D matrix in the model (i.e., number of animals). NS = ncol(x) (i.e., number of capture occasions). NX = number of matrices specified in the model. IX = 1 if an intercept is included, 0 otherwise. The j-th covariate matrix specified in the model can be accessed directly with capX[, IX+1+(NS*(j-1)):(NS*j)].
surX	A NAN by IY+(NY*NS) matrix containing covariate values for the survival model. Matrices specified in the model are column appended together. NAN = nrow(x) where y is a 2-D matrix in the model (i.e., number of animals). NS = ncol(y) (i.e., number of capture occasions). NY = number of matrices specified in the model. IY = 1 if an intercept is included, 0 otherwise. The j-th covariate matrix specified in the model can be accessed directly with capY[, IY+1+(NS*(j-1)):(NS*j)].
n.cap.covars	Number of matrices specified in the capture model (NX above).
n.sur.covars	Number of matrices specified in the survival model (NY above).
cap.intercept	TRUE or FALSE depending on whether an intercept was included in the capture model
sur.intercept	TRUE or FALSE depending on whether an intercept was included in the survival model
cap.vars	Vector of names for the NX covariates in the capture model.
sur.vars	Vector of names for the NY covariates in the survival model.

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cjs.estim, model.matrix, eval

Examples

```
# Synthetic example with 10 animals and 5 occasions
nan <- 10
ns <- 5
sex <- matrix( as.numeric(runif( nan ) > 0.5), nrow=nan, ncol=ns )
x <- matrix( runif( nan*ns ) , nrow=nan, ncol=ns )
F.cr.model.matrix( capture= ~ sex + x, survival= ~ -1 + x, nan, ns )
```

F.fit.table *F.fit.table - Produce a summary table of model fit statistics.*

Description

A utility function to compile a table of fit statistics from a list of MRA fitted objects contained in the .GlobalEnv (i.e., 'working') environment. The table produced by this routine contains model name, fit statistics (AICc or QAICc), and is ranked by (sorted by) one of these fit statistics.

Usage

```
F.fit.table( fits=ls(pattern="^fit"), rank.by= "qaicc", plausible.p=0.01 )
```

Arguments

fits	A character vector of MRA fitted object names to include in the summary table. These names do not need to have a common root name. The default value will use any object whose name starts with "fit" in the working directory (.GlobalEnv). An an example, if fitted objects are named "fit1.01", "fit1.02", and "fit1.03", fits should equal c("fit1.01", "fit1.02", "fit1.03"), or ls(pat="^fit1"), assuming no other objects in the working directory start with "fit1".
rank.by	A string (scalar) naming the model fit statistic to include in the summary table. The resulting table is sorted by this statistic. Possible values are: "qaicc" (the default), and "aicc".
plausible.p	A scalar specifying the cut-point in rank.by weight to use during determination of 'plausible' models. A model is defined to be 'plausible' if it has rank.by weight greater than plausible.p OR if the model's log-likelihood is greater than the minimum log likelihood amongst those that have rank.by weight greater than plausible.p. See explanation of plausible in Value section below.

Details

A rudimentary check for convergence is done on each fitted model. If this routine believes a model did not converge, the model is included in the table, but the model's fit statistics are set to Inf. The test for whether a model converged is (fit$exit.code == 1) & (fit$cov.code == 0) & (fit$df > 0), where fit is the fitted object.

Fitted objects are pulled from the .GlobalEnv environment. Usually, this is the current working directory. Currently, there is no way to pull fits from another environment, but a savvy R programmer could modify the where argument of the get command embedded in this routine.

Value

A data frame, sorted by rank.by, with the following columns

model.num	Model number assigned by this routine, equal to the position of the model in the input list of fits.
model.name	Name of the fitted object.
converged	Logical values indicating whether this routine thinks the model converged or not. Value is TRUE if the this routine thinks the model converged, FALSE otherwise.
n.est.parameters	Number of estimable parameters in the model. This is MRA's guess at the number of estimable parameters in the model, not length of the coefficient vector.
n.coefficients	Number of coefficients in the model. This is length of the coefficient vector without regard to number of estimable parameters. If n.coefficients > n.est.parameters, the model is not full rank, and at least one coefficient is probably not estimable.
loglike	value of the log likelihood evaluated at the maximum likelihood parameters.
aicc	AIC of the model including the small sample correction = AIC + (2*df*(df+1)) / (nan - df - 1)
delta.aicc	Difference between AICc for the model and the minimum AICc in the table.
aicc.wgt	AICc model weights. These weights equal exp(-.5*(delta.aicc)), scaled to sum to 1.0,
qaicc	QAIC of the model including the small sample correction = QAIC + (2*df*(df+1))/(nan - df - 1)
delta.qaicc	Difference between QAICc for the model and the minimum QAICc in the table.
qaicc.wgt	QAICc model weights. These weights equal exp(-.5*(delta.qaicc)), scaled to sum to 1.0,
plausible	Indicates 'plausible' models as defined by Bromaghin et al. (2013). The value in this column is TRUE if the model has rank.by weight greater than plausible.p OR if the model's log-likelihood is greater than the minimum log likelihood amongst those that have rank.by weight greater than plausible.p. The second condition in this scheme includes a model structure as 'plausible' when its log-likelihood is relatively high but it has been heavily penalized by the number of parameters. When the likelihood is parameterized to contain two or more linear models, this second condition is a reasonable criterion when model selection is done in a step-wise fashion on each model separately (see Bromaghin et al., 2013).

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

References

Bromaghin, J.F., McDonald, T. L., and Amstrup, S. C., (2013) "Plausible Combinations: An improved methods to evaluate the covariate structure of Cormack-Jolly-Seber mark-recapture models", Open Journal of Ecology, v.3, p. 11-22. (included in vignettes)

See Also

F.cjs.estim, F.huggins.estim, vignette("Bromaghin_etal_2013_OJE")

Examples

```
## Fit several CJS model to dipper data. Summarize fits.

## Time varying survival and capture (true CJS model)
data(dipper.histories)
ct <- as.factor( paste("T",1:ncol(dipper.histories), sep=""))
attr(ct,"nan")<-nrow(dipper.histories)
dipr.01 <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(ct,drop=c(1,6,7)), dipper.histories )

## Linear trend in survival
cT <- 1:ncol(dipper.histories)
dipr.02 <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(cT, nan=nrow(dipper.histories)),
  dipper.histories )

## No trend in survival
dipr.03 <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~1, dipper.histories )

## Summary table
F.fit.table( ls(pat="^dipr") )
```

`F.huggins.estim` *F.huggins.estim - Estimation of Huggins closed population capture-recapture model.*

Description

Estimates Huggin's closed population capture-recapture models with individual, time, and individual-time varying covariates using the "regression" parameterization of Amstrup et al (2006, Ch 9). For live recaptures only. A logistic link function is used to relate probability of capture to external covariates.

Usage

```
F.huggins.estim(capture, recapture=NULL, histories, remove=FALSE, cap.init, recap.init,
    nhat.v.meth=1, df=NA, link="logit", control=mra.control())
```

Arguments

`capture`	Formula specifying covariates to included in the initial capture probability model. Must be a formula object without a response. Specify ~, followed by the names of 2-D arrays of covariates to relate to initial capture probability. For example: 'capture = ~ age + sex', where age and sex are matrices of size NAN X NS containing the age and sex covariate values. NAN = number of individuals = number of rows in `histories` matrix (see below). NS = number of samples = number of columns in histories matrix (see below). Number of matrices specified in the initial capture model is assumed to be NX. Specify intercept only

model as 'capture = ~ 1'. Specify model without an intercept using 'capture = ~ -1 + x'.

`recapture` Formula specifying covariates to included in the recapture probability model, or NULL. Should be specified the same way as the `capture` model. For example: 'recapture = ~ behave + sex'. The number of covariates specified in the recapture model is NY. Total number of parameters this routine attempts to estimate is NX+NY. See `df` argument. If `recapture=NULL`, no recapture model (or the empty model) is estimated. In this case, recapture probabilities equal initial capture probabilities and both depend on the model in `capture`. Note that NULL models are specified without the ~.

`histories` A NAN X NS = (number of individuals) X (number of capture occasions) matrix containing capture histories. Capture histories are comprised of 0's and 1's only. 0 in cell (i,j) of this matrix means individual i was not captured on occasion j, 1 in cell (i,j) means individual i was captured on occasion j and released live back into the population. Because the population being sampled is assumed closed, deaths on capture (known removals) are not allowed. If deaths on capture occurred and an estimate of N at the beginning of the study is sought, remove the entire history, estimate N using this routine from the remaining histories, and add back the number of deleted histories.

`remove` A logical scalar, or vector of logical values, specifying which `capture` covariates to remove from the `recapture` model. By default (`remove=FALSE`), no capture covariates are removed, meaning all terms in the model for initial capture also appear in the model for recaptures *with the same coefficient values*. See Details section. If `remove` is a vector, each entry specifies whether the corresponding effect in `capture` should be removed from the `recapture` model. If `remove` is shorter than NX (the number of matrices in `capture`), it is replicated to have length NX.

`cap.init` (optional) Vector of initial values for coefficients in the initial capture model. One element per covariate in `capture`. This parameter does not usually need to be specified.

`recap.init` (optional) Vector of initial values for coefficients in the recapture model. One element per covariate in `recapture`. This parameter does not usually need to be specified.

`nhat.v.meth` Integer specifying method for computing variance estimate for the population size estimate. Currently, only `nhat.v.meth` = 1 is implemented. Plans are for `nhat.v.meth` = 2 to be a boot strap estimate of variance. `nhat.v.meth` = 1 is a delta method estimator utilizing the derivative of P(ever captured) w.r.t. the capture parameters. This is the same estimator as used in program MARK.

`df` External (override) model degrees of freedom to use during estimation. If `df` == NA, the number of parameters is estimated from the rank of the matrix of 2nd derivatives or Hessian, depending on `cov.meth` parameter. If `df` <= 0, the number of parameters will be set to NX+NY = the number of estimated coefficients. Otherwise, if `df` > 0, the supplied value is used. Only AIC, QAIC, AICc, and QAICc are dependent on this value (in their penalty terms).

`link` The link function to be used. The link function converts linear predictors in the range (-infinity, infinity) to probabilities in the range (0,1). Valid values for the

link function are "logit" (default), "sine", and "hazard". (see Examples in help for `F.cjs.estim` for a plot of the link functions)

- The "logit" link is $\eta = log(\frac{p}{1-p})$ with inverse $p = \frac{1}{1+exp(-\eta)}$.
- The "sine" link is $\eta = \frac{8asin(2p-1)}{\pi}$, which ranges from -4 to 4. The inverse "sine" link is $p = \frac{1+sin(\eta\pi/8)}{2}$ for values of η between -4 and 4. For values of η < -4, p = 0. For values of η > 4, p = 1. Scaling of the sine link was chosen to yield coefficients roughly the same magnitude as the logit link.
- The "hazard" link is $\eta = log(-log(1-p))$, with inverse $1-exp(-exp(\eta))$. The value of p from the inverse hazard link approaches 0 as η decreases. For values of η > 3, p = 1 for all intents and purposes.

`control` A list containing named control parameters for the minimization and estimation process. Control parameters include number of iterations, covariance estimation method, etc. Although the default values work in the vast majority of cases, changes to these variables can effect speed and performance for ill-behaved models. See `mra.control()` for a description of the individual control parameters.

Details

This routine compiles all the covariate matrices, then calls a Fortran routine to maximize the Huggins closed population likelihood. So-called heterogeneous models that utilize mixture distributions for probability of capture cannot be fitted via this routine.

If `remove=FALSE` (default) the models for initial capture and subsequent recapture are,

$$p_{ij} = \beta_0 + \beta_1 x_{ij1} + \ldots + \beta_a x_{ija}$$

and

$$c_{ij} = \beta_0 + \beta_1 x_{ij1} + \ldots + \beta_a x_{ija} + \gamma_0 + \gamma_1 z_{ij1} + \ldots + \gamma_b z_{ijb}$$

where the x's and z's are covariate values specified in `capture` and `recapture`, respectively, and the β's and γ's are estimated coefficients. (For brevity, 'a' has been substituted for NX, 'b' for NY.) In other words, by default all effects in the capture model also appear in the recapture model *with the same estimated coefficients*. This is done so that capture and recapture probabilities can be constrained to equal one another. If `capture=~1` and `recapture=NULL`, capture and recapture probabilities are constant and equal to one another. If `capture=~x1` and `recapture=NULL`, capture and recapture probabilities are equal, and both are the exact same function of covariate `x1`. A simple additive behavioral (trap happy or trap shy) effect is fitted by specifying an intercept-only model for recaptures, i.e., `capture=~x1+x2+...+xp` and `recapture=~1`.

When a Huggin's model object is printed using the default print method (`print.hug`), a "C" (for "capture") appears next to coefficients in the recapture model that are also in the initial capture model. These coefficients are fixed in the recapture model. A "B" (for "behavioral") appears next to free coefficients in the recapture model that *do not* appear in the initial capture model.

If `remove` is something other than FALSE, it is extended to have length NX, and if element *i* equals TRUE, the *i*-th effect in the capture model is removed from the recapture model. If `remove=c(FALSE, TRUE, FALSE)`, `capture=~x1+x2`, and `recapture=~x1+x3`, the models for initial capture and subsequent recapture are,

$$p_{ij} = \beta_0 + \beta_1 x_{ij1} + \beta_2 x_{ij2}$$

and

$$c_{ij} = \beta_0 + \beta_2 x_{ij2} + \gamma_0 + \gamma_1 x_{ij1} + \gamma_2 x_{ij3}.$$

Note that x1 appears in the recapture equation, but with a different estimated coefficient. If remove=TRUE, all capture effects are removed from the recapture model and the models are completely separate.

The ability to remove terms from the recapture model adds modeling flexibility. For example, if initial captures were hypothesized to depend on the variable sex, but recaptures were hypothesized to be constant (no sex effect), the arguments to fit this model would be capture=~sex, recapture=~1, and remove=TRUE. A pure time-varying model with different time effects in the initial and subsequent capture models can be fitted using capture=~tvar(1:ns,nan), recapture=~tvar(1:ns,nan), and remove=TRUE. In this case, the same model, but parameterized differently, can be fitted with remove=FALSE.

See Details of help(F.cjs.estim) for ways that 2-d matrices, 1-d vectors, and 1-d factors can be specified in the capture and recapture models.

If argument trace in a call to mra.control is set to something other than 0, a log file named mra.log is written to the current directory. See mra.control for actions associated with values of trace. CAREFUL: mra.log is overwritten each run.

Values in 2-d Matrix Covariates: Even though covariate matrices are required to be NAN x NS (same size as capture histories), there are not that many recapture parameters. Recapture parameters for the first occasion are not defined. For all covariates in the recapture model, only values in columns 2:ncol(histories) are used. See examples for demonstration.

Value

An object (list) of class c("hug","cr") with many components. Use print.hug to print it nicely. Use names(fit), where the call was fit <- F.huggins.estim(...), to see names of all returned components. To see values of individual components, issue commands like fit\$n.hat, fit\$se.n.hat, etc.

Components of the returned object are as follows:

histories	The input capture history matrix. Size NAN x NS
aux	Auxiliary information, mostly stored input values. This is a list containing: \$call, \$nan=number of individuals, \$ns=number of samples, \$nx=number of coefficients in the initial capture model, \$ny=number of coefficients in recapture model, \$cov.name=names of the covariates in both models (initial capture covariates first, then recapture covariates), \$ic.name=name of capture history matrix, \$mra.version=version number of this package, \$R.version=R version used, \$run.date=date the model was estimated.
loglik	Value of the Huggins log likelihood at it's maximum.
deviance	Model deviance = -2*loglik. This is relative deviance because all covariates are individual and time varying. When individual covariates are present, a saturated likelihood cannot be computed. Use this to compute deviance differences only.
aic	AIC for the model = deviance + 2*(df). df is either the estimated number of independent parameters (by default), or NX+NY, or a specified value, depending on the input value of df parameter.
aicc	AIC with small sample correction = AIC + (2*df*(df+1)) / (NAN - df - 1)

capcoef	Vector of estimated coefficients in the initial capture model. Length NX.
se.capcoef	Vector of estimated standard errors for coefficients in initial capture model. Length NX.
recapcoef	Vector of estimated coefficients in the recapture model. Length NY.
se.surcoef	Vector of standard errors for coefficients in recapture model. Length NY.
covariance	Variance-covariance matrix for the estimated model coefficients. Size (NX+NY) X (NX+NY).
p.hat	Matrix of estimated initial capture probabilities computed from the model. Size of this matrix is NAN x NS. Cell (i,j) is estimated probability of first capture for individual i during capture occasion j.
se.p.hat	Matrix of standard errors for estimated initial capture probabilities. Size NAN x NS.
c.hat	Matrix of estimated recapture probabilities computed from the model. Size NAN x NS. Cell (i,j) is estimated probability of capturing individual i during occasion j given that it was initially captured prior to j.
se.c.hat	Matrix of standard errors for estimated recapture probabilities. Size NAN X NS.
df	Number of estimable parameters in the model. df is either the estimated number of independent parameters (by default) based on rank of the variance matrix, or NX+NY, or a specified value, depending on the input value of df parameter.
message	A string indicating whether the maximization routine converged.
exit.code	Exit code from the maximization routine. Interpretation for exit.code is in message.
cov.code	A code indicating the method used to compute the covariance matrix.
cov.meth	String indicating method used to compute covariance matrix. Interprets cov.code.
n.hat	The Huggins estimate of population size. This estimate is sum(1/ pstar(i)), where pstar(i) is probability of observing individual i, which equals 1 - p.hat[i,1]*p.hat[i,2]* ... *p.hat[i,NS], where p.hat is the returned value of p.hat.
se.n.hat	Estimated standard error of n.hat. Computed using method specified in nhat.v.meth.
n.hat.lower	Lower limit of log based confidence interval for n.hat.
n.hat.upper	Upper limit of log based confidence interval for n.hat.
n.hat.conf	Confidence level for the interval on n.hat. Currently, confidence level cannot be changed from 95%.
nhat.v.meth	Code for method used to compute variance of n.hat. Currently, this is 1 only.
num.caught	Number of individuals ever captured = number of rows in the histories matrix.
n.effective	Effective sample size = number of observed individuals times number of occasions = NAN * NS

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

References

Huggins, R. M. 1989. On the statistical analysis of capture experiments. Biometrika 76:133-140.

Amstrup, S. C., T. L. McDonald, and B. F. J. Manly (editors). 2005. Handbook of Capture-Recapture Analysis. Princeton University Press.

See Also

print.hug, F.cjs.estim

Examples

```
# Fake the data for these examples
set.seed(3425)
ch.mat <- matrix( round(runif(30*5)), nrow=30, ncol=5)
ch.mat <- ch.mat[ apply(ch.mat,1,sum) > 0, ]  # no zero rows allowed
ct <- as.factor(1:ncol(ch.mat))
attr(ct,"nan") <- nrow(ch.mat)   # used to fit time varying factor
sex <- round(runif(nrow(ch.mat)))   # fake sex
attr(sex,"ns") <- ncol(ch.mat)

# Models parallel to the 8 Otis et al. models.
# see Amstrup et al. (2005, p. 77)

# Constant model (model M(0)).
hug.0 <- F.huggins.estim( ~1, NULL, ch.mat )

# Time varying model (model M(t))
hug.t <- F.huggins.estim( ~tvar(ct), NULL, ch.mat)

# Additive Behavioral model (model M(b))
hug.b <- F.huggins.estim( ~1, ~1, ch.mat )

# Time and Behavioral model (model M(tb))
hug.tb <- F.huggins.estim( ~tvar(ct), ~1, ch.mat )

# Individual effects (model M(h))
hug.h <- F.huggins.estim( ~ivar(sex), NULL, ch.mat )

# Individual and Behavioral effects (model M(bh))
hug.bh <- F.huggins.estim( ~ivar(sex), ~1, ch.mat )

# Individual and time effects (model M(th))
hug.th <- F.huggins.estim( ~ivar(sex)+tvar(ct), NULL, ch.mat )

# Individual, time, and behavoral effects (model M(tbh))
hug.tbh <- F.huggins.estim( ~ivar(sex)+tvar(ct), ~1, ch.mat )

# Time varying initial captures, recaptures are constant and depend on sex.
hug.custom1 <- F.huggins.estim( ~tvar(ct), ~ivar(sex), ch.mat, remove=TRUE )

# Compare hug.custom1 to the following: Time varying initial captures with
```

```
# time varying recaptures that depend on sex.
hug.custom2 <- F.huggins.estim( ~tvar(ct), ~ivar(sex), ch.mat, remove=FALSE )

# Values in first column of recapture covariates do not matter.
# Below, mod.1 and mod.2 are identical.
mod.1 <- F.huggins.estim( ~tvar(ct), ~tvar( c( 0,1,2,3,4), nrow(ch.mat)), ch.mat, remove=TRUE)
mod.2 <- F.huggins.estim( ~tvar(ct), ~tvar( c(-9,1,2,3,4), nrow(ch.mat)), ch.mat, remove=TRUE)
```

`F.sat.lik` *F.sat.lik*

Description

Calculate the log likelihood of a fully saturated time varying CJS model. Use to convert the relative deviance output by `F.cjs.estim` to actual deviance.

Usage

```
F.sat.lik(ch)
```

Arguments

`ch`	A capture history matrix consisting of 0's, 1's, and 2's.

Details

The number reported as `deviance` by `F.cjs.estim` is relative deviance, calculated as -2*log(likelihood). IF THERE ARE NO INDIVIDUAL-VARYING COVARIATES in the model, it is possible to compute the theoretical log-likelihood for a set of data assuming perfect prediction. This is the saturated log-likelihood. The actual deviance of a model is the deviance of the model relative to this theoretical maximum, computed as -2*((saturated log-likelihood) - 2*(model log-likelihood)).

In the parameterization of `F.cjs.estim`, all covariates are potentially individual and time varying, and in this case the saturated log-likelihood is unknown. Consequently, the saturated likelihood is not often needed in MRA. This routine was included as a utility function because the saturated likelihood is handy in some cases, including parametric bootstrapping to estimate C-hat.

Assuming `cjs.fit` is an estimated CJS model with time varying covariates only fit to histories in `cjs.hists`, compute deviance as

`-F.sat.lik(cjs.hists) - 2*cjs.fit\$loglik = cjs.fit\$deviance - F.sat.lik(cjs.hists)`

Value

A scalar equal to the value of the saturated CJS log-likelihood. The saturated log-likelihood is the theoretical best predictive model possible, and actual deviance is calculated relative to this. See Examples.

Note

CAUTION: This routine works for time varying models only. If individual-varying or individual-and-time-varying covariates are fitted in the model, the routine cannot sense it and will run but yield an incorrect answer. Use relative deviance reported by `F.cjs.estim` in this case.

Also, this routine will not run if animals have been removed (censored). I.e., the capture history matrix cannot have any 2's in it. Use relative deviance reported by `F.cjs.estim` when animals have been removed.

Author(s)

Eric V. Regehr (USGS, eregehr@usgs.gov) and Trent McDonald (WEST Inc., tmcdonald@west-inc.com)

References

Look up "saturated model" in the program MARK help file for the equations implemented by this function.

See Also

`F.cjs.estim`

Examples

```
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x2+x3+x4+x5+x6, ~x1+x2+x3+x4+x5, dipper.histories )
deviance <- -F.sat.lik( dipper.histories ) - 2*dipper.cjs$loglik
```

`F.step.cjs` *F.step.cjs - Stepwise model selection for CJS models.*

Description

Conducts automated stepwise model selection of CJS models. Selection includes forward steps followed by backward looks. Steps consider covariates in both the survival and probability of capture models.

Usage

```
F.step.cjs(cap.covars, surv.covars, fit.crit = "qaicc", signif.drop = 0,
  signif.increase = 0, plot = TRUE, ...)
```

Arguments

cap.covars	A character vector containing the names of single covariates or combinations of covariates to consider in the capture equation. See Details.
surv.covars	A character vector containing the names of single covariates or combinations of covariates to consider in the survival equation. See Details.
fit.crit	A character scalar specifying the model fit criterion to use during each step. This criterion will be used to rank variables during the forward steps and backward looks. Possible values are "qaicc" (the default), "aic", "qaic", and "aicc". During forward (addition) steps, the variable(s) that decreased fit.crit the most will be added to the model, assuming that the decrease is greater than or equal to abs(signif.drop) (see signif.drop for more). During backward (elimination) looks, the variable that increases fit.crit the most will be removed from the model, assuming that the increase is greater than or equal to abs(signif.increase).
signif.drop	A scalar specifying the decrease in fit.crit that should be considered "significant" during forward steps. This argument controls stopping and is functionally equivalent to the alpha-to-enter parameter of classical stepwise routines. Stepwise selection stops when the minimum fit.crit on the current iteration minus the minimum fit.crit on the previous iteration is greater than or equal to signif.drop. This means signif.drop should be less than or equal to 0, unless for some odd reason, steps that add variables without predictive abilities are desired. If signif.drop = 0 (the default), steps are halted when no variables decrease fit.crit. Thus, the default method yields the minimum fit.crit model among those considered during stepwise selection. If signif.drop = -2, steps are halted when fit.crit decreases by less than 2 (or increases) between the current and previous steps.
signif.increase	A scalar specifying the increase in fit.crit that should be is considered "significant" during backward looks. This argument controls elimination, but not stoppage. A variable in the current model is a candidate for removal if, upon removal, fit.crit increases by less than or equal to signif.increase. This means signif.increase should be greater than or equal to 0 to be meaningful, unless no variables should be removed, in which case set signif.increase = -Inf (or some other negative number). If 2 or more variables are candidates for removal during a single backward look step, the variable which increases fit.crit the most will be removed (the other will remain). If signif.increase = 0 (the default), variables are left in the model if they increase fit.crit when removed. If signif.increase = 2, variables in the current model are left in the model if they increase fit.crit by more than 2 units when removed.
plot	A scalar specifying whether the minimum fit.crit at the end on each forward-backward step should be plotted. This plot can be useful for assessing which variable(s) cause relatively "big" and which cause relatively "little" jumps in fit.crit.
...	Additional arguments to F.cjs.estim, like histories= , nhat.v.meth=, c.hat=, control=, etc.

Details

Elements of both the cap.covars and surv.covars arguments can specify the names of single covariates or sets of covariates to consider as a group to be included or excluded together. For example, if cap.covars = c("sex","ageclass1 + ageclass2 + ageclass3"), the routine will include and exclude sex as a single covariate with 1 estimated coefficient and ageclass1 + ageclass2 + ageclass3 as a set of 3 covariates (with 3 estimated coefficients) to be included and excluded as a set. This is useful if factor covariates are pre-coded as indicator variables. In the example above, specifying ageclass1 + ageclass2 + ageclass3 would make sense if age of an individual was classified into 1 of 4 classes, and ageclassX was a matrix with elements equal to 1 if the individual was in age class X during a capture occasion, and 0 otherwise.

Specifying a term like a + b + ab would ensure that main effect matrices (a and b) are included whenever the interaction matrix (ab) is included during model selection. However, this way of including interactions will only be useful if the main effects are not considered separately. That is, specifying cap.covars = c("a", "b", "a + b + ab") will not work because the routine does not know that "a" and "b" are components of "a + b + ab". Nonsense models like "a + b + ab + a + b" could result. Thus, this routine is likely only useful for terms that do not include interactions. A useful way to proceed in this case may be to use stepwise to select a model of main effects, then consider interactions.

Time varying and individual varying variables can be specified using the tvar and ivar functions in the elements of cap.covars and surv.covars. For example, specifying "tvar(year,nan=nrow(ch))" as an element of surv.covars, where year = 1:ncol(ch) and ch is the capture history matrix, will fit a linear year effect (1 coefficient) when this element is added during stepwise selection. Likewise, factors are preserved during selection. If year in the above example had been a factor (i.e., year = as.factor(1:ncol(ch))), separate effects for each year (ncol(ch) - 1 coefficients) would have been fitted when this effect came up for consideration during stepwise selection.

The variable to add or eliminate is selected after all variables in both the capture and survival models are considered. That is, the variable that decreases (or increases) fit.crit the most over both models is added. Selection does not iteratively fix one model and select variables in the other. For example, on one step, the variable that decreases fit.crit the most may be a survival covariate, while on the next step the variable that decreases fit.crit the most may be a capture covariate.

Value

The final CJS model resulting from application of stepwise model selection. This object is a valid MRA CJS model and has class 'cjs'. See help for F.cjs.estim for a description of the components of this object.

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cjs.estim, F.fit.table

Examples

```
#   Aquire data and intermediate variables
```

```
data(dipper.histories)
data(dipper.males)
ch <- dipper.histories
males <- dipper.males
ns <- ncol(ch)
nan <- nrow(ch)

#   Construct covariates
small.t <- as.factor( paste("T",1:ns, sep=""))
post.flood <- as.numeric( 1:ns >= 4 )
year <- 1:ns - (ns / 2)
males.postflood <- outer( c(males), post.flood ) # individual and time varying

print(dim(males.postflood))

#   Attach attributes to covariates.  For convienence only.
attr(small.t, "nan") <- nan
attr(small.t, "drop") <- c(1,2)
attr(year, "nan") <- nan
attr(post.flood, "nan") <- nan
attr(males, "ns") <- ns

#   Define pool of variables to be considered in each model
cap.vars <- c("tvar(small.t)","tvar(year)")
surv.vars <- c("tvar(small.t)","tvar(year)", "tvar(post.flood)", "ivar(males)",
  "males.postflood")

#   Do stepwise selection
final.fit <- F.step.cjs( cap.vars, surv.vars, histories=ch,
  control=mra.control(maxfn=500, cov.meth=2) )
```

`F.update.df` *F.update.df - Update degrees of freedom in a Cormack-Jolly-Seber fitted object*

Description

Updates the degrees of freedom in a fitted object to either the value estimated from the rank of the variance-covariance matrix, the number of coefficients, or a user-specified value.

Usage

```
F.update.df(fit, df=NA)
```

Arguments

`fit`	An MRA fitted CJS model. Class must be c("cjs", "cr"). These are produced by `F.cjs.estim`.

df — The new value for degrees of freedom. If df == NA, the number of parameters estimated from the rank of the matrix of 2nd derivatives or Hessian, depending on cov.meth parameter. If df <= 0, the number of parameters will be set to NX+NY = the number of estimated coefficients. Otherwise, if df > 0, the supplied value is used. The penalty terms of AIC, QAIC, AICc, and QAICc are recomputed using this value.

Value

An object (list) of class c("cjs","cr") with degrees of freedom, AIC, QAIC, AICc, and QAICc updated.

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cjs.estim

Examples

```
## Fit CJS model to dipper data, time-varying capture and survivals.
data(dipper.histories)
ct <- as.factor( paste("T",1:ncol(dipper.histories), sep=""))
attr(ct,"nan")<-nrow(dipper.histories)
dipper.cjs <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(ct,drop=c(1,6,7)), dipper.histories )

## Update the degrees of freedom
dipper.cjs <- F.update.df( dipper.cjs, -1 )
```

---

ivar — *Expand Individual-varying covariates in models*

Description

Expands a vector of individual-varying values into a 2-d matrix of the appropriate size for use in MRA model formulas.

Usage

```
ivar(x, ns=attr(x,"ns"), drop.levels=attr(x,"drop.levels"))
```

Arguments

x	The vector of individual varying values to expand. This can be a factor (see as.factor). It is assumed that length(x) = number of individuals. If not, an error will occur in whatever routine called this function (e.g., F.3d.model.matrix).
ns	Number of sampling occasions. Default is to use the 'ns' attribute of x. If ns is not specified or is not an attribute of x, an error is thrown.
drop.levels	A vector of integers specifying which levels of a factor to drop. Only applicable if x is a factor. By default, the the 'drop.levels' attribute of x is used. If x does not have a 'drop.levels' attribute, the first level of the factor is dropped. drop.levels=length(levels(x)) does the SAS thing and drops the last level of a factor. Specifying multiple levels to drop is acceptable. E.g., drop.levels=c(1,2,7) drops the 1st, 2nd, and 7th levels of the factor, whatever they are. First level of a factor is first element of levels(x). Second level of a factor is second element of levels(x), and so on. Setting drop.levels to 0, a negative number, or a number greater than the number of levels will not drop any levels (this is so-called cell mean coding). Keep in mind presence of the intercept.

Value

A 2-d matrix of size length(x) x ns suitable for passing to the Fortran DLL of MRA for estimation. Values within rows are constant, values across rows vary according to x. If x is a factor, this matrix contains 0-1 indicator functions necessary to fit the factor.

If x is a factor, attributes of the returned matrix are "levels" = levels of the factor and "contr" = contrasts used in the coding (always contr.treatment). For other contrast coding of factors, make your own 2-d matrix with a call to the appropriate function (like contr.poly).

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cjs.estim, tvar

Examples

```
nan <- 30
ns <- 5
age <- as.factor(sample( c("J","S1","S2","Adult"), size=nan, replace=TRUE ))
attr(age,"ns") <- ns

# Note that levels get reordered (by R default, alphabetically)
attr(age,"drop.levels") <- (1:length(levels(age)))[ levels(age) == "J" ]

age.mat <- ivar(age)  # level J is the reference
age.mat <- ivar(age, drop=4) # level S2 is the reference

# Look at 3-D matrix produced when called with a factor.
dim(age.mat) <- c(nan,ns,length(levels(age))-1)
```

```
print(age.mat)  # each page is the 2-d matrix used in the fit.
print(age.mat[1,,])

age.mat <- ivar(age, drop=c(3,4))  # level S1 and S2 are combined and are the reference

# compare above to
ivar( c(1,1,2,2,3,3), 5 )
```

---

lines.cjs *lines.cjs*

Description

Add a line to an existing CJS capture-recapture plot showing either N or survival estimates.

Usage

```
## S3 method for class 'cjs'
lines( x, what="n", animals=-1, occasions=-1, ... )
```

Arguments

x	CJS object from F.cr.estim
what	Indicator for what to plot. what = "n" plots estimates of size (i.e,. \$n.hat). what = "s" plots estimates of survival.
animals	Index of animals to plot. This is the row number for animals to include. E.g., if animals = c(1,4,10), the 1st, 4th, and 10th animals represented in the 1st, 4th, and 10th rows of the capture history matrix are plotted. Applies to survival estimates only.
occasions	Sampling occasions to plot. This must match the occasions argument to the last plot.cjs. If the first element of occasions is <= 0, all occasions are plotted. Otherwise, only occasions listed in occasions are plotted.
...	Additional arguments to lines (for N estimates) or matlines (for survival estimates). Arguments like col= and lty= may prove useful.

Details

This is a utility function for plotting. Lines are added to the current plot. A current plot must be displayed.

Value

Nothing. A value of 1 is invisibly returned.

Author(s)

Trent McDonald, WEST Inc., tmcdonald@west-inc.com

See Also

plot.cjs, lines, matlines

Examples

```
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }

# Fit constant capture probability, period (i.e., flood) effects on survival
dipper.cjs <- F.cjs.estim( ~1, ~x2+x3, dipper.histories )

plot(dipper.cjs, type="s", animals=1)
lines(dipper.cjs, what="s", animals=c(4, 10))
```

---

mra.control *mra.control - Control over MRA fitting process*

Description

Auxiliary function providing a user interface for mra fitting. Typically only used when calling one of the *.estim routines.

Usage

```
mra.control(algorithm=1, maxfn=1000, cov.meth=1, trace=0, tol=1e-07 )
```

Arguments

algorithm	Integer specifying the maximization algorithm to use. If algorithm = 1, the VA09AD algorithm from the HSL library is used. The VA09AD algorithm is very reliable, has been tested extensively (same algorithm as Program MARK), and will almost always find the maximum likelihood estimates. This parameter was added to allow easy addition of other algorithms in the future, but no other options are currently implemented because VA09AD works so well. Anything other than 1 throws a warning and resets the value to 1.
maxfn	Maximum number of likelihood evaluations allowed during the fitting process. This determines when the minimization routine stops and concludes that the problem will not converge. The routine stops after the likelihood has been evaluated maxfn times within the minimization routine (but not the hessian or gradient routine). Decreasing this value will cause the fitting routine to give up on badly-behaved models quicker and return faster. In some simulation where ill-conditioned data can be generated, decreasing this parameter can speed up the simulation tremendously. Increasing this value may allow marginally-behaved (slow to converge) models to converge.

`cov.meth` Integer specifying the covariance estimation method. `cov.meth` = 1 takes numerical 2nd partial derivatives. `cov.meth` = 2 inverts the Hessian of the maximization. Method 1 (numeric 2nd derivatives) is the preferred method, but is computationally expensive and therefore slower than method 2. The difference in speed is minimal when number of parameters < 15. Method 2 variances are generally very close to method 1 variances, and could be used during initial model fitting stages when exact estimation of variance may not be necessary.

`trace` Integer controlling output of intermediate results. If `trace` != 0, a few lines will be written to the R console and a log file (named `mra.log`) will be opened in the current directory and details of the fitting process will be written there. The level of detail written to the log depends on the value of trace. If `trace` == 1, details written to the log include TEST2 and TEST3 results used to determine c-hat, details from the fitting algorithm, the variance covariance matrix, singular values of the VC matrix used to determine `df`, and other statistics. If `trace` == 2, all trace == 1 details plus the likelihood output is written to the log. If `trace` == 3, all trace == 2 details plus the gradient is written to the log.

When using `algorithm` = 1 (VA09AD), `trace` has additional meaning. If `trace` > 3, the coefficient and gradient vectors are written every |trace| iterations and also on exit. Output is of the form: Function value, beta(1), beta(2), ..., beta(n), G(1), ...G(n). Coefficient and gradient values are suppressed if `trace` < 0 (only final results printed). Intermediate values from within VA09AD are suppressed if `trace` > `maxit` + 1, but other intermediate values are output.

`trace` > 1 probably produces more output than the casual (non-geek statistician) wants to look at. But, geek statisticians may find `trace` > 3 useful for assessing stability and determining when the routine gets stuck in a local minima.

`tol` Vector or scalar of tolerance(s) for coefficients in the model. Minimization stops and concludes it has converged when |delta.b(i)| < tol(i) for all i, where delta.b(i) is change in parameter i on an iteration.

Value

A list with the input arguments as components.

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

`F.cjs.estim`, `F.huggins.estim`

Examples

```
data(dipper.histories)
ct <- as.factor( paste("T",1:ncol(dipper.histories), sep=""))
attr(ct,"nan")<-nrow(dipper.histories)
dipper.cjs <- F.cjs.estim( ~tvar(ct,drop=c(1,2)), ~tvar(ct,drop=c(1,6,7)),
  dipper.histories, control=mra.control(trace=1, maxfn=200) )
```

plot.cjs *Plot CJS Model*

Description

Plot the population size or survival estimates for a Cormack-Jolly-Seber model estimated by F.cjs.estim

Usage

```
## S3 method for class 'cjs'
plot(x, type="n", ci = TRUE, smooth = TRUE, occasions = -1,
 animals = -1, smubass = 5, ...)
```

Arguments

x	An object of class 'cjs'. Objects of this class are estimated open population Cormack-Jolly-Seber models produced by F.cjs.estim.
type	Type of estimates to plot. type = "n" (the default) produces a plot of population size estimates versus capture occasion. type = "s" produces a plot of survival estimates versus capture occasion.
ci	Plot confidence intervals? If ci=TRUE, confidence intervals around population size or survival estimates are plotted (depending on 'type='), otherwise, only confidence intervals are not plotted.
smooth	Smooth estimates of population size? If type="n", smooth=TRUE will produce a smoothed (supsmu) line through plotted size estimates. Ignored for type="s".
occasions	Vector of occasion numbers to use in plot. If any(occasions <= 0), all occasions are plotted. Otherwise, plot the occasions specified. For example, if occasions = c(1,3,5), only estimates from the 1st, 3rd, and 5th capture occasion are plotted. If type = "n", occasion = 1 (1st occasion) cannot be plotted because it can't be estimated. If type = "s", occasion = ncol(y) (last occasion) cannot be plotted because no survival interval exist beyond the end of the study.
animals	Vector of individuals to plot. This parameter is used only when plotting survival estimates. For example, animals = c(1,2,10) plots the survival estimates of the 1st, 2nd, and 10th animals (rows 1, 2, and 10 of the survival estimate matrix).
smubass	Bass parameter for super-smoothed line, if called for by smooth=TRUE. Must be between 0 and 10. Larger numbers produce smoother lines.
...	Additional arguments to plot (for size estimates) or matplot (for survival estimates). Options such as ylim=, col=, cex.axis=, etc. may be useful.

Details

Confidence intervals on survival estimates cannot be plotted with this routine. To plot confidence intervals surrounding survival estimates, call this routine with type="s", then compute confidence intervals for survival estimates, and use lines to add lines to the plot.

Value

For type="s", the survival estimate matrix that was plotted is is invisibly returned. For type = "n", the smooth fit is invisibly returned if called for by smooth = TRUE, otherwise NA is invisibly returned if smooth = FALSE.

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cjs.estim, matplot, lines, plot

Examples

```
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x2+x3+x4+x5+x6, ~x1+x2+x3+x4+x5, dipper.histories )
plot(dipper.cjs)
print(dipper.cjs$s.hat)
plot(dipper.cjs, type="s", animals=1)
```

---

plot.nhat *Plot size estimates*

Description

Plot the population size estimates for a Cormack-Jolly-Seber model estimated by F.cjs.estim

Usage

```
## S3 method for class 'nhat'
plot(x, ci = TRUE, smooth = TRUE, occasions = -1,
 smubass = 5, ...)
```

Arguments

x	An object of class 'nhat', which inherits from 'cjs'. Objects of this class are estimated open population Cormack-Jolly-Seber models produced by F.cjs.estim.
ci	Plot confidence intervals? If ci=TRUE, confidence intervals around population size or survival estimates are plotted (depending on 'type='), otherwise, only confidence intervals are not plotted.
smooth	Smooth estimates of population size? If type="n", smooth=TRUE will produce a smoothed (supsmu) line through plotted size estimates. Ignored for type="s".

occasions	Vector of occasion numbers to use in plot. If any(occasions <= 0), all occasions are plotted. Otherwise, plot the occasions specified. For example, if occasions = c(1,3,5), only estimates from the 1st, 3rd, and 5th capture occasion are plotted. If type = "n", occasion = 1 (1st occasion) cannot be plotted because it can't be estimated. If type = "s", occasion = ncol(y) (last occasion) cannot be plotted because no survival interval exist beyond the end of the study.
smubass	Bass parameter for super-smoothed line, if called for by smooth=TRUE. Must be between 0 and 10. Larger numbers produce smoother lines.
...	Additional arguments to plot (for size estimates) or matplot (for survival estimates). Options such as ylim=, col=, cex.axis=, etc. may be useful.

Value

The smooth fit is invisibly returned if called for by smooth = TRUE, otherwise NA is invisibly returned.

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cjs.estim, matplot, lines, plot, plot.cjs

Examples

```
data(dipper.histories)
dipper.cjs <- F.cjs.estim( ~1, ~1, dipper.histories )
plot(dipper.cjs,type="n")

# See examples for F.cr.model.avg for a way to plot model averaged population size estimates.
```

---

predict.cjs *predict.cjs*

Description

Predictor method for CJS capture-recapture objects. Return expected values for all active cells in the model.

Usage

```
## S3 method for class 'cjs'
predict(object, ...)
```

Arguments

object	CJS capture-recapture model as output from F.cjs.estim
...	Additional arguments to other functions. Not used, but must be here for compatibility with the generic predict function.

Details

The only components of cjsobj needed are $histories, $p.hat, $s.hat

Value

A nan X ns matrix of fitted values (=cell expected value), where nan=number of animals and ns=number of samples. Fitted values in the non-active cells are set to NA. Non-active cells are those prior to and including the initial capture, and after the occasion on which an animal is known to have died. Computation of expected values is described in the details section of the help file for F.cjs.gof.

Author(s)

Trent McDonald, WEST Inc., tmcdonald@west-inc.com

See Also

F.cjs.estim, F.cjs.gof

Examples

```
# Fit CJS model to dipper data, time-varying capture and survivals.
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x2+x3+x4+x5+x6, ~x1+x2+x3+x4+x5, dipper.histories )
dipper.expected <- predict(dipper.cjs)
```

---

print.cjs *Print Cormack-Jolly-Seber (CJS) Models*

Description

Print method for Cormack-Jolly-Seber (CJS) models estimated by F.cjs.estim().

Usage

```
## S3 method for class 'cjs'
print(x, alpha=c(0.05,0.01), ...)
```

Arguments

x	An object of class "cjs" produced by F.cjs.estim()
alpha	A vector with length either 2 or 3 containing alpha levels used to put "*" or "**" beside the GOF results. One * is printed if significance is between alpha[1] and alpha[2] (i.e., if alpha[2] < p < alpha[1]). Two ** are printed if significance is less than alpha[2] (p < alpha[2]).
...	Arguments to other functions called by this one. Currently no other functions are called, so this is not used, but must be here for compatibility with the generic `print` function.

Value

Nothing is returned. This function is used exclusively for its side effects. It prints an object of class "cjs" in a nice human-readable format. If goodness-of-fit tests are present, they are printed. If population size estimates are present, they are printed.

Author(s)

Trent McDonald, Ph.D., WEST-INC, tmcdonald@west-inc.com

See Also

`F.cjs.estim`, `plot.cjs`

Examples

```
# Fit CJS model to dipper data, time-varying capture and survivals.
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x2+x3+x4+x5+x6, ~x1+x2+x3+x4+x5, dipper.histories )
print(dipper.cjs)
```

---

print.hug *Print Huggin's Model objects*

Description

Print method for Huggin's closed population model objects estimated by F.huggins.estim().

Usage

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

Arguments

x	An object of class "hug" produced by F.huggins.estim()
...	Arguments to other functions called by this one. Currently no other functions are called, so this is not used, but must be here for compatibility with the generic `print` function.

Details

If there are no free covariates in the recapture model (i.e., `recapture=NULL`), only the combined capture and recapture model is printed. If the recapture model has coefficients, coefficients in both are printed in separate columns.

When a recapture model has free coefficients, a "C" (for "capture") appears next to coefficients in the recapture model that also appear in the capture model. These coefficients are fixed in the recapture model because they are not free. These values were estimated from initial capture information. A "B" (for "behavioral") appears next to free coefficients in the recapture model that *do not* appear in the initial capture model.

Value

Nothing is returned. This function is used exclusively for its side effects. It prints an object of class "hug" in a nice human-readable format.

Author(s)

Trent McDonald, Ph.D., WEST-INC, tmcdonald@west-inc.com

See Also

`F.huggins.estim`

Examples

```
data(dipper.histories)
dipper.fit <- F.huggins.estim( ~1, histories= dipper.histories )
print(dipper.fit)
```

---

print.nhat *print.nhat*

Description

Print the estimates of N from a CJS object in a nice format.

Usage

```
## S3 method for class 'nhat'
print(x, width=10, ...)
```

Arguments

x	An object of class "nhat", which inherits from class "cr". This class of object is output from F.cr.model.avg.
width	The minimum width of columns in the table printed by this routine.
...	Arguments to other functions called by this one. Currently no other functions are called, so this is not used, but must be here for compatibility with the generic print function.

Details

Horvitz-Thompson estimates of N, along with standard errors, are printed. print.cjs also prints N estimates, if present, but as a brief, one-row summary. This routine prints a more complete table. Numerical values of the supsmu smooth of N estimates (bass = 0.5) associated with plots produced by plot.cjs are printed.

Value

Nothing. Run for side effects

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

plot.cjs, print.cjs, F.cjs.estim

Examples

```
# Fit CJS model to dipper data, time-varying capture and survivals.
data(dipper.histories)

# Method 1: explicit matricies
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x2+x3+x4+x5+x6, ~x1+x2+x3+x4+x5, dipper.histories )
print(dipper.cjs)

# Method 2: indicator vectors
x <- factor(1:ncol(dipper.histories), labels=paste("t",1:ncol(dipper.histories),sep=""))
nd <- nrow(dipper.histories)
dipper.cjs <- F.cjs.estim( ~tvar(x,nan=nd,drop=c(1,7)), ~tvar(x,nan=nd,drop=c(6,7)),
    dipper.histories)
print(dipper.cjs)
```

`residuals.cjs` *Residuals for CJS Model*

Description

Residual extraction routine for a CJS object. Returns Pearson or deviance residuals of a CJS capture-recapture model.

Usage

```
## S3 method for class 'cjs'
residuals(object, type="pearson", ...)
```

Arguments

`object`	a CJS (Cormack-Jolly-Seber capture-recapture) object, which is usually the result of calling F.cjs.estim
`type`	string indicating type of residual to return. Either "pearson" for Pearson residuals (i.e., (o - e)/sqrt(e*(1-e))) or "deviance" for deviance residuals (i.e., 2*sign(o-e)*sqrt(o*log(o/e) + (1-o)*log((1-o)/(1-e))))
`...`	Additional arguments to other functions. Not used, but must be here for compatibility with the generic `residuals` function.

Details

In almost all cases, a CJS model fitted by F.cjs.estim already has a `$residuals` component. This routine either extracts this component, or computes residuals of the component if not found.

Observed component (o(ij)) in formulas above is the capture indicator for animal i during occasion j. If animal i was seen during occasion j, o(ij) = 1. Otherwise, o(ij) = 0.

Expected component (e(ij)) in formula above is the expected value of the capture indicator for animal i during occasion j. In other words, o(ij) is a binomial random variable with expected value e(ij). Under the assumptions of a CJS model, e(ij) is computed as phi(i(1)) * phi(i(2)) * ... * phi(i(j-1)) * p(ij), where p(ij) is the estimated capture probability of animal i during occasion j, and phi(i(1)) is estimated survival during the first interval following initial capture of the animal, phi(i(2)) is survival during the second interval after initial capture, and phi(i(j-1)) is survival during the interval just prior to occasion j.

Value

A NAN X NS matrix of residuals, where NAN = number of animals and NS = number of capture occasions. Residuals in the non-active cells are set to NA. Non-active cells are those prior to and including the initial capture, and after the occasion on which an animal is known to have died.

If type = "pearson", the residual for active cell (i,j) is (o(ij) - e(ij)) / sqrt(e(ij) * (1 - e(ij))).

If type = "deviance", the residual for active cell (i,j) is 2 * sign(o(ij) - e(ij)) * sqrt(o(ij)*log(o(ij) / e(ij)) + (1 - o(ij)) * log((1 - o(ij)) / (1 - e(ij)))).

Observed (o(ij)) and expected (e(ij)) are defined in Details.

Author(s)

Trent McDonald

See Also

F.cjs.estim, predict.cjs

Examples

```
# Fit CJS model to dipper data, time-varying capture and survivals.
data(dipper.histories)
xy <- F.cjs.covars( nrow(dipper.histories), ncol(dipper.histories) )
for(j in 1:ncol(dipper.histories)){ assign(paste("x",j,sep=""), xy$x[,,j]) }
dipper.cjs <- F.cjs.estim( ~x2+x3+x4+x5+x6, ~x1+x2+x3+x4+x5, dipper.histories )
residuals(dipper.cjs)
```

---

tvar *Expand Time-varying covariates in models*

Description

Expands a vector of time-varying values into a 2-d matrix for use in MRA model formulas.

Usage

```
tvar(x, nan=attr(x,"nan"), drop.levels=attr(x,"drop.levels"))
```

Arguments

x	The vector of time varying values to expand. This can be a factor (see as.factor). It is assumed that length(x) = number of sampling occasions. If not, an error will occur in whatever routine called this function (e.g., F.3d.model.matrix).
nan	Number of individuals. Default is to use the 'nan' attribute of x. If nan is not specified or is not an attribute of x, an error is thrown.
drop.levels	A vector of integers specifying which levels of a factor to drop. Only applicable if x is a factor. By default, the the 'drop.levels' attribute of x is used. If x does not have a 'drop.levels' attribute, the first level of the factor is dropped. drop.levels=length(levels(x)) does the SAS thing and drops the last level of a factor. Specifying multiple levels to drop is acceptable. E.g., drop.levels=c(1,2,7) drops the 1st, 2nd, and 7th levels of the factor. First level of a factor is first element of levels(x). Second level of a factor is second element of levels(x), and so on. Setting drop.levels to 0, a negative number, or a number greater than the number of levels will not drop any levels (this is so-called cell mean coding). Keep in mind presence of the intercept.

Value

A 2-d matrix of size nan x length(x) suitable for passing to the Fortran DLL of MRA for estimation. Values within columns are constant, values across columns vary according to x. If x is a factor, this matrix contains 0-1 indicator functions necessary to fit the factor.

If x is a factor, attributes of the returned matrix are "levels" = levels of the factor and "contr" = contrasts used in the coding (always contr.treatment). For other contrast coding of factors, make your own 2-d matrix with a call to the appropriate function (like contr.poly).

Author(s)

Trent McDonald, WEST-INC, tmcdonald@west-inc.com

See Also

F.cjs.estim, tvar

Examples

```
nan <- 30
ns <- 5
time.occ <- as.factor(paste("T",1:ns, sep=""))
attr(time.occ,"nan") <- nan
attr(time.occ,"drop.levels") <- ns

time.mat <- tvar(time.occ) # Last occasion is the reference, the SAS and MARK default.

time.mat <- tvar(as.factor(1:ns),nan,ns) #equivalent to above.

#   Look at 3-d matrix produced when called with factors
dim(time.mat) <- c(nan,ns,length(levels(time.occ))-1)
print(time.mat)  # each page is the 2-d matrix used in the fit.
print(time.mat[1,,])

#  compare above to
tvar( 1:ns, nan )
```

Index

∗ **datasets**
dipper.data, 4
∗ **models**
F.3d.model.matrix, 5
F.cjs.covars, 6
F.cjs.estim, 7
F.cjs.gof, 15
F.cjs.simulate, 18
F.cr.model.avg, 22
F.cr.model.matrix, 25
F.fit.table, 27
F.huggins.estim, 29
F.sat.lik, 35
F.step.cjs, 36
F.update.df, 39
ivar, 40
lines.cjs, 42
mra.control, 43
plot.cjs, 45
plot.nhat, 46
predict.cjs, 47
print.cjs, 48
print.hug, 49
print.nhat, 50
residuals.cjs, 52
tvar, 53
∗ **package**
mra-package, 2
∗ **survival**
F.cjs.estim, 7
F.cjs.simulate, 18
F.cr.model.avg, 22
F.fit.table, 27
F.huggins.estim, 29
F.step.cjs, 36
F.update.df, 39
residuals.cjs, 52

CJS *(*F.cjs.estim*)*, 7
cjs *(*F.cjs.estim*)*, 7
Cormack *(*F.cjs.estim*)*, 7
cr.estim *(*F.cjs.estim*)*, 7

dipper.data, 4
dipper.histories *(*dipper.data*)*, 4
dipper.males *(*dipper.data*)*, 4

eval, *6*, *26*

F.3d.model.matrix, 5
F.cjs.covars, 6, *14*
F.cjs.estim, *7*, 7, *17*, *20*, *24*, *26*, *28*, *31*, *34*, *36*, *38*, *40, 41*, *44*, *46–49*, *51*, *53, 54*
F.cjs.gof, *14*, 15, *48*
F.cjs.simulate, 18
F.cr.model.avg, 22
F.cr.model.matrix, *6*, 25
F.fit.table, *24*, 27, *38*
F.huggins.estim, *24*, *28*, 29, *44*, *50*
F.sat.lik, 35
F.step.cjs, 36
F.update.df, *14*, 39

Huggins *(*F.huggins.estim*)*, 29
huggins *(*F.huggins.estim*)*, 29

ivar, *6*, *14*, 40

Jolly *(*F.cjs.estim*)*, 7

lines, *43*, *46, 47*
lines.cjs, 42

matlines, *43*
matplot, *46, 47*
model.matrix, *6*, *26*
mra *(*mra-package*)*, 2
mra-package, 2
mra.control, *14*, *32*, 43

plot, *46, 47*

plot.cjs, *14*, *24*, *43*, 45, *47*, *49*, *51*
plot.nhat, 46
predict.cjs, 47, *53*
print.cjs, *14*, *17*, 48, *51*
print.hug, *34*, 49
print.nhat, 50

residuals.cjs, *14*, 52

Seber *(*F.cjs.estim*)*, 7

tvar, *6*, *14*, *41*, 53, *54*