

Package: markedTMB (via r-universe)

November 4, 2024

Type Package

Version 0.3.0

Date 2021-11-29

Title Mark-Recapture Analysis for Survival Estimation

Author Jeff Laake <jefflaake@gmail.com>

Maintainer Jeff Laake <jefflaake@gmail.com>

Description Functions for fitting various models to capture-recapture data including mixed-effects multistate models, multi-variate state models and multistate live and dead recovery structure for survival estimation. All models are implemented with TMB.

Depends R (>= 3.5.0), lme4, methods, parallel, TMB

Imports graphics, stats, utils, truncnorm, coda, Matrix, numDeriv, expm, optimx (>= 2013.8.6), data.table, knitr, kableExtra, bookdown, mvtnorm, TMB (>= 1.7.22)

Suggests ggplot2

LazyLoad yes

VignetteBuilder knitr

License GPL (>=2)

RoxygenNote 7.1.2

Encoding UTF-8

LazyData true

LinkingTo TMB

Repository <https://dsjohnson.r-universe.dev>

RemoteUrl <https://github.com/jlaake/markedTMB>

RemoteRef master

RemoteSha 519dd14104380885922b765f96dfa32b9f685963

Contents

backward_prob	3
boot_pred_states	4
cjs.accumulate	4
cjs_delta	5
coef.crm	6
compute_matrices	7
compute_real	7
convert.link.to.real	9
create.dm	10
create.dmdf	11
create.fixed.matrix	15
create.links	15
crm	16
crm.wrapper	20
deriv_inverse.link	21
dipper	22
dmat_hsmm2hmm	23
fix.parameters	24
function.wrapper	25
global_decode	25
hsmm2hmm	26
initiate_pi	27
inverse.link	28
load_lib	29
local_decode	29
make.design.data	30
merge_design.covariates	31
mixed.model.admb	32
mscjs_tmb	33
msjsu_tmb	37
msjs_tmb	39
msld_tmb	41
mstrata	43
mvmscjs_tmb	44
mvms_design_data	53
omega	54
Paradise_shelduck	54
Phi.mean	56
predict.crm	57
print.crm	58
print.crmlist	59
proc.form	59
process.ch	60
process.data	61
resight.matrix	64
sealions	65

backward_prob 3

set.fixed	66
set.initial	67
set.scale	67
setup.model	68
setup.parameters	69
setup_tmb	71
set_mvms	71
skagit	72
smsld_tmb	73
splitCH	75
valid.parameters	76
Values	77

Index 78

<i>backward_prob</i>	<i>Computes backward probabilities</i>
----------------------	--

Description

Computes backward probability sequence for a set of capture histories

Usage

`backward_prob(object, state.names = NULL, par = NULL)`

Arguments

<code>object</code>	fitted crm model
<code>state.names</code>	if not NULL use these labels instead of strata.labels in model
<code>par</code>	if not NULL use these values for matrix calculations

Value

array of backward probabilities (one for each id, state, occasion)

Author(s)

Jeff Laake

References

Zucchini, W. and I.L. MacDonald. 2009. Hidden Markov Models for Time Series: An Introduction using R. Chapman and Hall, Boca Raton, FL. See page 61.

boot_pred_states	<i>Parametric bootstrap of state predictions</i>
------------------	--

Description

Bootstraps the state predictions for a model by assuming a multivariate normal distribution for the model parameter estimates. For each bootstrap replicate it replaces model parameter estimates with a random sample from a multivariate normal where the mean vector is the fitted model parameter estimates and the var-cov matrix is the estimated matrix from the model fit.

Usage

```
boot_pred_states(model, nboot = 0, beta_only = FALSE, beta_reps = NULL)
```

Arguments

model	fitted model object
nboot	number of bootstrap replicates; if 0 it returns the model state predictions with global_decode at the estimated parameter values
beta_only	if TRUE, generates beta_reps from model and returns it. This is for cases where the same values of beta_reps are used for another bootstrap (eg projections using survival)
beta_reps	if NULL the values are generated; otherwise it is a matrix of values with rows being the reps

Value

If nboot=0, a matrix of model state predictions where rows are the individual histories and columns are the occasions. If nboot>0, a list of length nboot with state prediction matrices is returned.

Author(s)

Jeff Laake

cjs.accumulate	<i>Accumulates common capture history values</i>
----------------	--

Description

To speed up computation, animals with the same capture history and design matrix are accumulated and represented by a frequency. Computes starting values for Phi and p parameters from the list of design matrices and the summarized data list including ch matrix and first and last vectors. If any values are missing (NA) or abs(par)>5, they are set to 0.

Usage

```
cjs.accumulate(x, model_data, nocc, freq, chunk_size)
```

Arguments

x	data
model_data	list of design matrices, fixed parameters and time intervals all which can vary by animal
nocc	number of capture occasions
freq	frequency of each capture history before accumulation
chunk_size	size that determines number of pieces of data/design matrix that are handled. Smaller chunk_size requires more time but less memory. 1e7 is default set in cjs.

Value

modified model_data list that is accumulated

Author(s)

Jeff Laake

cjs_delta

HMM Initial state distribution functions

Description

Functions that compute the initial probability distribution for the states. Currently only CJS, MS models and MS models with state uncertainty are included and these all use cjs_delta to assign a known state.

Usage

```
cjs_delta(pars, m, F, T, start)
```

Arguments

pars	list of real parameter matrices (id by occasion) for each type of parameter
m	number of states
F	initial occasion vector
T	number of occasions
start	matrix with values that are first occasion and for some CJS type models the state of first observation

Value

2-d array of initial state probability vectors for each id

Author(s)

Jeff Laake

References

Zucchini, W. and I.L. MacDonald. 2009. Hidden Markov Models for Time Series: An Introduction using R. Chapman and Hall, Boca Raton, FL. 275p.

coef.crm

Extract coefficients

Description

Extracts the beta coefficients from the model results.

Usage

```
## S3 method for class 'crm'  
coef(object, ...)
```

Arguments

object	crm model result
...	generic arguments not used here

Value

returns a dataframe with estimates and standard errors and confidence intervals if hessian=TRUE on model run.

Author(s)

Jeff Laake

See Also

[crm](#)

compute_matrices	<i>Compute HMM matrices</i>
------------------	-----------------------------

Description

Computes gamma, dmat and delta (initial) matrices(arrays) and returns them in a list.

Usage

```
compute_matrices(object, state.names = NULL, par = NULL)
```

Arguments

object	fitted crm model
state.names	if not NULL use these labels instead of strata.labels in model
par	if not NULL use these values for matrix calculations

Value

list with gamma, dmat and delta arrays

Author(s)

Jeff Laake

compute_real	<i>Compute estimates of real parameters</i>
--------------	---

Description

Computes real estimates and their var-cov for a particular parameter.

Usage

```
compute_real(  
  model,  
  parameter = NULL,  
  ddl = NULL,  
  unique = TRUE,  
  vcv = FALSE,  
  se = FALSE,  
  chat = 1,  
  subset = NULL,  
  select = NULL,  
  showDesign = FALSE,
```

```

include = NULL,
uselink = FALSE,
merge = FALSE,
unit_scale = TRUE,
simplify = FALSE,
show.fixed = FALSE
)

```

Arguments

model	model object
parameter	name of real parameter to be computed (eg "Phi" or "p")
ddl	list of design data
unique	TRUE if only unique values should be returned unless non-NULL subset is specified
vcv	logical; if TRUE, computes and returns v-c matrix of real estimates
se	logical; if TRUE, computes std errors and confidence intervals of real estimates
chat	over-dispersion value
subset	logical expression using fields in real dataframe; if used gives all estimates which ignores unique=TRUE
select	character vector of field names in real that you want to include
showDesign	if TRUE, show design matrix instead of data
include	vector of field names always to be included even when select or unique specified
uselink	default FALSE; if TRUE uses link values in evaluating uniqueness
merge	default FALSE but if TRUE, the ddl for the parameter is merged (cbind) to the estimates but only if unique=FALSE
unit_scale	default TRUE, if FALSE any time scaled parameter (e.g. Phi,S) is scaled when computing real value such that it represents the length of the interval rather than a unit interval
simplify	if TRUE, simplifies ddl to unique values prior to calculation; not valid with random effects
show.fixed	if TRUE, returns fixed parameter values as well as estimated values

Details

This code is complicated primarily by the mlogit parameters which are computed in 2 stages: 1) log link and 2) summation to normalize. The mlogit is handled differently depending on the model. For MS and JS models, one of the parameters is computed by subtraction (specified as `addone==TRUE`) whereas the HMM models (`addone=FALSE`) specify a parameter for each cell and one is fixed by the user to 1. The latter is preferable because it then provides an estimate and a std error for each parameter whereas the subtracted value is not provided for MS and JS.

This function differs from `compute.real` in RMark because it only computes the values for a single parameter whereas the function with the same name in RMark can compute estimates from multiple parameters (eg Phi and p).

Value

A data frame (real) is returned if vcv=FALSE; otherwise, a list is returned also containing vcv.real:

real	data frame containing estimates, and if vcv=TRUE it also contains standard errors and confidence intervals
vcv.real	variance-covariance matrix of real estimates

Author(s)

Jeff Laake

convert.link.to.real *Convert link values to real parameters*

Description

Computes real parameters from link values

Usage

```
convert.link.to.real(x, model = NULL, links = NULL, fixed = NULL)
```

Arguments

x	Link values to be converted to real parameters
model	model object
links	vector of character strings specifying links to use in computation of reals
fixed	vector of fixed values for real parameters that are needed for calculation of reals from mlogits when some are fixed

Details

Computation of the real parameter from the link value is relatively straightforward for most links and the function [inverse.link](#) is used. The only exception is parameters that use the mlogit link which requires the transformation across sets of parameters. This is a convenience function that does the necessary work to convert from link to real for any set of parameters. The appropriate links are obtained from model\$links unless the argument links is specified and they will over-ride those in model.

Value

vector of real parameter values

Author(s)

Jeff Laake

See Also

[inverse.link](#), [compute_real](#)

create.dm

Creates a design matrix for a parameter

Description

Creates a design matrix using the design dataframe, a formula and any intervals defined for time, cohort and age.

Usage

```
create.dm(x, formula, time.bins=NULL, cohort.bins=NULL, age.bins=NULL,
          chunk_size=1e7, remove.intercept=NULL, remove.unused.columns=TRUE)
```

```
create.dml(ddl, model.parameters, design.parameters, restrict=FALSE,
           chunk_size=1e7, remove.unused.columns=TRUE, simplify=FALSE)
```

Arguments

x	design dataframe created by create.dmdf
formula	formula for model in R format
time.bins	any bins of time to collapse values
cohort.bins	any bins of cohort to collapse values
age.bins	any bins of cohort to collapse values
chunk_size	specifies amount of memory to use in creating design matrices; amount used is 8*chunk_size/1e6 MB (default 80MB)
remove.intercept	if TRUE, forces removal of intercept in design matrix
remove.unused.columns	if TRUE, unused columns are removed; otherwise they are left
ddl	Design data list which contains a list element for each parameter type; if NULL it is created
design.parameters	Specification of any grouping variables for design data for each parameter
model.parameters	List of model parameter specifications
restrict	if TRUE, only use design data with Time >= Cohort
simplify	if TRUE simplifies real parameter structure for some models; at this time it is not more efficient so ignore

Value

create.dm returns a fixed effect design matrix constructed with the design dataframe and the formula for a single parametre. It excludes any columns that are all 0. create.dml returns a list with an element for for each parameter with a sub-list for the fixed effect (fe) and random effects. re.list is a list with an element for each random component containing re.dm and indices for that random effect (eg (1lid) + (1time) would produce elements for id and time.

Author(s)

Jeff Laake

create.dmdf	<i>Creates a dataframe with all the design data for a particular parameter in a crm model</i>
-------------	---

Description

Creates a dataframe with all the design data for a particular parameter in a crm model which currently includes "cjs" or "js". These design data are fundamentally different than the design data created for mark models as explained below.

Usage

```
create.dmdf(x, parameter, time.varying = NULL, fields = NULL)
```

Arguments

x	processed dataframe from function process.data
parameter	list with fields defining each values for each parameter; as created by setup.parameters
time.varying	vector of field names that are time-varying for this parameter
fields	character vector containing field names for variables in x to be included in design matrix dataframe; if NULL all other than ch are included

Details

This function is intended to be called from [make.design.data](#). It takes the data in x and creates a dataframe with all of the data needed to fit capture-recapture models (crm) which currently includes "cjs" (Cormack-Jolly-Seber) or "js" (POPAN formulation of the Jolly-Seber model). Before jumping into the details it is useful to have an understanding of the differences between MARK (via the mark in RMark function) and the package mra written by Trent McDonald. With MARK, animals can be placed in groups and the parameters for the model specified via PIMs (parameter index matrices) link the parameters to the specific animals. For example, if for a particular group the Phi PIM is

```
1 2 3
  4 5
    6
```

Then animals in that group that were first caught/released on occasion 1 have the parameters 1,2,3 for the 3 occasions. Those first caught/released on occasion 2 have parameters 4 and 5 for occasions 2 and 3 and those first caught/released on occasion 3 have parameter 6 for occasion 3. Another group of animals would have a different set of indices for the same set of parameters so they could be discriminated. Many people find this rather confusing until they get used to it but even then if you have many different groups and many occasions, then the indexing process is prone to error. Thus, the rationale for RMark which automates the PIM construction and its use is largely transparent to the user. What RMark does is to create a data structure called design data that automatically assigns design data to the indices in the PIM. For example, 1 to 6 would be given the data used to create that group and 1 to 3 would be assigned to cohort 1, ... and 1 would be assigned to occasion 1 and 2 and 4 would be assigned to occasion 2 etc. It also creates an age field which follows the diagonals and can be initialized with the initial age at the time of first capture which is group specific. With a formula and these design data a design matrix can be constructed for the model where the row in the design matrix is the parameter index (e.g., the first 6 rows would be for parameters 1 to 6 as shown above). That would be all good except for individual covariates which are not group-specific. MARK handles individual covariates by specifying the covariate name (eg "weight") as a string in the design matrix. Then for each capture history it plugs in the actual covariate values for that animal to complete the design matrix for that animal. For more details see Laake and Rexstad (2008).

From a brief look at package *mra* and personal communication with Trent McDonald, I give the following brief and possibly incorrect description of the package *mra* at the time of writing (28 Aug 2008). In that package, the whole concept of PIMS is abandoned and instead covariates are constructed for each occasion for each animal. Thus, each animal is effectively put in its own group and it has a parameter for each occasion. This novel approach is quite effective at blurring the lines between design data and individual covariate data and it removes the needs for PIMS because each animal (or unique capture history) has a real parameter for each occasion. The downside of the package *mra* is that every covariate is assumed to be time-varying and any factor variables like time are coded manually as dummy variables for each level rather than using the R facilities for handling factor variables in the formula to create the design matrix.

In the *crm* functions in this package I have used basic idea in *mra* but I have taken a different approach to model development that allows for time-varying covariates but does not restrict each covariate to be time-varying and factor variables are used as such which removes the need to construct dummy variables; although the latter could still be used. First an example is in order to explain how this all works. Consider the following set of capture histories for small capture-recapture data set with 4 capture occasions:

```
1001 0111 0011
```

To relate the current structure to the concept of PIMS I define the following matrix

```
1 2 3
4 5 6
7 8 9
```

If you think of these as Φ parameter indices, then 1 to 3 are survivals for the intervals 1-2,2-3,3-4 for animal 1, and 4-6 and 7-9 are the same for animals 2 and 3. This matrix would have a row for each animal. Now you'll notice that for animal 2 parameter 4 is not needed and for animal 3,

parameters 7 and 8 are not needed because they are prior to their entry in the study. While that is certainly true there is no harm in having them and the advantage comes in being able to have a complete matrix in R rather than having a triangular one.

So now we are finally getting to the description of what this function does. It constructs a dataframe with a row for each animal-occasion. Following on with the example above, depending on how the arguments are set the following dataframe could be constructed:

row	time	Time	cohort	Cohort	age	Age	initial.age
1	1	0	1	0	0	0	0
2	2	1	1	0	1	1	0
3	3	2	1	0	2	2	0
4	1	0	2	1	0	0	0
5	2	1	2	1	1	1	0
6	3	2	2	1	2	2	0
7	1	0	3	2	0	0	0
8	2	1	3	2	1	1	0
9	3	2	3	2	2	2	0

The fields starting with a lowercase character (time,cohort,age) are created as factor variables and those with an uppercase are created as numeric variables. Note: the age field is bounded below by the minimum initial.age to avoid creating factor levels with non-existent data for occasions prior to first capture that are not used. For example, an animal first caught on occasion 2 with an initial.age=0 is technically -1 on occasion 1 with a time.interval of 1. However, that parameter would never be used in the model and we don't want a factor level of -1.

A formula of ~time would create a design matrix with 3 columns (one for each factor level) and ~Time would create one with 2 columns with the first being an intercept and the second with the numeric value of Time.

Now here is the simplicity of it. The following few expressions in R will convert this dataframe into a matrix of real parameters (assuming beta=c(1,1,1) that are structured like the square PIM matrix without the use of PIMs.

```
nocc=4
x=data.frame(ch=c("1001","0111","0011"),stringsAsFactors=FALSE)
beta=c(1,1,1)
x.proc=process.data(x,model="cjs")
Phi.dmdf=make.design.data(x.proc)$Phi
Phi.dm=create.dm(Phi.dmdf,~time)
Phimat=matrix(plogis(Phi.dm
```

Note that the order of the columns for Phi.dmdf differs slightly from what is shown above. Also, plogis is an R function that computes the inverse-logit. Once you have the matrix of Phi and p values the calculation of the likelihood is straightforward using the formulation of Pledger et al. (2003). The values in the design dataframe are not limited to these fields. The 2 arguments time.varying and fields are vectors of character strings which specify the names of the dataframe columns in x that should be included. For example if x contained a field sex with the values "M","F","M" for the 3 records in our example, and the argument fields=c("sex") was used then a column named sex would be included in design dataframe with the values "M","M","M","F","F","F","M","M","M".

The value of the column `sex` in `x` is repeated for each of the occasions for that animal (capture history). Now if the value of the field changes for each occasion then we use the argument `time.varying` instead. To differentiate the values in the dataframe `x` the columns are named with an occasion number. For example, if the variable was named `cov` and it was to be used for `Phi`, then the variables would be named `cov1`, `cov2`, `cov3` in `x`. Let's say that `x` was structured as follows:

```
ch  cov1 cov2 cov3
1001  1  0  1
0111  0  2  1
0011  0  0  0
```

If you specified the argument `time.varying=c("cov")` then in the design dataframe a field named `cov` would be created and the values would be `1,0,1,0,2,1,0,0,0`. Thus the value is both animal and occasion specific. Had the covariate been used for `p` then they would be named `cov2`, `cov3`, `cov4` because the covariate is for those occasions for `p` whereas for `Phi` the covariate is labelled with the occasion that begins the interval. Any number of fields can be specified in `fields` and `time.varying` that are specified in `x`.

The input dataframe `x` has a few minor requirements on its structure. First, it must contain a field called `ch` which contains the capture-history as a string. Note that in general strings are converted to factor variables by default when they are put into a dataframe but as shown above that can be controlled by the argument `stringsAsFactors=FALSE`. The capture history should be composed only of 0 or 1 and they should all be the same length (at present no error checking on this). Although it is not necessary, the dataframe can contain a field named `freq` which specifies the frequency of that capture history. If the value of `freq` is negative then these are treated as loss on capture at the final capture occasion (last 1). If `freq` is missing then a value of 1 is assumed for all records. Another optional field is `initial.age` which specifies the age of the animal at the time it was first captured. This field is used to construct the `age` and `Age` fields in the design dataframe. The default is to assume `initial.age=0` which means the age is really time since first marked. Any other fields in `x` are user-specified and can be a combination of factor and numeric variables that are either time-invariant or time-varying (and named appropriately).

The behavior of `create.dmdf` can vary depending on the values of `begin.time` and `time.intervals`. An explanation of these values and how they can be used is given in [process.data](#).

Value

A dataframe with all of the individual covariate data and the standard design data of `time`, `Time`, `cohort`, `Cohort`, `age` and `Age`; where lowercase first letter implies a factor and uppercase is a numeric variable for those variables.

Author(s)

Jeff Laake

References

- Laake, J. and E. Rexstad (2007). RMark – an alternative approach to building linear models in MARK. Program MARK: A Gentle Introduction. E. Cooch and G. C. White.
- Pledger, S., K. H. Pollock, et al. (2003). Open capture-recapture models with heterogeneity: I. Cormack-Jolly-Seber model. *Biometrics* 59(4):786-794.

create.fixed.matrix *Create parameters with fixed matrix*

Description

Creates fixed matrix in parameters from ddl\$fix values

Usage

```
create.fixed.matrix(ddl, parameters)
```

Arguments

ddl	design data list
parameters	parameter specification list

Value

parameters with fixed matrix set

Author(s)

Jeff Laake

create.links *Creates a 0/1 vector for real parameters with sin link*

Description

For each row in a given design matrix it assigns a value 1, if the columns used in the design matrix are only used as an identity matrix (only one 1 and remaining columns all 0).

Usage

```
create.links(dm)
```

Arguments

dm	design matrix
----	---------------

Value

A vector of length=nrow(dm) with value of 0 except for rows that can accommodate a sin link (identity design matrix). This function is not currently used because it has not been thoroughly tested.

Author(s)

Jeff Laake

`crm`*Capture-recapture model fitting function*

Description

Fits user specified models to some types of capture-recapture wholly in R and not with MARK. A single function that processes data, creates the design data, makes the crm model and runs it

Usage

```
crm(  
  data,  
  ddl = NULL,  
  begin.time = 1,  
  model = "MSCJS",  
  title = "",  
  model.parameters = list(),  
  design.parameters = list(),  
  initial = NULL,  
  groups = NULL,  
  time.intervals = NULL,  
  debug = FALSE,  
  method = "nlminb",  
  hessian = FALSE,  
  accumulate = TRUE,  
  chunk_size = 1e+07,  
  control = list(),  
  refit = 1,  
  itnmax = 5000,  
  scale = NULL,  
  run = TRUE,  
  compile = FALSE,  
  extra.args = NULL,  
  strata.labels = NULL,  
  clean = FALSE,  
  save.matrices = FALSE,  
  savef = TRUE,  
  getreals = FALSE,  
  real.ids = NULL,  
  check = FALSE,  
  useHess = FALSE,  
  optimize = TRUE,  
  vcv = FALSE,
```



```

    unit_scale = TRUE,
    ...
)

```

Arguments

<code>data</code>	Either the raw data which is a dataframe with at least one column named <code>ch</code> (a character field containing the capture history) or a processed dataframe
<code>ddl</code>	Design data list which contains a list element for each parameter type; if <code>NULL</code> it is created
<code>begin.time</code>	Time of first capture(release) occasion
<code>model</code>	Type of c-r model (eg, "cjs", "js")
<code>title</code>	Optional title; not used at present
<code>model.parameters</code>	List of model parameter specifications
<code>design.parameters</code>	Specification of any grouping variables for design data for each parameter
<code>initial</code>	Optional vector of initial values for beta parameters; if named from previous analysis only relevant values are used
<code>groups</code>	Vector of names factor variables for creating groups
<code>time.intervals</code>	Intervals of time between the capture occasions
<code>debug</code>	if <code>TRUE</code> , shows optimization output
<code>method</code>	optimization method
<code>hessian</code>	if <code>TRUE</code> , computes v-c matrix using hessian
<code>accumulate</code>	if <code>TRUE</code> , like capture-histories are accumulated to reduce computation
<code>chunk_size</code>	specifies amount of memory to use in accumulating capture histories and design matrices; amount used is $8 * \text{chunk_size} / 1e6$ MB (default 80MB)
<code>control</code>	control string for optimization functions
<code>refit</code>	non-zero entry to refit
<code>itnmax</code>	maximum number of iterations for optimization
<code>scale</code>	vector of scale values for parameters
<code>run</code>	if <code>TRUE</code> , it runs model; otherwise if <code>FALSE</code> can be used to test model build components
<code>compile</code>	if <code>TRUE</code> forces re-compilation of <code>tpl</code> file
<code>extra.args</code>	optional character string that is passed to <code>admb</code> if <code>use.admb==TRUE</code>
<code>strata.labels</code>	labels for strata used in capture history; they are converted to numeric in the order listed. Only needed to specify unobserved strata. For any unobserved strata $p=0..$
<code>clean</code>	if <code>TRUE</code> , deletes the <code>tpl</code> and executable files for <code>admb</code> if <code>use.admb=T</code>
<code>save.matrices</code>	for HMM models this option controls whether the <code>gamma</code> , <code>dmat</code> and <code>delta</code> matrices are saved in the model object

savef	if TRUE, save the makeAdFun result from TMB to report real values and matrices
getreals	if TRUE, compute real values and std errors for TMB models; may want to set as FALSE until model selection is complete
real.ids	vector of id values for which real parameters should be output with std error information for TMB models; if NULL all ids used
check	if TRUE values of gamma, dmat and delta are checked to make sure the values are valid with initial parameter values.
useHess	if TRUE, the TMB hessian function is used for optimization; using hessian is typically slower with many parameters but can result in a better solution
optimize	if TRUE, optimizes to get parameter estimates; set to FALSE to extract estimates of ADREPORTed values only
vcv	if TRUE, computes var-covariance matrix of ADREPORTed values
unit_scale	default TRUE, if FALSE any time scaled parameter (e.g. Phi,S) is scaled when computing real value such that it represents the length of the interval rather than a unit interval
...	optional arguments passed to js or cjs and optimx

Details

This function is operationally similar to the function `mark` in RMark in that it is a shell that calls several other functions to perform the following steps: 1) `process.data` to setup data and parameters and package them into a list (processed data), 2) `make.design.data` to create the design data for each parameter in the specified model, 3) `create.dm` to create the design matrices for each parameter based on the formula provided for each parameter, 4) call to the specific function for model fitting. As with `mark` the calling arguments for `crm` are a compilation of the calling arguments for each of the functions it calls (with some arguments renamed to avoid conflicts). `crm` expects to find a value for `ddl`. Likewise, if the data have not been processed, then `ddl` should be NULL. This dual calling structure allows either a single call approach for each model or alternatively the preferred method where the data are processed and the design data (`ddl`) created once and then a whole series of models can be analyzed without repeating those steps.

There are some optional arguments that can be used to set initial values and control other aspects of the optimization. The optimization is done with the R package/function `optimx` and the arguments `method` and `hessian` are described with the help for that function.

In the current implementation, a logit link is used to constrain the parameters in the unit interval (0,1) except for probability of entry which uses an `mlogit` and `N` which uses a log link. For the `probitCJS` model, a `probit` link is used for the parameters. These could be generalized to use other link functions. Following the notation of MARK, the parameters in the link space are referred to as `beta` and those in the actual parameter space of `Phi` and `p` as reals.

Initial values can be set in 2 ways. 1) Define a list of named vectors with the initial beta parameter values (eg logit link) in `initial`. The names of the vectors should be the parameter names in the model. Any unspecified values are set to 0. 2) Specify a previously run model for `initial`. The code will match the names of the current design matrix to the names in `beta` and use the appropriate initial values. Any non-specified values are set to 0. If no value is specified for `initial`, all beta are started at a value of 0.

If you have a study with unequal time intervals between capture occasions, then these can be specified with the argument `time.intervals`.

The argument `accumulate` defaults to TRUE. When it is TRUE it will accumulate common capture histories that also have common design and common fixed values (see below) for the parameters. This will speed up the analysis because in the calculation of the likelihood it loops over the unique values. In general the default will be the best unless you have many capture histories and are using many individual covariate(s) in the formula that would make each entry unique. In that case there will be no effect of accumulation but the code will still try to accumulate. In that particular case by setting `accumulate=FALSE` you can skip the code run for accumulation.

Most of the arguments controlling the fitted model are contained in lists in the arguments `model.parameters` and `design.parameters` which are similar to their counterparts in `mark` in `RMark`. Each is a named list with the names being the parameters in the model (e.g., `Phi` and `p` in "cjs" and "Phi", "p", "pent", "N" in "js"). Each named element is also a list containing various values defining the design data and model for the parameter. The elements of `model.parameters` can include `formula` which is an R formula to create the design matrix for the parameter and `fixed` is a matrix of fixed values as described below. The elements of `design.parameters` can include `time.varying`, `fields`, `time.bins`, `age.bins`, and `cohort.bins`. See [create.dmdf](#) for a description of the first 2 and [create.dm](#) for a description of the last 3.

Real parameters can be set to fixed values by specifying values for a field called `fix` in the design data for a parameter. If the value of `fix` is NA the parameter is estimated and if it is not NA then the real parameter is fixed at that value. If you also specify `fixed` as described above, they will over-ride any values you have also set with `fix` in the design data. To set all of the real values for a particular occasion you can use the following example with the `dipper` data as a template:

```
model.parameters=list(Phi=list(formula=~1, fixed=cbind(1:nrow(dipper), rep(2, nrow(dipper))), rep(1, nrow(dipper))))
```

The above sets `Phi` to 1 for the interval between occasions 2 and 3 for all animals.

Alternatively, you could do as follows:

```
data(dipper) dp=process.data(dipper) ddl=make.design.data(dp) ddl$Phi$fix=ifelse(ddl$Phi$time==2,1,NA)
```

At present there is no modification of the parameter count to address fixing of real parameters except that if by fixing reals, a beta is not needed in the design it will be dropped. For example, if you were to use `~time` for `Phi` with survival fixed to 1 for time 2, then then beta for that time would not be included.

Value

crm model object with class=("crm", submodel)

Author(s)

Jeff Laake

See Also

[make.design.data](#), [process.data](#)

 crm.wrapper

Automation of model runs

Description

Some functions that help automate running a set of crm models based on parameter specifications.

Usage

```
crm.wrapper(model.list, data, ddl=NULL, models=NULL, base="",
            external=TRUE, run=TRUE, env=NULL, replace=FALSE, ...)

create.model.list(parameters)

model.table(model.list)

load.model(x)

crmlist_fromfiles(filenamees=NULL, external=TRUE)

rerun_crm(data, ddl, model.list, method=NULL, modelnums=NULL, initial=NULL, ...)
```

Arguments

model.list	matrix of model names contained in the environment of models function; each row is a model and each column is for a parameter and the value is formula name
data	Either the raw data which is a dataframe with at least one column named ch (a character field containing the capture history) or a processed dataframe. For rerun_crm this should be the processed dataframe
ddl	Design data list which contains a list element for each parameter type; if NULL it is created; For rerun_crm, must be the same ddl as used with original run cannot be NULL
models	a function with a defined environment with model specifications as variables; values of model.list are some or all of those variables
base	base value for model names
external	if TRUE, model results are stored externally; otherwise they are stored in crmlist
run	if TRUE, fit models; otherwise just create dml to test if model data are correct for formula
env	environment to find model specifications if not parent.frame
replace	if TRUE replace any existing externally saved model
...	additional arguments passed to crm; for rerun_crm can be used to set hessian=TRUE for specific models after they have been run
parameters	character vector of parameter names
x	filename of externally stored model

method	vector of methods to use for optimization if different than previous run in re-run_crm
modelnums	model numbers to be re-run instead of those that did not converge
initial	either a fitted crm model or the model number in model.list to use for starting values
filenames	for non-Windows machine, vector of filenames for external files must be specified in crmlist_fromfiles including .rda extension

Details

create.model.list creates all combinations of model specifications for the specified set of parameters. In the calling environment it looks for objects named parameter.xxxxxx where xxxxxx can be anything. It creates a matrix with a column for each parameter and as many rows needed to create all combinations. This can be used as input to crm.wrapper.

crm.wrapper runs a sequence of crm models by constructing the call with the arguments and the parameter specifications. The parameter specifications can either be in the local environment or in the environment of the named function models. The advantage of the latter is that it is self-contained such that sets of parameter specifications can be selected without possibility of being over-written or accidentally changed whereas with the former the set must be identified via a script and any in the environment will be used which requires removing/recreating the set to be used.

Value

create.model.list returns a matrix for crm.wrapper; crm.wrapper runs and stores models externally and returns a list of model results and a model selection table; load.model returns model object that is stored externally

Author(s)

Jeff Laake

See Also

[crm](#)

deriv_inverse.link *Derivatives of inverse of link function (internal use)*

Description

Computes derivatives of inverse of link functions (real estimates) with respect to the beta parameters which are used for delta method computation of the var-cov matrix of real parameters.

Usage

```
deriv_inverse.link(real, x, link)
```

Arguments

real	Vector of values of real parameters
x	Matrix of design values
link	Type of link function (e.g., "logit")

Details

Note: that function was renamed to `deriv_inverse.link` to avoid S3 generic class conflicts. The derivatives of the inverse of the link functions are simple computations using the real values and the design matrix values. The body of the function is as follows:

```
switch(link, logit=x*real*(1-real), log=x*real,
loglog=-real*x*log(real), cloglog=-log(1-real)*x*(1-real), identity=x,
mlogit=x*real*(1-real))
```

Value

Vector of derivative values computed at values of real parameters

Author(s)

Jeff Laake

See Also

[inverse.link](#), [compute_real](#)

dipper

Dipper capture-recapture data

Description

A capture-recapture data set on European dippers from France that accompanies MARK as an example analysis using the CJS and POPAN models. The dipper data set was originally described as an example by Lebreton et al (1992).

Format

A data frame with 294 observations on the following 2 variables.

ch a character vector containing the encounter history of each bird

sex the sex of the bird: a factor with levels Female Male

Details

There isn't a specific CJS model in markedTMB but you can use the model MSCJS with a dummy state that is never observed. This example shows how that can be done to fit standard JS models.

Note that the covariate "sex" defined in dipper has values "Male" and "Female". It cannot be used directly in a formula for MARK without using it to define groups because MARK.EXE will be unable to read in a covariate with non-numeric values. By using groups="sex" in the call the `process.data` a factor "sex" field is created that can be used in the formula. Alternatively, a new covariate could be defined in the data with say values 0 for Female and 1 for Male and this could be used without defining groups because it is numeric. This can be done easily by translating the values of the coded variables to a numeric variable. Factor variables are numbered 1..k for k levels in alphabetic order. Since Female < Male in alphabetic order then it is level 1 and Male is level 2. So the following will create a numeric sex covariate.

```
dipper$numeric.sex=as.numeric(dipper$sex)-1
```

Source

Lebreton, J.-D., K. P. Burnham, J. Clobert, and D. R. Anderson. 1992. Modeling survival and testing biological hypotheses using marked animals: case studies and recent advances. *Ecol. Monogr.* 62:67-118.

Examples

```
data(dipper)
ms1=process.data(dipper,model="MSCJS",strata.labels=c("1","2"))
ms1.ddl=make.design.data(ms1)
ms1.ddl$Psi$fix[ms1.ddl$Psi$stratum!=ms1.ddl$Psi$tostratum]=0
ms1.ddl$p$fix=NA
ms1.ddl$p$fix[ms1.ddl$p$stratum==2]=0
mod_cjs=crm(ms1,ms1.ddl,model.parameters=list(S=list(formula=~1),
                                                    p=list(formula=~time)),hessian=TRUE)
hmm_mat=compute_matrices(mod_cjs,state.names=c("Alive","NULL","Dead"))
```

dmat_hsmm2hmm

Create expanded state-dependent observation matrix for HMM from HSMM

Description

Creates expanded state-dependent matrix for HMM from aggregated state-dependent observation matrix.

Usage

```
dmat_hsmm2hmm(dmat, mv)
```

Arguments

dmat	state-dependent observation matrix for aggregated states
mv	vector of dwell time distribution lengths

Value

expanded state-dependent observation matrix for hmm to approximate hsmm

Author(s)

Jeff Laake

fix.parameters	<i>Fixing real parameters in crm models</i>
----------------	---

Description

Creates matrix with appropriate structure for fixing real parameters in a crm model for a specified set of capture histories and occasions.

Usage

```
fix.parameters(x, subset = NULL, occasions, values)
```

Arguments

x	processed data list
subset	logical expression for set of animals; used in subset function
occasions	vector of occasion numbers
values	either single or vector of values; if latter, must match length of occasions

Value

Matrix with 3 columns: 1) animal row number, 2) occasion, 3) value for real parameter

Author(s)

Jeff Laake

function.wrapper	<i>Utility extract functions</i>
------------------	----------------------------------

Description

Several functions have been added to extract components from externally saved crm models function.wrapper accepts a character string for a model, loads it and then runs the specified function on it. Currently, only 2 functions have been defined: fx.aic to compute aic and fx.par.count to extract parameter count. Currently parameters other than x (eg chat) are passed through the environment. Possibly could have used ...

Usage

```
function.wrapper(x,fx,base="",...)
```

```
fx.aic(x)
```

```
fx.par.count(x)
```

Arguments

x	character string for the model stored as external object (.rda)
fx	function to be called for each model
base	base name for models
...	additional values that are added to environment of fx (eg chat for fx.aic)

Value

extracted value defined by function

Author(s)

Jeff Laake

global_decode	<i>Global decoding of HMM</i>
---------------	-------------------------------

Description

Computes sequence of state predictions for each individual

Usage

```
global_decode(object, state.names = NULL, par = NULL)
```

Arguments

object	fitted crm model (must be an HMM model)
state.names	names to over-ride default names for states
par	if not NULL use these values for matrix calculations

Value

matrix of state predictions

Author(s)

Jeff Laake

References

Zucchini, W. and I.L. MacDonald. 2009. Hidden Markov Models for Time Series: An Introduction using R. Chapman and Hall, Boca Raton, FL. See page 82.

hsmm2hmm

Compute transition matrix for HMM from HSMM

Description

Computes expanded transition matrix for HMM from aggregated state transition matrix and dwell time distributions.

Usage

```
hsmm2hmm(omega, dm, eps = 1e-10)
```

Arguments

omega	transition matrix for aggregated states
dm	list of dwell time distribution vectors; each list element is vector of dwell time probabilities in order of aggregated states
eps	epsilon value for setting small probabilities to 0; anything less than eps is set to exactly 0.

Value

transition matrix for hmm to approximate hsmm

Author(s)

Walter Zucchini

initiate_pi	<i>Setup fixed values for pi in design data</i>
-------------	---

Description

Creates field fix in pi design data and either sets to NA to be estimated or 1 if intercept or 0 if not possible. It uses observed data at initial entry to decide the appropriate fix values.

Usage

```
initiate_pi(data, ddl)
```

Arguments

data	Processed data list; resulting value from process.data
ddl	design data list created by make.design.data

Details

It is possible that the state will be known for some individuals and unknown for others. The function `initiate_pi` modifies the design data for pi by adding the field `fix` which can be used to fix the value of pi for an individual with known state at release. It also sets up the mlogit structure for pi by setting one of the fixed values to 1 such that the value for pi is computed by subtraction. By default it chooses the first factor level of each of the state variables that is unknown. If some of the state variables are known but others are not, it sets `fix` to 0 for the values of the known state variables that don't match the known value.

For example, with a bivariate case with first state A,B or C and second variable 1 or 2. Only the second variable can be unknown. There are 6 possible combinations of the state variables: A1,A2,B2,B2,C1,C2. Assume a capture history is 0,Au,0,B1,B2. There will be 6 records in the design data for this record with strata A1,A2,B2,B2,C1,C2. The `occ` (occasion field) will be 2 because pi is only for the first occasion. After running `initiate_pi`, the fix values will be 1,NA,0,0,0,0. A1 will be the intercept and computed by subtraction (`fix=1`), the value for A2 will be estimated (NA) and all of the others are not possible (0) because it is known to be in A.

If both can be uncertain, and the capture history was 0,uu,0,B1,B2, then the fix values will be 1,NA,NA,NA,NA,NA. With a mix of partially known and completely unknown the formulation for the mlogit could get complicated making it difficult to specify an understandable formula because the restrictions on the mlogit will change, so think hard and long about what you are doing. It may be necessary to construct "multiple" formula in the sense that you use indicator variables (0/1) as interactions.

Value

ddl with `fix` field added to pi dataframe

Author(s)

Jeff Laake

inverse.link *Inverse link functions (internal use)*

Description

Computes values of inverse of link functions for real estimates.

Usage

```
inverse.link(x, link)
```

Arguments

x	Matrix of design values multiplied by the vector of the beta parameter values
link	Type of link function (e.g., "logit")

Details

The inverse of the link function is the real parameter value. They are simple functions of $X \cdot \text{Beta}$ where X is the design matrix values and Beta is the vector of link function parameters. The body of the function is as follows:

```
switch(link, logit=exp(x)/(1+exp(x)), log=exp(x),
loglog=exp(-exp(-x)), cloglog=1-exp(-exp(x)), identity=x,
mlogit=exp(x)/(1+sum(exp(x))) )
```

The link="mlogit" only works if the set of real parameters are limited to those within the set of parameters with that specific link. For example, in POPAN, the pent parameters are of type "mlogit" so the probabilities sum to 1. However, if there are several groups then each group will have a different set of pent parameters which are identified by a different grouping of the "mlogit" parameters (i.e., "mlogit(1)" for group 1, "mlogit(2)" for group 2 etc). Thus, in computing real parameter values (see [compute_real](#)) which may have varying links, those with "mlogit" are not used with this function using link="mlogit". Instead, the link is temporarily altered to be of type "log" (i.e., inverse=exp(x)) and then summed over sets with a common value for "mlogit(j)" to construct the inverse for "mlogit" as $\exp(x)/(1+\text{sum}(\exp(x)))$.

Value

Vector of real values computed from $x=X \cdot \text{Beta}$

Author(s)

Jeff Laake

See Also

[compute_real](#), [deriv_inverse.link](#)

load_lib	<i>Load library for TMB model</i>
----------	-----------------------------------

Description

Loads the model specific library (eg dll) for a model

Usage

```
load_lib(model)
```

Arguments

model	type of crm model
-------	-------------------

local_decode	<i>Local decoding of HMM</i>
--------------	------------------------------

Description

Computes state predictions one at a time for each occasion for each individual

Usage

```
local_decode(object, state.names = NULL, par = NULL)
```

Arguments

object	fitted crm model (must be an HMM model)
state.names	names for states if over-riding defaults
par	if not NULL use these values for matrix calculations

Value

matrix of state predictions

Author(s)

Jeff Laake

References

Zucchini, W. and I.L. MacDonald. 2009. Hidden Markov Models for Time Series: An Introduction using R. Chapman and Hall, Boca Raton, FL. See page 80.

make.design.data *Create design dataframes for crm*

Description

For each type of parameter in the analysis model (e.g, p, Phi), this function creates design data for model fitting from the original data. These design data expand the original data record for a single(freq=1)/multiple(freq>1) individuals to many records where each record is for a single occasion. The design data can be specific to the parameter and a list of design data frames are returned with a dataframe for each parameter.

Usage

```
make.design.data(data, parameters = list())
```

Arguments

data	Processed data list; resulting value from process.data
parameters	Optional list containing a list for each type of parameter (list of lists); each parameter list is named with the parameter name (eg Phi); each parameter list can contain vectors named age.bins,time.bins and cohort.bins, time.varying, and static
age.bins	bins for binning ages
time.bins	bins for binning times
cohort.bins	bins for binning cohorts
time.varying	vector of field names that are time varying for this parameter
static	vector of field names that are to be included that are not time varying for this parameter

Details

For each record in the design data, default data fields are created based on the model. For example, for CJS, the default fields include cohort, age, time which are factor variables and Cohort, Age, and Time which are numeric versions of the same fields. The values of these can be altered by values of begin.time, time.intervals and initial.ages set in the call to [process.data](#). In addition if groups are identified the grouping factor variables are added to the design data. Any other fields in the data are repeated on each record for the animal(s) (eg weight), unless they are defined as time.varying in which case the fields should be named with the convention xn where x is the base name for the variable and n is the time value (eg, td1999, td2000,...). For time varying fields the variable name in the design data is the base name and the value for the record is the one for that occasion(time). The variables can be refined for each parameter by including argument static with the character vector of static fields to include.

After creating design data, you can create a field in the dataframe for a parameter named fix and it can be assigned values at the real parameter for that occasion/id will be fixed at the value. For parameters that are not supposed to be fixed, the field fix should be assigned NA. For example,

ddl\$Phi\$fix=ifelse(ddl\$Phi\$time==2,1,NA) will assign all real Phi values to 1 for the interval beginning at time 2. If there is no field fix, then no parameters are fixed. For mlogit parameters, a fix field is added automatically and the value 1 is specified for the stratum that is supposed to be computed by subtraction and the remainder are set to NA. If some Psi are not possible then those values can be changed to 0.

The following variable names are reserved and should not be used in the data: occ,age,time,cohort,Age,Time,Cohort,Y,Z,initial

Value

The function value is a list of data frames. The list contains a data frame for each type of parameter in the model (e.g., Phi and p for CJS). The names of the list elements are the parameter names (e.g., Phi). The structure of the dataframe depends on the calling arguments and the model & data structure. In addition the value of parameters argument is saved as design.parameters.

Author(s)

Jeff Laake

See Also

[process.data](#), [merge_design.covariates](#)

merge_design.covariates

Merge time (occasion) and/or group specific covariates into design data

Description

Adds new design data fields from a dataframe into a design data list (ddl) by matching via time and/or group field in the design data.

Usage

```
merge_design.covariates(ddl, df, bygroup = FALSE, bytime = TRUE)
```

Arguments

ddl	current design dataframe for a specific parameter and not the entire design data list (ddl)
df	dataframe with time(occasion) and/or group-specific data
bygroup	logical; if TRUE, then a field named group should be in df and the values can then be group specific.
bytime	logical; if TRUE, then a field named time should be in df and the values can then be time specific.

Details

Design data can be added to the parameter specific design dataframes with R commands. This function simplifies the process by enabling the merging of a dataframe with a time and/or group field and one or more time and/or group specific covariates into the design data list for a specific model parameter. This is a replacement for the older function `merge.occasion.data`. Unlike the older function, it uses the R function `merge` but before merging it makes sure all of the fields exist and that you are not merging data that already exists in the design data. It also maintains the row names in the case where design data have been deleted prior to merging the design covariate data.

If `bytime=TRUE`, the dataframe `df` must have a field named `time` that matches 1-1 for each value of `time` in the design data list (`ddl`). All fields in `df` (other than `time/group`) are added to the design data. If you set `bygroup=TRUE` and have a field named `group` in `df` and its values match the group fields in the design data then group-specific values can be assigned for each time if `bytime=TRUE`. If `bygroup=TRUE` and `bytime=FALSE` then it matches by group and not by time.

Value

Design dataframe (for a particular parameter) with new fields added. See `make.design.data` for a description of the design data list structure. The return value is only one element in the list rather than the entire list as with the older function `merge.occasion.data`.

Author(s)

Jeff Laake

See Also

[make.design.data](#), [process.data](#)

mixed.model.admb

Mixed effect model construction

Description

Functions that develop structures needed for a mixed effect model

Usage

```
mixed.model.admb(formula, data)

mixed.model(formula, data, indices=FALSE)

mixed.model.dat(x, con, idonly, n)

reindex(x, id)
```


Arguments

formula	formula for mixed effect mode in the form used in lme4; ~fixed +(re lg1) +...+(ren gn)
data	dataframe used to construct the design matrices from the formula
x	list structure created by mixed.model.admb
con	connection to data file which contents will be appended
id	vector of factor values used to split the data up by individual capture history
idonly	TRUE, if random effects not crossed
n	number of capture history records
indices	if TRUE, outputs structure with indices into dm for random effects

Details

mixed.model.admb - creates design matrices and supporting index matrices for use of mixed model in ADMB

mixed.model - creates design matrices and supporting index matrices in an alternate list format that is not as easily used in ADMB

mixed.model.dat - writes to data file (con) for fixed and random effect structures

reindex - creates indices for random effects that are specific to the individual capture history; it takes re.indices, splits them by id and creates a ragged array by id (used.indices) with the unique values for that id. index.counts is the number of indices per id to read in ragged array. It then changes re.indices to be an index to the indices within the id from 1 to the number of indices within the id.

Value

mixed.model.admb returns a list with elements re.dm, a combined design matrix for all of the random effects; and re.indices, matrix of indices into a single vector of random effects to be applied to the design matrix location. mixed.model returns a list (re.list) with an element for each random effect structure. The contents are a standard design matrix (re.dm) if indices==FALSE and a re.dm and re.indices which matches the structure of mixed.model.admb. mixed.model will be more useful with R than ADMB.

Author(s)

Jeff Laake

mscjs_tmb

Fitting function for Multistate CJS models with TMB

Description

A function for computing MLEs for a Multi-state Cormack-Jolly-Seber open population capture-recapture model for processed dataframe x with user specified formulas in parameters that create list of design matrices dm1. This function can be called directly but is most easily called from `crm` that sets up needed arguments.

Usage

```

mscjs_tmb(
  x,
  ddl,
  fullddl,
  dml,
  model_data = NULL,
  parameters,
  accumulate = TRUE,
  initial = NULL,
  method,
  hessian = FALSE,
  debug = FALSE,
  chunk_size = 1e+07,
  refit,
  itnmax = NULL,
  control = NULL,
  scale,
  re = FALSE,
  extra.args = "",
  getreals = FALSE,
  useHess = FALSE,
  savef = TRUE,
  ...
)

```

Arguments

<code>x</code>	processed dataframe created by <code>process.data</code>
<code>ddl</code>	list of simplified dataframes for design data
<code>fullddl</code>	list of complete dataframes for design data; created by call to <code>make.design.data</code>
<code>dml</code>	list of design matrices created by <code>create.dm</code> from formula and design data
<code>model_data</code>	a list of all the relevant data for fitting the model including <code>imat</code> , <code>S.dm</code> , <code>p.dm</code> , <code>Psi.dm</code> , <code>S.fixed</code> , <code>p.fixed</code> , <code>Psi.fixed</code> and <code>time.intervals</code> . It is used to save values and avoid accumulation again if the model was re-rerun with an additional call to <code>cjs</code> when using <code>autoscale</code> or re-starting with initial values. It is stored with returned model object.
<code>parameters</code>	equivalent to <code>model.parameters</code> in <code>crm</code>
<code>accumulate</code>	if <code>TRUE</code> will accumulate capture histories with common value and with a common design matrix all parameters to speed up execution
<code>initial</code>	list of initial values for parameters if desired; if each is a named vector from previous run it will match to columns with same name
<code>method</code>	method to use for optimization; see <code>optim</code>
<code>hessian</code>	if <code>TRUE</code> will compute and return the hessian
<code>debug</code>	if <code>TRUE</code> will print out information for each iteration

chunk_size	specifies amount of memory to use in accumulating capture histories; amount used is $8 \times \text{chunk_size} / 1e6$ MB (default 80MB)
refit	non-zero entry to refit
itnmax	maximum number of iterations
control	control string for optimization functions
scale	vector of scale values for parameters
re	if TRUE creates random effect model admbcjsre.tpl and runs admB optimizer
extra.args	optional character string that is passed to tmb
getreals	if TRUE, compute real values and std errors for TMB models; may want to set as FALSE until model selection is complete
useHess	if TRUE, the TMB hessian function is used for optimization; using hessian is typically slower with many parameters but can result in a better solution
savef	if TRUE, save optimization function in model for reporting
...	not currently used

Details

It is easiest to call `mscjs_tmb` through the function `crm`. Details are explained there.

Value

The resulting value of the function is a list with the class of `crm,cjs` such that the generic functions `print` and `coef` can be used.

beta	named vector of parameter estimates
lnl	$-2 \times \log$ likelihood
AIC	$\ln l + 2 \times$ number of parameters
convergence	result from <code>optim</code> ; if 0 <code>optim</code> thinks it converged
count	<code>optim</code> results of number of function evaluations
reals	dataframe of data and real S and p estimates for each animal-occasion excluding those that occurred before release
vcv	var-cov matrix of betas if <code>hessian=TRUE</code> was set

Author(s)

Jeff Laake

References

Ford, J. H., M. V. Bravington, and J. Robbins. 2012. Incorporating individual variability into mark-recapture models. *Methods in Ecology and Evolution* 3:1047-1054.

Examples

```

# this example requires TMB
# The same example is in the RMark package and it is included here to
# illustrate the differences in the handling of mlogit parameters between RMark
# and marked. The MARK software handles parameters like Psi which must sum to 1
# by excluding one of the cells that is used as a reference cell and is computed by
# subtracting the other cell values from 1 so the total sums to 1. This is often
# handled with an mlogit parameter in which the cell values are exp(beta) and the
# reference cell is set to 1 and the values are divided by the sum across the cells
# so the resulting values are probabilities that sum to 1. In marked, instead of removing
# one of the cells, all are included and the user must select which should be the
# reference cell by setting the value fix=1 for that cell and others are NA so they are
# estimated. For transition parameters like Psi, the default design data is setup so
# that the probability of remaining in the cell (stratum=tostratum) is the reference cell
# and fix set to 1. Thus, this means 2 changes are needed to the script in RMark.
# The first is to remove the statement skagit.ddl$Psi$fix=NA because that over-rides
# the default fix values. The other is to add
# skagit.ddl$Psi$fix[skagit.ddl$Psi$stratum=="B"&skagit.ddl$Psi$tostratum=="B"&
# skagit.ddl$Psi$time==5]=0
# to change the value from 1 to 0 which forces movement from B to A in the interval 5 to 6. If
# this is not done then Psi B to B=Psi B to A=0.5 because each is 1 and when they are normalized
# they are divided by the sum which is 2 (1/2).
data(skagit)
skagit.processed=process.data(skagit,model="MSCJS",groups=c("tag"),strata.labels=c("A","B"))
skagit.ddl=make.design.data(skagit.processed)
#
# p
#
# Can't be seen at 5A or 2B,6B (the latter 2 don't exist)
skagit.ddl$p$fix=ifelse((skagit.ddl$p$stratum=="A"&skagit.ddl$p$time==5) |
(skagit.ddl$p$stratum=="B"&skagit.ddl$p$time%in%c(2,6)),0,NA)
# Estimated externally from current data to allow estimation of survival at last interval
skagit.ddl$p$fix[skagit.ddl$p$tag=="v7"&skagit.ddl$p$time==6&skagit.ddl$p$stratum=="A"]=0.687
skagit.ddl$p$fix[skagit.ddl$p$tag=="v9"&skagit.ddl$p$time==6&skagit.ddl$p$stratum=="A"]=0.975
#
# Psi
#
# only 3 possible transitions are A to B at time interval 2 to 3 and
# for time interval 3 to 4 from A to B and from B to A
# rest are fixed values
##### change for RMark to marked; remove next line
#skagit.ddl$Psi$fix=NA
# stay in A for intervals 1-2, 4-5 and 5-6
skagit.ddl$Psi$fix[skagit.ddl$Psi$stratum=="A"&
skagit.ddl$Psi$tostratum=="B"&skagit.ddl$Psi$time%in%c(1,4,5)]=0
# stay in B for interval 4-5
skagit.ddl$Psi$fix[skagit.ddl$Psi$stratum=="B"&skagit.ddl$Psi$tostratum=="A"&
&skagit.ddl$Psi$time==4]=0
# leave B to go to A for interval 5-6
skagit.ddl$Psi$fix[skagit.ddl$Psi$stratum=="B"&skagit.ddl$Psi$tostratum=="A"&
skagit.ddl$Psi$time==5]=1
##### change for RMark to marked; add next line to set B to B to 0 otherwise it has

```

```
##### been set to 1 by default which would make psi B to B = psi B to A = 0.5
skagit.ddl$Psi$fix[skagit.ddl$Psi$stratum=="B"&skagit.ddl$Psi$tostratum=="B"&
skagit.ddl$Psi$time==5]=0
# "stay" in B for interval 1-2 and 2-3 because none will be in B
skagit.ddl$Psi$fix[skagit.ddl$Psi$stratum=="B"&skagit.ddl$Psi$tostratum=="A"&
skagit.ddl$Psi$time%in%1:2]=0
#
# S
#
# None in B, so fixing S to 1
skagit.ddl$S$fix=ifelse(skagit.ddl$S$stratum=="B"&skagit.ddl$S$time%in%c(1,2),1,NA)
skagit.ddl$S$fix[skagit.ddl$S$stratum=="A"&skagit.ddl$S$time==4]=1
# fit model
p.timexstratum.tag=list(formula=~time:stratum+tag,remove.intercept=TRUE)
Psi.sxtime=list(formula=~-1+stratum:time)
S.stratumxtime=list(formula=~-1+stratum:time)
#
mod1=crm(skagit.processed,skagit.ddl,
model.parameters=list(S=S.stratumxtime,p= p.timexstratum.tag,Psi=Psi.sxtime),hessian=TRUE)
```

msjsu_tmb

*Fitting function for Multistate JS models with state uncertainty via
TMB*

Description

A function for computing MLEs for a Multi-state Jolly-Seber open population capture-recapture model for processed dataframe `x` with user specified formulas in `parameters` that create list of design matrices `dml`. This function can be called directly but is most easily called from `crm` that sets up needed arguments.

Usage

```
msjsu_tmb(
  x,
  ddl,
  fullddl,
  dml,
  model_data = NULL,
  parameters,
  accumulate = TRUE,
  initial = NULL,
  method,
  hessian = FALSE,
  debug = FALSE,
  chunk_size = 1e+07,
  refit,
  itnmax = NULL,
```

```

control = NULL,
scale,
re = FALSE,
extra.args = "",
getreals = FALSE,
useHess = FALSE,
savef = TRUE,
...
)

```

Arguments

x	processed dataframe created by <code>process.data</code>
ddl	list of simplified dataframes for design data
fullddl	list of complete dataframes for design data; created by call to <code>make.design.data</code>
dml	list of design matrices created by <code>create.dm</code> from formula and design data
model_data	a list of all the relevant data for fitting the model including <code>imat</code> , <code>S.dm</code> , <code>p.dm</code> , <code>Psi.dm</code> , <code>pent.dm</code> , <code>pi.dm</code> , <code>S.fixed</code> , and <code>time.intervals</code> . It is used to save values and avoid accumulation again if the model was re-rerun with an additional call when using <code>autoscale</code> or re-starting with initial values. It is stored with returned model object.
parameters	equivalent to <code>model.parameters</code> in <code>crm</code>
accumulate	if TRUE will accumulate capture histories with common value and with a common design matrix all parameters to speed up execution
initial	list of initial values for parameters if desired; if each is a named vector from previous run it will match to columns with same name
method	method to use for optimization; see <code>optim</code>
hessian	if TRUE will compute and return the hessian
debug	if TRUE will print out information for each iteration
chunk_size	specifies amount of memory to use in accumulating capture histories; amount used is $8 * \text{chunk_size} / 1e6$ MB (default 80MB)
refit	non-zero entry to refit
itnmax	maximum number of iterations
control	control string for optimization functions
scale	vector of scale values for parameters
re	if TRUE creates random effect model <code>admbcjsre.tpl</code> and runs <code>admb</code> optimizer
extra.args	optional character string that is passed to <code>tmb</code>
getreals	if TRUE, compute real values and std errors for TMB models; may want to set as FALSE until model selection is complete
useHess	if TRUE, the TMB hessian function is used for optimization; using hessian is typically slower with many parameters but can result in a better solution
savef	if TRUE, save optimization function in model for reporting
...	not currently used

Details

It is easiest to call `msjs_tmb` through the function `crm`. Details are explained there.

Value

The resulting value of the function is a list with the class of `crm,msjs` such that the generic functions `print` and `coef` can be used.

<code>beta</code>	named vector of parameter estimates
<code>lnl</code>	$-2 \cdot \log$ likelihood
<code>AIC</code>	$\ln l + 2 \cdot$ number of parameters
<code>convergence</code>	result from <code>optim</code> ; if 0 <code>optim</code> thinks it converged
<code>count</code>	<code>optim</code> results of number of function evaluations
<code>reals</code>	dataframe of data and real S and p estimates for each animal-occasion excluding those that occurred before release
<code>vcv</code>	var-cov matrix of betas if <code>hessian=TRUE</code> was set

Author(s)

Jeff Laake

msjs_tmb

Fitting function for Multistate JS models with TMB

Description

A function for computing MLEs for a Multi-state Jolly-Seber open population capture-recapture model for processed dataframe `x` with user specified formulas in `parameters` that create list of design matrices `dml`. This function can be called directly but is most easily called from `crm` that sets up needed arguments.

Usage

```
msjs_tmb(
  x,
  ddl,
  fullddl,
  dml,
  model_data = NULL,
  parameters,
  accumulate = TRUE,
  initial = NULL,
  method,
  hessian = FALSE,
  debug = FALSE,
```

```

    chunk_size = 1e+07,
    refit,
    itnmax = NULL,
    control = NULL,
    scale,
    re = FALSE,
    extra.args = "",
    getreals = FALSE,
    useHess = FALSE,
    savef = TRUE,
    ...
)

```

Arguments

<code>x</code>	processed dataframe created by <code>process.data</code>
<code>ddl</code>	list of simplified dataframes for design data
<code>fullddl</code>	list of complete dataframes for design data; created by call to <code>make.design.data</code>
<code>dml</code>	list of design matrices created by <code>create.dm</code> from formula and design data
<code>model_data</code>	a list of all the relevant data for fitting the model including <code>imat</code> , <code>S.dm</code> , <code>p.dm</code> , <code>Psi.dm</code> , <code>pent.dm</code> , <code>pi.dm</code> , <code>S.fixed</code> , and <code>time.intervals</code> . It is used to save values and avoid accumulation again if the model was re-rerun with an additional call when using <code>autoscale</code> or re-starting with initial values. It is stored with returned model object.
<code>parameters</code>	equivalent to <code>model.parameters</code> in <code>crm</code>
<code>accumulate</code>	if TRUE will accumulate capture histories with common value and with a common design matrix all parameters to speed up execution
<code>initial</code>	list of initial values for parameters if desired; if each is a named vector from previous run it will match to columns with same name
<code>method</code>	method to use for optimization; see <code>optim</code>
<code>hessian</code>	if TRUE will compute and return the hessian
<code>debug</code>	if TRUE will print out information for each iteration
<code>chunk_size</code>	specifies amount of memory to use in accumulating capture histories; amount used is $8 * \text{chunk_size} / 1e6$ MB (default 80MB)
<code>refit</code>	non-zero entry to refit
<code>itnmax</code>	maximum number of iterations
<code>control</code>	control string for optimization functions
<code>scale</code>	vector of scale values for parameters
<code>re</code>	if TRUE creates random effect model <code>admbcjsre.tpl</code> and runs <code>admb</code> optimizer
<code>extra.args</code>	optional character string that is passed to <code>tmb</code>
<code>getreals</code>	if TRUE, compute real values and std errors for TMB models; may want to set as FALSE until model selection is complete
<code>useHess</code>	if TRUE, the TMB hessian function is used for optimization; using hessian is typically slower with many parameters but can result in a better solution
<code>savef</code>	if TRUE, save optimization function in model for reporting
<code>...</code>	not currently used

Details

It is easiest to call `msjs_tmb` through the function `crm`. Details are explained there.

Value

The resulting value of the function is a list with the class of `crm,msjs` such that the generic functions `print` and `coef` can be used.

<code>beta</code>	named vector of parameter estimates
<code>lnl</code>	$-2 \times \log$ likelihood
<code>AIC</code>	$\ln l + 2 \times$ number of parameters
<code>convergence</code>	result from <code>optim</code> ; if 0 <code>optim</code> thinks it converged
<code>count</code>	<code>optim</code> results of number of function evaluations
<code>reals</code>	dataframe of data and real S and p estimates for each animal-occasion excluding those that occurred before release
<code>vcv</code>	var-cov matrix of betas if <code>hessian=TRUE</code> was set

Author(s)

Jeff Laake

msld_tmb

Fitting function for Multistate CJS live-dead models with TMB

Description

A function for computing MLEs for a Multi-state Cormack-Jolly-Seber open population capture-recapture with dead recoveries for processed dataframe `x` with user specified formulas in `parameters` that create list of design matrices `dml`. This function can be called directly but is most easily called from `crm` that sets up needed arguments.

Usage

```
msld_tmb(
  x,
  ddl,
  dml,
  model_data = NULL,
  parameters,
  accumulate = TRUE,
  initial = NULL,
  method,
  hessian = FALSE,
  debug = FALSE,
  chunk_size = 1e+07,
```

```

    refit,
    itnmax = NULL,
    control = NULL,
    scale,
    re = FALSE,
    extra.args = "",
    getreals = FALSE,
    useHess = FALSE,
    savef = TRUE,
    ...
)

```

Arguments

<code>x</code>	processed dataframe created by <code>process.data</code>
<code>ddl</code>	list of simplified dataframes for design data; created by call to <code>make.design.data</code>
<code>dml</code>	list of design matrices created by <code>create.dm</code> from formula and design data
<code>model_data</code>	a list of all the relevant data for fitting the model including <code>imat</code> , <code>S.dm</code> , <code>r.dm</code> , <code>p.dm</code> , <code>Psi.dm</code> , <code>S.fixed</code> , <code>r.fixed</code> , <code>p.fixed</code> and <code>time.intervals</code> . It is used to save values and avoid accumulation again if the model was re-rerun with an additional call to <code>cjs</code> when using <code>autoscale</code> or re-starting with initial values. It is stored with returned model object.
<code>parameters</code>	equivalent to <code>model.parameters</code> in <code>crm</code>
<code>accumulate</code>	if TRUE will accumulate capture histories with common value and with a common design matrix for all parameters speed up execution
<code>initial</code>	list of initial values for parameters if desired; if each is a named vector from previous run it will match to columns with same name
<code>method</code>	method to use for optimization; see <code>optim</code>
<code>hessian</code>	if TRUE will compute and return the hessian
<code>debug</code>	if TRUE will print out information for each iteration
<code>chunk_size</code>	specifies amount of memory to use in accumulating capture histories; amount used is $8 * \text{chunk_size} / 1e6$ MB (default 80MB)
<code>refit</code>	non-zero entry to refit
<code>itnmax</code>	maximum number of iterations
<code>control</code>	control string for optimization functions
<code>scale</code>	vector of scale values for parameters
<code>re</code>	if TRUE creates random effect model <code>admbcjsre.tpl</code> and runs <code>admb</code> optimizer
<code>extra.args</code>	optional character string that is passed to <code>tmb</code>
<code>getreals</code>	if TRUE, compute real values and std errors for TMB models; may want to set as FALSE until model selection is complete
<code>useHess</code>	if TRUE, the TMB hessian function is used for optimization; using hessian is typically slower with many parameters but can result in a better solution
<code>savef</code>	if TRUE, save optimization function in model for reporting
<code>...</code>	not currently used

Details

It is easiest to call `ms1d_tmb` through the function `crm`. Details are explained there.

Value

The resulting value of the function is a list with the class of `crm,cjs` such that the generic functions `print` and `coef` can be used.

<code>beta</code>	named vector of parameter estimates
<code>lnl</code>	$-2 \times \log$ likelihood
<code>AIC</code>	$\ln l + 2 \times$ number of parameters
<code>convergence</code>	result from <code>optim</code> ; if 0 <code>optim</code> thinks it converged
<code>count</code>	<code>optim</code> results of number of function evaluations
<code>reals</code>	dataframe of data and real S and p estimates for each animal-occasion excluding those that occurred before release
<code>vcv</code>	var-cov matrix of betas if <code>hessian=TRUE</code> was set

Author(s)

Jeff Laak

mstrata	<i>Multistrata example data</i>
---------	---------------------------------

Description

An example data set which appears to be simulated data that accompanies MARK as an example analysis using the Multistrata model.

Format

A data frame with 255 observations on the following 2 variables.

ch a character vector containing the encounter history of each bird with strata

freq the number of birds with that capture history

Details

This is a data set that accompanies program MARK as an example for the Multistrata model and is also in the RMark package. Here I use it to show the 3 ways models can be fitted to multistrata data. The model MSCJS is not run because it requires ADMB or the exe constructed from ADMB which is not available if downloaded from CRAN.

Examples

```

data(mstrata)
ms1=process.data(mstrata,model="MSCJS",strata.labels=c("A","B","C"))
ms1.ddl=make.design.data(ms1)
mod_mscjs=crm(ms1,ms1.ddl,model.parameters=list(Psi=list(formula=~-1+stratum:tostratum),
                                                  p=list(formula=~time)),hessian=TRUE,
                                                  save.matrices=TRUE)

mod_mscjs
mod_mscjsr=crm(ms1,ms1.ddl,model.parameters=list(Psi=list(formula=~-1+stratum:tostratum),
                                                         p=list(formula=~1+(1|time))),hessian=TRUE)

mod_mscjsr
# strata.labels for MVMS models must be specified as a list because more than one variable
# can be used
ms2=process.data(mstrata,model="MVMSJCJS",strata.labels=list(state=c("A","B","C")))
ms2.ddl=make.design.data(ms2)
ms2.ddl$delta$fix=1
# uses TMB with mvmscjs
mod_mvmscjs=crm(ms2,ms2.ddl,model.parameters=list(Psi=list(formula=~-1+stratum:tostratum),
                                                         p=list(formula=~time)),hessian=TRUE)

mod_mvmscjs

```

mvmscjs_tmb

TMB version: Fitting function for Multivariate Multistate CJS with uncertainty models

Description

A function for computing MLEs for MVMS models following Johnson et al (2015) via TMB. It works very much like mscjs but with more parameters. While this function is for fitting the model via ADMB, the documentation contained here is also relevant when this type of model is fitted with optimx and FORTRAN code (use.admb=FALSE, use.tmb=FALSE).

Usage

```

mvmscjs_tmb(
  x,
  ddl,
  fullddl,
  dml,
  model_data = NULL,
  parameters,
  accumulate = TRUE,
  initial = NULL,
  method,
  hessian = FALSE,
  debug = FALSE,
  chunk_size = 1e+07,

```

```

    refit,
    itnmax = NULL,
    control = NULL,
    re = FALSE,
    sup,
    getreals = FALSE,
    real.ids = NULL,
    useHess = FALSE,
    optimize = TRUE,
    vcv = FALSE,
    savef = TRUE,
    ...
)

```

Arguments

<code>x</code>	processed dataframe created by <code>process.data</code>
<code>ddl</code>	list of dataframes for design data; created by call to <code>make.design.data</code> and then simplified
<code>fullddl</code>	list of dataframes for design data prior to simplification
<code>dml</code>	list of design matrices created by <code>create.dm</code> from formula and design data
<code>model_data</code>	a list of all the relevant data for fitting the model including <code>imat</code> , <code>Phi.dm</code> , <code>p.dm</code> , <code>Psi.dm</code> , <code>Phi.fixed</code> , <code>p.fixed</code> , <code>Psi</code> , and <code>time.intervals</code> . It is stored with returned model object.
<code>parameters</code>	equivalent to <code>model.parameters</code> in <code>crm</code>
<code>accumulate</code>	if TRUE will accumulate capture histories with common value and with a common design matrix for S and p to speed up execution
<code>initial</code>	list of initial values for parameters if desired; if each is a named vector from previous run it will match to columns with same name
<code>method</code>	method to use for optimization; see <code>optim</code>
<code>hessian</code>	if TRUE will compute and return the hessian
<code>debug</code>	if TRUE will print out information for each iteration
<code>chunk_size</code>	specifies amount of memory to use in accumulating capture histories; amount used is $8 * \text{chunk_size} / 1e6$ MB (default 80MB)
<code>refit</code>	non-zero entry to refit
<code>itnmax</code>	maximum number of iterations
<code>control</code>	control string for optimization functions
<code>re</code>	currently ignored
<code>sup</code>	supplemental index values for constructing mvms model; these are defined in the <code>crm</code> code
<code>getreals</code>	if TRUE, compute real values and std errors for TMB models; may want to set as FALSE until model selection is complete
<code>real.ids</code>	vector of id values for which real parameters should be output with std error information

useHess	if TRUE, the TMB hessian function is used for optimization; using hessian is typically slower with many parameters but can result in a better solution
optimize	if TRUE, optimizes to get parameter estimates; set to FALSE to extract estimates of ADREPORTed values only
vcv	if TRUE, computes var-covariance matrix of ADREPORTed values
savef	if TRUE, save the makeAdFun result from TMB to report real values and matrices
...	not currently used

Details

It is easiest to call this function or any model built in marked through the function `crm`. This function has been exported but to fit a model it should be called through the `crm` function which does some testing of the arguments to avoid errors and sets up the calling arguments appropriately.

The `mvmscjs` model is quite flexible but also quite complex. Prior to using this model, read Johnson et al. (2015) and particularly Supplement B which goes through the code used in the analysis of Johnson et al. (2015). Supplement B is useful but there have been a number of changes to the code since the paper was written and it is far from complete. The documentation here will fill in some of those blanks.

There are 5 classes of parameters in an `mvms` model. They are 1)Phi - survival probability, 2) p-capture probability, 3) Psi - state transition probabilities, 4) delta - certainty/uncertainty state probabilities, 5) pi - initial state probability. The final class pi was described in Johnson et al. (2015) but was not implemented in marked at the time the paper was written. However, with the sealions example, the initial state is known for all releases and pi is not relevant.

Before I jump into a description of each parameter, I'll describe some characteristics that are common to all of the parameters. First, all of the parameters are probabilities and thus must be in the unit interval. To accomplish this each of the parameters uses either a logit or mlogit (multinomial logit) link function.

A logit link is $\log(\theta/(1-\theta)) = a + bx \dots$ where θ is the probability (called a real parameter in MARK terminology) and $a + bx \dots$ is the set of covariates (e.g. x) and working parameters (a, b) that are unbounded. Working parameters are called beta parameters in MARK terminology. The inverse logit link is $1/(1 + \exp(-a - bx))$ for this simple example and it can be computed from the `plogis` function in R (`plogis(a + bx)`). A largish negative value of the link (eg -5) mean θ will be nearly 0 and a largish positive value (+5) means θ is nearly 1.

The mlogit link is simply an extension of the logit link for more than 2 classes. For example, the logit link for survival is for the 2 classes alive/dead. I'll use the logit link as a simple example to describe the way mlogit links are formed in marked. If survival is $\Phi = 1/(1 + \exp(-a)) = \exp(a)/(1 + \exp(a))$ then mortality is $1 - \Phi = 1 - \exp(a)/(1 + \exp(a)) = 1/(1 + \exp(a))$. Imagine that we have 2 link values a - for survival and b - for mortality. Then we can rewrite the inverse logit function for Φ as $\exp(a)/(\exp(b) + \exp(a))$ and then $1 - \Phi = \exp(b)/(\exp(b) + \exp(a))$. It is obvious that $\Phi + 1 - \Phi = 1$ which must be the case. Now, if you actually tried to fit a model with both a and b the parameters a and b are not uniquely identifiable because there is only one parameter Φ . One of the values must be a reference cell which means the value is fixed. Typically we assume $b = 0$ which yields $\exp(0) = 1$ and we get the logit link.

Now consider 3 classes (A,B,C) and we'll use the link values a, b, c for a generic parameter θ . As with the 2 class example, the probabilities are $\theta(A) = \exp(a)/(\exp(a) + \exp(b) + \exp(c))$, $\theta(B) = \exp(b)/(\exp(a) + \exp(b) + \exp(c))$

and $\theta(C) = \exp(c) / (\exp(a) + \exp(b) + \exp(c))$. As before $\theta(A) + \theta(B) + \theta(C) = 1$ but for a multinomial with 3 classes there are only 2 free estimable parameters due to the constraint that they have to sum to 1. Just as with the example above, a reference cell must be selected. If we chose B as the reference cell then $b=0$ and the probabilities become $\theta(A) = \exp(a) / (\exp(a) + 1 + \exp(c))$, $\theta(B) = 1 / (\exp(a) + 1 + \exp(c))$ and $\theta(C) = \exp(c) / (\exp(a) + 1 + \exp(c))$. This extends to an k -class multinomial where $k-1$ parameters are freely estimable and 1 must be fixed.

For parameters with logit links there is an obvious choice for the reference cell. For example, for survival probability we are interested in describing the parameters that affect survival, so we use death as the reference cell which is computed by subtraction ($1-\Phi$). Likewise, for capture (sighting) probability we use not caught ($1-p$) as the reference cell. For mlogit links, there is not always an obvious choice for a reference cell and it can be an advantageous to allow the reference cell to be flexible and user-defined.

To understand how to set the reference cell, you should review the material in the help file for `create.dmdf` and Supplement B of Johnson et al. (2015) which describes the design data but I'll give a brief description here. All model building in the `marked` package revolves around the design data. If the design data are incorrect the model will be incorrect. Briefly, the design data contain the data for the parameter model, providing the maximum structure from which specific models can be built. For example, for p there are $M-1$ records for each of the I individuals and M occasions (note that if the data are accumulated then an "individual" is actually a unique set of capture history and covariate data that represents "freq" animals). Thus, capture probability could be allowed to differ for each occasion, individual, age, strata, or any combination thereof. In practice, such a model would be overparameterized, but this structure allows maximum flexibility; users still need to use formula to provide specific constraints to reduce the number of parameters to a reasonable level. With this design data approach essentially there is a place holder for each possible p and any or all of those values can be fixed to a real value.

"Real" parameters can be fixed by adding a field called `fix` to the design data for a parameter. The value of `fix` should be either NA if the parameter is to be estimated or it should be the value of the real parameter (inverse link value). For example, we could fix the capture probability for time 2 to be 0 for all individuals at time 2 or at time 2 in strata B because there was no capture effort. Or you could fix $p=0$ for an individual at a time because it was temporarily removed. Or you could fix $p=1$ for a set of individuals that were always recaptured via telemetry.

Now let's go back to the mlogit parameters. There is an important difference between the way `marked` and `RMark/MARK` work with regard to mlogit parameters like Ψ for state transition probabilities. An mlogit parameter is one in which the sum of the probabilities is 1. For Ψ , if I'm in stratum A and can go to B or C or remain in A, the probabilities A to A, A to B and A to C must sum to 1 because that is all of the possibilities. In `RMark/MARK` the design data would only contain 2 records which are determined based on what you select as `subtract.stratum`. If `subtract.stratum` was set as A for the A stratum, the design data for Ψ would only contain records for A to B and A to C. The value for A to A would be computed by subtraction and is the reference cell.

By contrast, in `marked`, all 3 records are in the design data and to set the transition from A to A as the reference cell (computed by subtraction) the value of `fix` is set to 1. So if `fix` is the inverse logit value why is it being set to 1 for the mlogit? Because the link for these parameters is actually a log-link and the inverse link is $\exp(\text{link})$. So the link value is fixed to 0 and the inverse link ("real value") is $\exp(0)=1$. The mlogit parameter is constructed by dividing each $\exp()$ value by the sum of the $\exp()$ values across the set defining the possible classes for the multinomial. Following along with the example,

```

Psi_AtoB=exp(beta_AtoB)/(1+exp(beta_AtoB)+exp(beta_AtoC))
Psi_AtoC=exp(beta_AtoC)/(1+exp(beta_AtoB)+exp(beta_AtoC)) and
Psi_AtoA=1-Psi_AtoB-Psi_AtoC = 1/(1+exp(beta_AtoB)+exp(beta_AtoC)).

```

The "appropriate" set will depend on the mlogit parameter and the data/model structure.

I used this structure for several reasons. First, you get a real parameter estimate for the subtracted stratum which you don't get in RMark/MARK. Secondly, you can change the value to be subtracted at will and it is not fixed across the entire model fit, but you do have to be careful when specifying the model when you do that because the formula specifies the parameters for those that are not fixed. Finally, you can change the reference cell simply by modifying the value of fix, whereas with the subtract.stratum approach the design data would have to be recreated which can take some time with large data sets.

Next I'll briefly describe each parameter in the model and its link function structure.

Phi — Phi is survival probability in the model. Survival probability is also called S in the straight multi-state model. In retrospect I wish I had stayed with S but I didn't. In capture-recapture Phi is usually reserved for apparent survival rate which might include permanent emigration. With a multi-state model, the presumption is that "all" states are covered and animals can't emigrate permanently to some unobservable state. Whether that is true or not is always a question. Anyhow I use Phi for survival in mvmscjs models.

Phi uses a logit link ($\log(\text{Phi}/(1-\text{Phi})) = a + bx \dots$) to relate survival to any covariates which have parameter values across the entire real line. Phi is what I call an interval parameter in that it describes the probability of surviving during an interval of time between sampling occasions. The design data for interval parameters are describe in terms of the time and age of the animal at the beginning of the interval. For example, if occasions are at time 1,2,...J there will be J-1 data records for each individual with time values 1,...,J-1.

p — p is recapture/resighting probability. While I'll often fall back and call it capture or sighting probability, technically it is recapture or resighting because this model is based on a Cormack-Jolly-Seber (CJS) formulation which does not model initial "capture" probability and only models events after they are marked and released. p also uses the logit link and it is an occasion parameter because it describes an event associated with an occasion. Because they are "recaptures", there are J-1 design data records for each individual at times 2,...,J.

An example of using fix for p is given in the sealions example. For the data, there was no survey effort in stratum A for the last occasion.

```

ddl$p$fix = ifelse(ddl$p$Time==17 & ddl$p$area=="A", 0, ddl$p$fix)

```

Psi — Psi describes the transition probabilities of moving between states. For each individual at each occasion for each of the possible states, it is formulated as a multinomial with a cell probability for movement from the state to all of the other possible states and remaining in the in the same state. Thus if there are M states, then there are M*M records for each individual for each of the J-1 intervals. It is treated as interval parameter with movement occurring between occasions but after survival. If an animal is in state A it survives and moves to state B with probability $\text{Phi}_A * \text{Psi}_{AtoB}$. If those were to be reversed you could use a trick proposed by Bill Kendall of adding dummy occasions between each real set of occasions and fixing real parameters. For example using the first 2 occasions with data A0, the data would become A00 with 3 occasions. For the first interval all the Phis would be fixed to 1 and Psi_{AtoB} would be estimated. For occasion 2, all

the ps would be fixed to 0 because there are no observations and for the second interval all the Psi's should be fixed so it will remain in the same state (eg. Psi_AtoA=1 and Psi_Atox=0).

This can get rather involved when you have more than 1 variable describing the states which is the whole purpose of the multi-variate multi-state model. It just needs some careful thought. As an example, here are some of the columns of the design data for the first interval for the first individual for stratum A- which for this example is area A with missing left and right tags. There are 8 records because there are 8 possible strata - 2 areas *2 left tag states * 2 right tag states.

```
> head(ddl$Psi[,c(1:10,20),],8)
  id occ stratum tostratum area ltag rtag toarea toltag tortag fix
1  1  1  A--      A--      A   -   -      A     -     -     1
1  1  1  A--      A--      A   -   -      A     -     +     0
1  1  1  A--      A--      A   -   -      A     +     -     0
1  1  1  A--      A--      A   -   -      A     +     +     0
1  1  1  A--      S--      A   -   -      S     -     -    NA
1  1  1  A--      S--      A   -   -      S     -     +     0
1  1  1  A--      S--      A   -   -      S     +     -     0
1  1  1  A--      S--      A   -   -      S     +     +     0
```

When the design data are created, it creates fields based on the definition of strata.labels in the call to process.data. For this example it was strata.labels=list(area=c("A","S"),ltag=c("+","-","u"),rtag=c("+","-","u")). The design data include a stratum and tostratum field which are the concatenated values of all the variables defining the states. In addition field and tofield variables are created for each strata variables. Here the fields are area,ltag,rtag and the tofields are toarea, toltag and tortag. Each simply partitions the stratum and tostratum fields. Note that if only a single variable is used to define the strata.labels the the design data will only contain a stratum and tostratum fields.

Now, when the design data are initially created for Psi, it creates the field fix and assigns the value NA to all values except for the record in which stratum=tostratum which gets the value of fix=1 making it the reference cell. Those are the default values and it is up to you to change as needed. In this case almost all of the Psi values are fixed to 0 because they cannot occur. The only possible movement is from A- to S- because all of the other transitions would have it gaining tags that have already been lost. As another example, the stratum A++ would have all of the values of fix=NA except for tostratum=A++ which would be fixed to 1.

delta — delta is the probability of being certain or uncertain about each state variable for a given stratum. The definition can change based on how you define the reference cell. For delta, the number of records depends on the structure of strata.labels and the number of state variables and how many state variables can be uncertain. Below are the design data for a single stratum for a single individual and occasion from the sealions example.

```
id occ stratum area ltag rtag obs.area obs.ltag obs.rtag obs.ltag.u obs.rtag.u
1  1  1  A--      A   -   -      A     -     -     0     0
1  1  1  A--      A   -   -      A     -     u     0     1
1  1  1  A--      A   -   -      A     u     -     1     0
1  1  1  A--      A   -   -      A     u     u     1     1
```

For this example, the area is certain but the left tag and right tag status can be uncertain. The above data are for the stratum A-. Each field used to define the states has a field value and an obs.field

value (eg., ltag, obs.ltag). The fields obs.ltag.u and obs.rtag.u were created in code after the call to make.design.data (see ?sealions). There would be a set like these for each of the 8 possible stratum value for id=1 and occ=1. Because each tag status can be either known or uncertain there are 2*2 records and the sum of the 4 cell probabilities must sum to 1. The easiest way to do that is by specifying fix=1 for one of the cells and assigning the other 3 fix=NA so they are estimated. If you set fix=1 for the first record (observation A-), then the parameters for delta are describing the probability of being uncertain. Whereas, if you set fix=1 for the fourth record (observation Auu) then the parameters for delta are describing the probability of being certain.

Now the use of fix is not required to set a reference cell because you can accomplish the same thing by defining the formula for delta such that the link value is 0 for one of the cells. For example, the formula for delta in the example is

```
~ -1 + obs.ltag.u + obs.rtag.u + obs.ltag.u:obs.rtag.u)
```

The numeric value of obs.ltag.u + obs.rtag.u + obs.ltag.u*obs.rtag.u (: equivalent to multiplication here) is 0 for the first record and -1 in the formula specifies that there shouldn't be an intercept. Thus regardless of the 3 beta values for delta, the link value for the first record is 0 and the inverse link value is exp(0)=1 which is equivalent to setting fix=1 for the first record to set it as the reference cell. If you run the sealions example with the current version of the software it will give the message:

```
No values provided for fix for delta. At least need to set a reference cell
```

Messages like this are warnings and do not stop the model from being fitted because as shown above it is possible to set the reference cell with an appropriate formula. If a reference cell is not set the beta estimates are not all identifiable. Symptoms include troubles with optimization, different starting values giving different beta estimates for the parameter with the same likelihood value, and most easily seen is large standard errors for all of the beta estimates involved.

pi — pi is the probability of being in a state for the initial occasion. It is only used when there is one or more individual capture histories have an uncertain (u) state variable value for the initial occasion of the capture history. For the sealions example, all are released in the state S++ so there was no uncertainty.

The design data for pi includes a record for each possible stratum for each individual. For the sealions example, the following are the design data for the first individual:

```
id occ stratum area ltag rtag fix
1 1 A-- A - - 0
1 1 A-+ A - + 0
1 1 A+- A + - 0
1 1 A++ A + + 0
1 1 S-- S - - 0
1 1 S-+ S - + 0
1 1 S+- S + - 0
1 1 S++ S + + 1
```

When the design data are created the code recognizes that the initial value for this individual capture history is known for all of the strata values so it assigns 0 to the strata that are not the initial value and 1 for the strata that is the initial value.

To demonstrate what is obtained when there is uncertainty I modified one of the S++ values to Suu and this is the resulting design data

id	occ	stratum	area	ltag	rtag	fix
335	1	A--	A	-	-	0
335	1	A-+	A	-	+	0
335	1	A+-	A	+	-	0
335	1	A++	A	+	+	0
335	1	S--	S	-	-	1
335	1	S-+	S	-	+	NA
335	1	S+-	S	+	-	NA
335	1	S++	S	+	+	NA

All of the fix values for area A are still 0 because area is certain and it is in area S. However, each tag status is uncertain so there are 4 possible true values. S- is set to the reference cell by default and the other 3 would be estimated.

There is a subtlety to the definition of pi that should be understood. The structure of the default design data for pi means that pi is the conditional probability of being in a state given there is uncertainty and that an animal is first observed in that period. In other words, it is the probability of being in states just for the records that have uncertainty and are first observed at a given time. For the sealions, consider having released 1/4 of the animals in A with all certain tag status and for the 3/4 released in S neither tag status was known. In that case pi would only relate to S and the proportion given both tags will possibly be less in S than in both areas combined. However, if you defined fix for all records as above the estimated pi would apply to the whole set.

The mvmscjs model is constructed as a hidden Markov model as described in Johnson et al. (2015). As such the 5 parameter classes (Phi,p,Psi,delta and pi) are used to construct the matrices gamma (eq 2.13 in Johnson et al. (2015)), dmat (eq 2.14 in Johnson et al. (2015)) and delta which is called pi in Johnson et al. (2015). In the code I used the name delta for the matrix which has I rows and M columns because the symbol delta is used in Zucchini et al's book on HMMs. So think of delta as the matrix and pi as the values in each row of the matrix. I should have used pi as the matrix as well to avoid confusion between the delta parameters and the matrix delta for initial values.

One other difference between the coding and Johnson et al. (2015) is that the 0 observation in dmat is the last row in Johnson et al (2015) and it is the first row in the code. Likewise, for gamma the death state is the last row in Johnson et al. (2015) and the first row in the code. In Johnson et al. (2015) dmat and gamma are matrices but in the code they are 4-d arrays where the first 2 dimensions are individuals and occasions and the last 2 dimensions are the matrices defined in Johnson et al. (2015).

Johnson et al. (2015) show delta being used for each occasion including the initial release occasion. In fact, design data for delta are created for each occasion. But in constructing the formula there is one exception that you need to understand. If the initial values are either all certain or all uncertain then formula for delta is not used for that initial release because all values of dmat are either 0 or 1.

If these matrices are setup incorrectly the optimization will likely fail. In general, that is all handled by the code but it is possible to mess up the values in those matrices if the values of fix are incorrectly defined. One place where this can occur is if the value of fix=0 for all records defining a multinomial because when it normalizes to sum to 1, it will produce a NaN due to 0/0. To prevent problems the function HMMLikelihood with the argument check=TRUE is run before attempting the optimization. With the starting parameter values it checks to make sure that the columns of dmat sum to 1 and the rows of gamma sum to 1 and all values of pi in delta sum to 1. If any of these fail outside of rounding error, the program will issue errors showing the problematic records and values and will stop.

Unobservable states can be included in the data structure and model but you must be careful in constructing formulas. Keep in mind that there are no observations for unobservable states. That states the obvious but it can be easily forgotten when constructing formulas. For an easy example, consider a univariate example with area defining states. Will assume that we have areas A,B and C that are observable and X which is unobservable. For Psi you can have transitions to and from the X and the other states. Obviously, you'll need to set $\text{fix}=0$ for p for X because it can't be seen. Also, if you were to set the formula for Phi to be $\sim\text{stratum}$ there are no data in X so it will not be able to estimate survival in X. So don't do that. Likewise, the unobservable state value X should not be in a formula for delta because it represents certainty/uncertainty of state variables for observations and there are no observations in X. Also, because this is a CJS formulation that is conditional on the first release (observation), the formula for pi cannot include X, again because there are no observations.

To avoid including parameters for levels of a factor for unobservable state values, the easiest approach is to set fix values for the design data rows for the unobservable states. For example, if $\text{fix}=0$ for p for stratum X, then $\sim\text{stratum}$ could be used in the formula and it would have an intercept and parameters for B and C. The stratum level X would be excluded because all rows of the resulting design matrix with fixed values are set to 0 and then any columns which have all 0 values are dropped which would drop the X factor level. Another approach is to define a different variable that excludes the factor level. Note that this approach would not work to set $p=0$ for the unobservable state. Assume in addition to area we had another variable to define states and it could be uncertain. If we thought that the uncertainty in that variable was related to area, we could use a formula for delta like $\sim-1+\text{notX}:\text{area}$ where notX was a variable with value 1 when the area was not X and was 0 when area was X. This would result in parameters for A,B and C but not X. A similar approach would be to create 0/1 dummy variables A,B,C which were 1 when the record was for $\text{area}=A,B,C$ respectively and 0 otherwise. Then you could use the delta formula $\sim-1+A+B+C$ specifically excluding X. It is not important for X to have a value for delta because there are no observations for X and if you look at the last row (observation =0) in eq 2.14 in Johnson et al. (2015) you'll see it is $1-p$ and doesn't use delta.

marked doesn't include any specific robust design models because they can be relatively easily handled by fixing real parameters. In a standard closed multi-state robust design, during the secondary periods $\text{Phi}=1$ and $\text{Psi_AtoA}=1$ for all states (no transitions). Any variation can be created simply by fixing the appropriate parameters. For example, with the sealions example, you might allow transitions between areas but assume no transitions for tag status.

Value

The resulting value of the function is a list with the class of `crm,cjs` such that the generic functions `print` and `coef` can be used.

<code>beta</code>	named vector of parameter estimates
<code>lnl</code>	$-2 \times \log$ likelihood
<code>AIC</code>	$\ln l + 2 \times$ number of parameters
<code>convergence</code>	result from <code>optim</code> ; if 0 <code>optim</code> thinks it converged
<code>count</code>	<code>optim</code> results of number of function evaluations
<code>reals</code>	dataframe of data and real S and p estimates for each animal-occasion excluding those that occurred before release
<code>vcv</code>	var-cov matrix of betas if <code>hessian=TRUE</code> was set

Author(s)

Jeff Laake

References

Johnson, D. S., J. L. Laake, S. R. Melin, and DeLong, R.L. 2015. Multivariate State Hidden Markov Models for Mark-Recapture Data. 31:233-244.

mvms_design_data	<i>Multivariate Multistate (mvms) Design Data</i>
------------------	---

Description

Creates a dataframe with design data for MvMS model for a single occasion

Usage

```
mvms_design_data(df.states, df = NULL, transition = TRUE)
```

Arguments

df.states	dataframe of states created from set_mvms
df	is dataframe of observations; provided for parameter delta in which both state and observation are needed in design data
transition	if TRUE, creates design data for a state transition (from to); otherwise just state variables

Value

a dataframe to be used with design data for mvms model

Author(s)

Jeff Laake

Examples

```
x=set_mvms(list(location=c("A","B","C"),repro_status=c("N","P","u")))
mvms_design_data(x$df.states)
mvms_design_data(x$df.states,transition=FALSE)
```

 omega

Compute 1 to k-step transition proportions

Description

Computes 1 to k-step forward transition proportions in each state with a single transition matrix or a 3-d array of transition matrices.

Usage

```
omega(x, k = NULL, labels = NULL)
```

Arguments

x either a transition matrix, list of transition matrices or 3-d array (a set of transition matrices)

k if x is a transition matrix, this is number of steps 1 to k

labels labels for states except for last which is always dead and is added at end

Author(s)

Jeff Laake

Paradise_shelduck

Multistate Live-Dead Paradise Shelduck Data

Description

Paradise shelduck recapture and recovery data in multistrata provided by Richard Barker and Gary White.

Format

A data frame with 1704 observations of 3 variables

ch a character vector containing the capture history (each is 2 character positions LD) for 6 occasions

freq capture history frequency

sex Male or Female

Author(s)

Jeff Laake

References

Barker, R.J, White,G.C, and M. McDougall. 2005. MOVEMENT OF PARADISE SHELDUCK BETWEEN MOLT SITES: A JOINT MULTISTATE-DEAD RECOVERY MARK-RECAPTURE MODEL. JOURNAL OF WILDLIFE MANAGEMENT 69(3):1194-1201.

Examples

```
# In the referenced article, there are 3 observable strata (A,B,C) and 3 unobservable
# strata (D,E,F). This example is setup by default to use only the 3 observable strata
# to avoid problems with multiple modes in the likelihood.
# Code that uses all 6 strata are provided but commented out.
# With unobservable strata, simulated annealing should
# be used (options="SIMANNEAL")
data("Paradise_shelduck")
# change sex reference level to Male to match design matrix used in MARK
ps$sex=relevel(ps$sex,"Male")
# Process data with MSLiveDead model using sex groups and specify only observable strata
ps_dp=process.data(ps,model="MSLD",groups="sex",strata.labels=c("A","B","C"))
# Process data with MSLiveDead model using sex groups and specify observable and
# unobservable strata
# ps_dp=process.data(ps,model="MSLD",groups="sex",strata.labels=c("A","B","C","D","E","F"))
# Make design data and specify constant PIM for Psi to reduce parameter space. No time
# variation was allowed in Psi in the article.
ddl=make.design.data(ps_dp)
# Fix p to 0 for unobservable strata (only needed if they are included)
ddl$p$fix=NA
ddl$p$fix[ddl$p$stratum%in%c("D","E","F")]=0
# Fix p to 0 for last occasion
ddl$p$fix[ddl$p$time%in%6:7]=0.0
# Fix survival to 0.5 for last interval to match MARK file (to avoid confounding)
ddl$s$fix=NA
ddl$s$fix[ddl$s$time==6]=0.5
# create site variable for survival which matches A with D, B with E and C with F
ddl$s$site="A"
ddl$s$site[ddl$s$stratum%in%c("B","C")]=as.character(ddl$s$stratum[ddl$s$stratum%in%c("B","C")])
ddl$s$site[ddl$s$stratum%in%c("E")]="B"
ddl$s$site[ddl$s$stratum%in%c("F")]="C"
ddl$s$site=as.factor(ddl$s$site)
# create same site variable for recovery probability (r)
ddl$r$site="A"
ddl$r$site[ddl$r$stratum%in%c("B","C")]=as.character(ddl$r$stratum[ddl$r$stratum%in%c("B","C")])
ddl$r$site[ddl$r$stratum%in%c("E")]="B"
ddl$r$site[ddl$r$stratum%in%c("F")]="C"
ddl$r$site=as.factor(ddl$r$site)
# Specify formula used in MARK model
S.1=list(formula=~-1+sex+time+site)
p.1=list(formula=~-1+stratum:time)
r.1=list(formula=~-1+time+sex+site)
Psi.1=list(formula=~-1+stratum:tostratum)
# Run top model from paper but only for observable strata
mod_msld=crm(ps_dp,ddl,model.parameters=list(S=S.1,p=p.1,r=r.1,Psi=Psi.1),
method="nlminb",hessian=TRUE,save.matrices=TRUE)
```

```
# Run top model from paper for all strata using simulated annealing (commented out)
# mod_msld=crm(ps_dp,ddl,model.parameters=list(S=S.1,p=p.1,r=r.1,Psi=Psi.1),
#           method="SANN",itnmax=6e6,hessian=TRUE)
```

Phi.mean *Various utility parameter summary functions*

Description

Several functions have been added to compute mean values and boxplots of values. Currently they have only been defined for Phi and p for cjs and they are not generic.

Usage

```
Phi.mean(x,age=0,time=NULL,age.bins=NULL,age.levels=NULL)
```

```
p.mean(x,age=0,time=NULL,age.bins=NULL,age.levels=NULL)
```

```
Phi.boxplot(x,age=0,time=NULL,sex=NULL)
```

```
p.boxplot(x,age=0,time=NULL,sex=NULL)
```

Arguments

x	dataframe of reals contained in model
age	at which Phi or p should be shown across time
time	at which Phi or p should be shown across ages
age.bins	bins for age in which values are summarized
age.levels	labels for age.bins
sex	for which Phi or p should be shown across ages

Value

matrix of labelled values for Phi.mean and p.mean or boxplot object

Author(s)

Jeff Laake

predict.crm

*Compute estimates of real parameters***Description**

Computes real estimates and their var-cov for a particular subset of parameters. You can compute real estimates for a subset of values or a new set of values is to specify a limited range of the values in ddl for each parameter. Make sure to include a complete set of values that spans the factor levels and individual covariates used in the formulas for the model object or you will receive an error that the number of columns in the design matrix does not match the number of beta parameters. You cannot change the levels of any factor variable or modify the design data in anyway that changes the design matrix.

Usage

```
## S3 method for class 'crm'
predict(object,ddl=NULL,parameter=NULL,unique=TRUE,
        vcv=FALSE,se=FALSE,chat=1,subset=NULL,select=NULL,
        real.ids=NULL,merge=FALSE,unit_scale=TRUE,...)
```

Arguments

object	model object;
ddl	list of dataframes for design data (not simplified)
parameter	name of real parameter to be computed (eg "Phi")
unique	TRUE if only unique values should be returned; if TRUE forces computation even if estimates are in model object
vcv	logical; if TRUE, computes and returns v-c matrix of real estimates; if TRUE forces computation even if estimates are in model object
se	logical; if TRUE, computes std errors and conf intervals of real estimates
chat	over-dispersion value
subset	logical expression using fields in real dataframe
select	character vector of field names in real that you want to include
real.ids	animal ids passed to TMB code for computation of real parameter values
merge	default FALSE but if TRUE, the ddl for the parameter is merged (cbind) to the estimates
unit_scale	default TRUE, if FALSE any time scaled parameter (e.g. Phi,S) is scaled when computing real value such that it represents the length of the interval rather than a unit interval
simplify	if TRUE, simplifies ddl to unique values prior to calculation; not valid with random effects
show.fixed	if TRUE, returns fixed parameter values as well as estimated values
...	generic arguments not used here

Details

If the real estimates are in the model object and se and vcv are FALSE and ddl not specified, the code will simply pull the values from the model object.

Value

A data frame (real) is returned if vcv=FALSE; otherwise, a list is returned also containing vcv.real:

real	data frame containing estimates, and if vcv=TRUE it also contains standard errors and confidence intervals
vcv.real	variance-covariance matrix of real estimates

Author(s)

Jeff Laake

print.crm

Print model results

Description

Provides a printed simple summary of the model results.

Usage

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

Arguments

x	crm model result or list of model results
...	generic arguments not used here

Value

prints a simple summary of the model to the screen and returns NULL.

Author(s)

Jeff Laake

See Also

[crm](#)

print.crmlist	<i>Print model table from model list</i>
---------------	--

Description

Print model table from model list

Usage

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

Arguments

x	list of model results
...	generic arguments not used here

Value

None

Author(s)

Jeff Laake

See Also

[crm](#)

proc.form	<i>Mixed effect model formula parser Parses a mixed effect model in the lme4 structure of ~fixed +(re1 g1) +...+(ren gn)</i>
-----------	--

Description

Mixed effect model formula parser

Parses a mixed effect model in the lme4 structure of ~fixed +(re1|g1) +...+(ren|gn)

Usage

```
proc.form(f)
```

Arguments

f	formula for mixed effect mode in the form used in lme4; ~fixed +(re1 g1) +...+(ren gn)
---	--

Value

A list with elements `fix.model` and `re.model`. `fix.model` contains the formula for the fixed effects; `re.model` contains elements `sub`, the grouping formula and `model` the design formula for the random effect. Each formula is of type character and must be wrapped with `as.formula` in use with `model.matrix`

Author(s)

Devin Johnson <devin.johnson@noaa.gov>

process.ch

Process release-recapture history data

Description

Creates needed constructs from the release-recapture history.

Usage

```
process.ch(ch, freq = NULL, all = FALSE)
```

Arguments

<code>ch</code>	Vector of character strings; each character string is composed of either a constant length sequence of single characters (01001) or the character string can be comma separated if more than a single character is used (1S,1W,0,2W). If comma separated, each string must contain a constant number of elements.
<code>freq</code>	Optional vector of frequencies for <code>ch</code> ; if missing assumed to be <code>a</code> ; if <code><0</code> indicates a loss on capture
<code>all</code>	<code>FALSE</code> is okay for <code>cjs</code> unless R code used to compute <code>lnl</code> instead of FORTRAN; must be <code>true</code> for <code>js</code> because it returns additional quantities needed for entry prob.

Value

<code>nocc</code>	number of capture occasions
<code>freq</code>	absolute value of frequency for each <code>ch</code>
<code>first</code>	vector of occasion numbers for first 1
<code>last</code>	vector of occasion numbers for last 1
<code>loc</code>	vector of indicators of a loss on capture if set to 1
<code>chmat</code>	capture history matrix
<code>FtoL</code>	1's from first (1) to last (1) and 0's elsewhere; only if <code>all==TRUE</code>
<code>Fplus</code>	1's from occasion after first (1) to <code>nocc</code> (last occasion); only if <code>all==TRUE</code>
<code>Lplus</code>	1's from occasion after last (1) to <code>nocc</code> ; only if <code>all==TRUE</code>
<code>L</code>	1's from last (1) to <code>nocc</code> ; only if <code>all==TRUE</code>

Author(s)

Jeff Laake

process.data

*Process encounter history dataframe for MARK analysis***Description**

Prior to analyzing the data, this function initializes several variables (e.g., number of capture occasions, time intervals) that are often specific to the capture-recapture model being fitted to the data. It also is used to 1) define groups in the data that represent different levels of one or more strata.labels factor covariates (e.g., sex), 2) define time intervals between capture occasions (if not 1), and 3) create an age structure for the data, if any.

Usage

```
process.data(data,begin.time=1,model="CJS",mixtures=1,groups=NULL,
             allgroups=FALSE,age.var=NULL,initial.ages=c(0),
             time.intervals=NULL,nocc=NULL,accumulate=TRUE,
             strata.labels=NULL)
```

```
accumulate_data(data)
```

Arguments

data	A data frame with at least one field named ch which is the capture (encounter) history stored as a character string. data can also have a field freq which is the number of animals with that capture history. The default structure is freq=1 and it need not be included in the dataframe. data can also contain an arbitrary number of covariates specific to animals with that capture history.
begin.time	Time of first capture occasion or vector of times if different for each group
model	Type of analysis model.
mixtures	Number of mixtures in closed capture models with heterogeneity
groups	Vector of factor variable names (in double quotes) in data that will be used to create groups in the data. A group is created for each unique combination of the levels of the factor variables in the list.
allgroups	Logical variable; if TRUE, all groups are created from factors defined in groups even if there are no observations in the group
age.var	An index in vector groups for a variable (if any) for age
initial.ages	A vector of initial ages that contains a value for each level of the age variable groups[age.var]; if the data contain an initial.age field then it will be used instead.
time.intervals	Vector of lengths of time between capture occasions or matrix of time intervals with a row for each animal and column for each interval between occasions.

nocc	number of occasions for Nest type; either nocc or time.intervals must be specified
accumulate	if TRUE, aggregates data with same values and creates freq field for count of records
strata.labels	labels for strata used in capture history; they are converted to numeric in the order listed. Only needed to specify unobserved strata; for any unobserved strata p=0.

Details

For examples of data, see [dipper](#). The structure of the encounter history and the analysis depends on the analysis model to some extent. Thus, it is necessary to process a dataframe with the encounter history (ch) and a chosen model to define the relevant values. For example, number of capture occasions (nocc) is automatically computed based on the length of the encounter history (ch) in data. The default time interval is unit time (1) and if this is adequate, the function will assign the appropriate length. A processed data frame can only be analyzed using the model that was specified. The model value is used by the functions [make.design.data](#) and [crm](#) to define the model structure as it relates to the data. Thus, if the data are going to be analysed with different underlying models, create different processed data sets with the model name as an extension. For example, `dipper.cjs=process.data(dipper)`.

This function will report inconsistencies in the lengths of the capture history values and when invalid entries are given in the capture history.

The argument `begin.time` specifies the time for the first capture occasion and not the first time the particular animal was caught or released. This is used in creating the levels of the time factor variable in the design data and for labelling parameters. If the `begin.time` varies by group, enter a vector of times with one for each group. It will add a field `begin.time` to the data with the value for each individual. You can also specify a `begin.time` field in the data allowing each animal to have a unique `begin.time`. Note that the time values for survivals are based on the beginning of the survival interval and capture probabilities are labeled based on the time of the capture occasion. Likewise, age labels for survival are the ages at the beginning times of the intervals and for capture probabilities it is the age at the time of capture/recapture.

The `time.intervals` argument can either be a vector of lengths of times for each interval between occasions that is constant for all animals or a matrix which has a row for each animal and a column for each interval which lets the intervals vary by animals. These intervals are used to construct the design data and are used for the field `time.interval` which is used to adjust parameters like Φ and S to a constant per unit time interval (eg annual survival rates). On occasion it can be useful to leave the `time.interval` to remain at default of 1 or some other vector of `time.intervals` to construct the design data and then modify the `time.interval` value in the design data. For example, assume that cohort marking and release is done between sampling occasions. The initial survival from release to the next sampling occasion may vary by release cohort, but the remainder of the survivals are between sampling occasions. In that case it is easier to let `time.interval=1` (assuming unit interval (eg year) between sampling occasions but then modifying `ddlPhitime.interval` to the value for the first interval after each release to be the partial year from release to next sampling occasion. In this way everything is labelled with annual quantities but the first partial year survival is adjusted to an annual rate.

Note that if you specify `time.intervals` as a matrix, then `accumulate` is set to FALSE so that the number of rows in the data can be checked against the number of rows in the `time.intervals` matrix

and thus data cannot be accumulated because at present it doesn't use values of `time.intervals` to determine which records can be accumulated.

`groups` is a vector of variable names that are contained in data. Each must be a factor variable. A group is created for each unique combination of the levels of the factor variables. For example `groups=c("sex", "ageclass", "region")` creates groups defined by the levels of sex, ageclass and region. the code will only use groups that have 1 or more capture histories unless `allgroups=TRUE`.

The argument `age.var=2` specifies that the second grouping variable in `groups` represents an age variable. It could have been named something different than `ageclass` but it should not be named `age` as that is reserved in `marked`. `initial.age` specifies that the age at first capture of the age levels. For example `initial.age=0:2` specifies that the initial ages are 0,1 and 2 for the age class levels designated as 1,2,3. The actual ages for the age classes do not have to be sequential or ordered, but ordering will cause less confusion. Thus levels 1,2,3 could represent initial ages of 0,4,6 or 6,0,4. The default for `initial.age` is 0 for each group, in which case, `age` represents time since marking (first capture) rather than the actual age of the animal. If the data contains an `initial.age` field then it overrides any other values and lets each animal have a unique `initial.age` at first capture/release.

The following variable names are reserved and should not be used in the data: `id` (animal id) `ch`(capture history) `freq` (number of animals with that `ch`/data) `occ`,`age`,`time`,`cohort`,`Age`,`Time`,`Cohort`,`time.interval`,`fix`

Value

from `process.data` `processed.data` (a list with the following elements)

<code>data</code>	original raw dataframe with group factor variable added if groups were defined
<code>model</code>	type of analysis model (eg, "cjs" or "js")
<code>freq</code>	a dataframe of frequencies (same # of rows as <code>data</code> , number of columns is the number of groups in the data. The column names are the group labels representing the unique groups that have one or more capture histories.
<code>nocc</code>	number of capture occasions
<code>time.intervals</code>	length of time intervals between capture occasions
<code>begin.time</code>	time of first capture occasion
<code>initial.ages</code>	an initial age for each group in the data; Note that this is not the original argument but is a vector with the initial age for each group.
<code>group.covariates</code>	factor covariates used to define groups

from `accumulate_data` a dataframe with same column structure as argument with addition of `freq` (if not any) and reduced to unique rows with `freq` accumulating number of records.

Author(s)

Jeff Laake

See Also

[dipper,crm](#)

Examples

```

data(dipper)
dipper.process=process.data(dipper,groups="sex",strata.labels=c("1","2"))
# create some artificial age data as an example
dipper$ageclass=factor(c(rep("A",100),rep("J",194)))
dipper.process=process.data(dipper,groups=c("sex","ageclass"),age.var=2,initial.ages=c(1,0),strata.labels=c("1

```

resight.matrix

Various utility functions

Description

Several functions have been added to help visualize data including `resight.matrix` which provides for each cohort the number of releases on the diagonal and the number resighted on each occasion in the upper-triangular matrix. `naive.survival` provides a naive survival estimate at each time for each cohort. The estimate for time i is the number resighted at time $i+1$ or later divided by the number seen at time i or later. These values should be interpreted cautiously because they are influenced by capture probability in year i but it is useful to identify particularly high or low survival values. Functions `Phi.mean` and `p.mean` compute average real parameter values Φ or p across time for a single age or across ages for a single time.

Usage

```

resight.matrix(x)

naive.survival(x)

mcmc_mode(x)

```

Arguments

`x` processed data list - result from `process.data` in marked or real estimates from fitted model

Value

matrix of values with cohort and year labels

Author(s)

Jeff Laake

sealions

*Multivariate State example data***Description**

An example data set using California sea lions to demonstrate the Multivariate state model with 3 variables defining the states : area, ltag and rtag. The left tag (ltag) and right tag (rtag) variables can be unknown. #'

Format

A data frame with 485 observations on the following 3 variables.

ch a character vector of 3-character values for each occasion separated by commas.

sex the sex of the sea lion designated as F or M

weight the anomaly of the weight from the sex-specific mean

Examples

```
### Load packages ###
# The splines package is only necessary for fitting b-spline curves used in the paper
# It is not required for the multivariate models in the marked package
library(splines)

# Get data
data(sealions)

# Process data for multivariate models in marked
dp=process.data(sealions,model="mvmscjs",
  strata.labels=list(area=c("A","S"),ltag=c("+","-","u"),rtag=c("+","-","u")))

### Make design data
ddl=make.design.data(dp)

# Create pup variable for Phi
ddl$Phi$pup=ifelse(ddl$Phi$Age==0,1,0)
ddl$Phi$sex=factor(ddl$Phi$sex)

# Detection model
# Set final year (2014) p=0 (no resight data) for ANI
ddl$p$fix = ifelse(ddl$p$Time==17 & ddl$p$area=="A", 0, ddl$p$fix)

# Delta model
# create indicator variables for 'unknown' tag observations
ddl$delta$obs.ltag.u = ifelse(ddl$delta$obs.ltag=="u", 1, 0)
ddl$delta$obs.rtag.u = ifelse(ddl$delta$obs.rtag=="u", 1, 0)

# Psi model
# Set Psi to 0 for cases which are not possible - missing tag to having tag
```

```

ddl$Psi$fix[as.character(ddl$Psi$ltag)=="-"&as.character(ddl$Psi$rtag)==""]=0
ddl$Psi$fix[as.character(ddl$Psi$rtag)=="-"&as.character(ddl$Psi$ltag)==""]=0
# Create indicator variables for transitioning between states
ddl$Psi$AtoS=ifelse(ddl$Psi$area=="A"&ddl$Psi$toarea=="S",1,0) # ANI to SMI movement
ddl$Psi$StoA=ifelse(ddl$Psi$area=="S"&ddl$Psi$toarea=="A",1,0) # SMI to ANI movement
ddl$Psi$lpm=ifelse(ddl$Psi$ltag=="+"&ddl$Psi$rtag=="-",1,0) # Losing left tag
ddl$Psi$rpm=ifelse(ddl$Psi$rtag=="+"&ddl$Psi$ltag=="-",1,0) # Losing right tag
ddl$Psi$sex=factor(ddl$Psi$sex)

# formulas
Psi.1=list(formula=~-1+ AtoS:sex + AtoS:sex:bs(Age) + StoA:sex + StoA:sex:bs(Age) +
           I(lpm+rpm) +I(lpm+rpm):Age + lpm:rpm)
p.1=list(formula=~-1+time:area)
delta.1=list(formula= ~ -1 + obs.ltag.u + obs.rtag.u + obs.ltag.u:obs.rtag.u)
Phi.1=list(formula=~sex*bs(Age)+pup:weight+area)

# Fit model with TMB
mod_sealions=crm(dp,ddl,model.parameters=list(Psi=Psi.1,p=p.1,delta=delta.1,Phi=Phi.1),
method="nlminb",hessian=TRUE,save.matrices=TRUE)

```

set.fixed

Set fixed real parameter values in ddl

Description

Merges fixed real parameter values in parameter specification with those specified with fix in the design data list and returns the fixed values in the design data list.

Usage

```
set.fixed(ddl, parameters)
```

Arguments

ddl	design data list
parameters	parameter specification list

Value

design data list with fixed values

Author(s)

Jeff Laake

set.initial	<i>Set initial values</i>
-------------	---------------------------

Description

Sets initial values specified in a list.

Usage

```
set.initial(pars, dml, initial)
```

Arguments

pars	character vector of parameter names
dml	list of design matrices created by create.dm from formula and design data
initial	list of vectors for parameter initial values

Value

List of initial values for each parameter in the model

Author(s)

Jeff Laake

set.scale	<i>Scaling functions</i>
-----------	--------------------------

Description

Set scale, scale dm and scale/unscale parameters

Usage

```
set.scale(pars, model_data, scale)
scale.dm(model_data, scale)
scale.par(par, scale)
unscale.par(par, scale)
```

Arguments

pars	character vector of parameter names
model_data	list of data/design objects
scale	list or vector of parameter scales
par	list of parameter vectors or vector of parameter values

Value

List of scale values for set.scale, model.data with scaled design matrices for scale.dm, vector of scaled parameter values for scale.par, and list of unscaled parameter vectors for unscale.par

Author(s)

Jeff Laake

setup.model	<i>Defines model specific parameters (internal use)</i>
-------------	---

Description

Compares model, the name of the type of model (eg "CJS") to the list of acceptable models to determine if it is supported and then creates some global fields specific to that type of model that are used to modify the operation of the code.

Usage

```
setup.model(model, nocc, mixtures = 1)
```

Arguments

model	name of model type (must be in vector valid.models)
nocc	length of capture history string
mixtures	number of mixtures

Details

In general, the structure of the different types of models (e.g., "CJS", "Recovery",...etc) are very similar with some minor exceptions. This function is not intended to be called directly by the user but it is documented to enable other models to be added. This function is called by other functions to validate and setup model specific parameters. For example, for live/dead models, the length of the capture history is twice the number of capture occasions and the number of time intervals equals the number of capture occasions because the final interval is included with dead recoveries. Whereas, for recapture models, the length of the capture history is the number of capture occasions and the number of time intervals is 1 less than the number of occasions. This function validates that the model is valid and sets up some parameters specific to the model that are used in the code.

Value

model.list	- a list with following elements
etype	encounter type string for MARK input; typically same as model
nocc	number of capture occasions
num	number of time intervals relative to number of occasions (0 or -1)
mixtures	number of mixtures if any
derived	logical; TRUE if model produces derived estimates

Author(s)

Jeff Laake

See Also

[setup.parameters](#), [valid.parameters](#)

setup.parameters	<i>Setup parameter structure specific to model (internal use)</i>
------------------	---

Description

Defines list of parameters used in the specified type of model (model) and adds default values for each parameter to the list of user specified values (eg formula, link etc).

Usage

```
setup.parameters(
  model,
  parameters = list(),
  nocc = NULL,
  check = FALSE,
  number.of.groups = 1
)
```

Arguments

model	type of model ("CJS", "Burnham" etc)
parameters	list of model parameter specifications
nocc	number of occasions (value only specified if needed)
check	if TRUE only the vector of parameter names is returned par.list
number.of.groups	number of groups defined for data

Value

The return value depends on the argument check. If it is TRUE then the return value is a vector of the names of the parameters used in the specified type of model. For example, if model="CJS" then the return value is c("Phi", "p"). This is used by the function `valid.parameters` to make sure that parameter specifications are valid for the model (i.e., specifying recovery rate r for "CJS" would give an error). If the function is called with the default of check=FALSE, the function returns a list of parameter specifications which is a modification of the argument parameters which adds parameters not specified and default values for all types of parameters that were not specified. The list length and names of the list elements depends on the type of model. Each element of the list is itself a list with varying numbers of elements which depend on the type of parameter although some elements are the same for all parameters. Below the return value list is shown generically with parameters named p1,...,pk.

```

p1  List of specifications for parameter 1
p2  List of specifications for parameter 2
.
.
.
pk  List of specifications for parameter k

```

The elements for each parameter list all include:

```

begin    0 or 1; beginning time for the first
          parameter relative to first occasion
num      0 or -1; number of parameters relative to
          number of occasions
type     type of PIM structure; either "Triang" or "Square"
formula  formula for parameter model (e.g., ~time)
link     link function for parameter (e.g., "logit")

```

and may include:

```

share    only valid for p in closed capture models;
          if TRUE p and c models shared
mix      only valid for closed capture heterogeneity
          models; if TRUE mixtures are used
rows     only valid for closed capture heterogeneity models
fixed    fixed values specified by user and
          not used modified in this function

```

Author(s)

Jeff Laake

See Also

[setup.model.valid.parameters](#)

setup_tmb

TMB setup

Description

Sets up executable for the .cpp file (tpl) by looking for exe in package directory or compiles cpp file in local directory (clean=FALSE) or from package directory.

Usage

```
setup_tmb(tpl, clean = FALSE, debug = FALSE)
```

Arguments

tpl	character string for admb template file
clean	if TRUE, deletes the tpl (.cpp) and executable files in local directory and copies from package directory
debug	if TRUE, writes out debug line values in code

set_mvms

Multivariate Multistate (mvms) Specification

Description

Creates list data structure from mvms specification

Usage

```
set_mvms(x)
```

Arguments

x	a multivariate multistate (mvms) specification as described above
---	---

Details

Accepts a mvms specification which is a list with named character vectors and optionally a vector named exclude. The length of the list (except for exclude) is the multivariate dimension. The name of each dimension is the list element name. Each character vector specifies the one character labels for the states of that dimension and optionally a reserved character "u" to specify that there is state uncertainty for that dimension. The name vector exclude can be used to remove variable combinations that cannot occur in the data.

The code tests to make sure that the input mvms specification is of the correct structure and it stops with an error message if not. The code returns a list structure with a number of elements described under return value below.

Value

a list with the following elements: 1) mvms - the input specification, 2) nd - the number of dimensions, 3) df - the dataframe containing all combinations of observations across dimensions including uncertain states, 4) df.states - the dataframe with all combinations of states across dimensions, 5) uncertain - boolean vector with nd elements indicating whether there is uncertainty in states for each dimension.

Author(s)

Jeff Laake

Examples

```
set_mvms(list(location=c("A","B","C"),repro_status=c("N","P","u"),exclude=c("CP")))
```

skagit

An example of the Multistrata (multi-state) model in which states are routes taken by migrating fish.

Description

An example of the Multistrata (multi-state) model in which states are routes taken by migrating fish.

Format

A data frame with 100 observations on the following 2 variables.

ch capture history

tag tag type v7 or v9

Author(s)

Megan Moore <megan.moore at noaa.gov>

Examples

```
# There are just two states which correspond to route A and route B. There are 6 occasions
# which are the locations rather than times. After release at 1=A there is no movement
# between states for the first segment, fish are migrating downriver together and all pass 2A.
# Then after occasion 2, migrants go down the North Fork (3A) or the South Fork (3B),
# which both empty into Skagit Bay. Once in saltwater, they can go north to Deception Pass (4A)
# or South to a receiver array exiting South Skagit Bay (4B). Fish in route A can then only go
# to the Strait of Juan de Fuca, while fish in route B must pass by Admiralty Inlet (5B).
# Then both routes end with the array at the Strait of Juan de Fuca.
#
#      1A
#      |
```



```

#           2A
#         /   \
#       3A     3B
#     / \   / \
#  4A 4B 4A 4B
#  |   \ /   |
#  5A  5B 5A 5B
#     \ \ /  /
#           6
#
# from 3A and 3B they can branch to either 4A or 4B; branches merge at 6
# 5A does not exist so p=0; only survival from 4A to 6 can be
# estimated which is done by setting survival from 4A to 5A to 1 and
# estimating survival from 5A to 6 which is then total survival from 4A to 6.

# See help for mscjs_tmb for an example that explains difference between marked and RMark
# with regard to treatment of mlogit parameters like Psi.

```

smsld_tmb

Fitting function for Multistate CJS live-dead models with TMB

Description

A function for computing MLEs for a Multi-state Cormack-Jolly-Seber open population capture-recapture with dead recoveries for processed dataframe `x` with user specified formulas in parameters that create list of design matrices `dml`. This function can be called directly but is most easily called from `crm` that sets up needed arguments.

Usage

```

smsld_tmb(
  x,
  ddl,
  dml,
  model_data = NULL,
  parameters,
  accumulate = TRUE,
  initial = NULL,
  method,
  hessian = FALSE,
  debug = FALSE,
  chunk_size = 1e+07,
  refit,
  itnmax = NULL,
  control = NULL,
  scale,
  re = FALSE,
  extra.args = "",

```

```

    getreals = FALSE,
    useHess = FALSE,
    savef = FALSE,
    ...
)

```

Arguments

<code>x</code>	processed dataframe created by <code>process.data</code>
<code>ddl</code>	list of simplified dataframes for design data; created by call to <code>make.design.data</code>
<code>dml</code>	list of design matrices created by <code>create.dm</code> from formula and design data
<code>model_data</code>	a list of all the relevant data for fitting the model including <code>imat</code> , <code>S.dm</code> , <code>r.dm</code> , <code>p.dm</code> , <code>Psi.dm</code> , <code>S.fixed</code> , <code>r.fixed</code> , <code>p.fixed</code> and <code>time.intervals</code> . It is used to save values and avoid accumulation again if the model was re-rerun with an additional call to <code>cjs</code> when using <code>autoscale</code> or re-starting with initial values. It is stored with returned model object.
<code>parameters</code>	equivalent to <code>model.parameters</code> in <code>crm</code>
<code>accumulate</code>	if TRUE will accumulate capture histories with common value and with a common design matrix for all parameters speed up execution
<code>initial</code>	list of initial values for parameters if desired; if each is a named vector from previous run it will match to columns with same name
<code>method</code>	method to use for optimization; see <code>optim</code>
<code>hessian</code>	if TRUE will compute and return the hessian
<code>debug</code>	if TRUE will print out information for each iteration
<code>chunk_size</code>	specifies amount of memory to use in accumulating capture histories; amount used is $8 * \text{chunk_size} / 1e6$ MB (default 80MB)
<code>refit</code>	non-zero entry to refit
<code>itnmax</code>	maximum number of iterations
<code>control</code>	control string for optimization functions
<code>scale</code>	vector of scale values for parameters
<code>re</code>	if TRUE creates random effect model <code>admbejsre.tpl</code> and runs <code>admb</code> optimizer
<code>extra.args</code>	optional character string that is passed to <code>tmb</code>
<code>getreals</code>	if TRUE, compute real values and std errors for TMB models; may want to set as FALSE until model selection is complete
<code>useHess</code>	if TRUE, the TMB hessian function is used for optimization; using <code>hessian</code> is typically slower with many parameters but can result in a better solution
<code>savef</code>	if TRUE, save optimization function in model for reporting
<code>...</code>	not currently used

Details

It is easiest to call `msld_tmb` through the function `crm`. Details are explained there.

Value

The resulting value of the function is a list with the class of `crm,cjs` such that the generic functions `print` and `coef` can be used.

<code>beta</code>	named vector of parameter estimates
<code>lnl</code>	$-2 \times \log$ likelihood
<code>AIC</code>	$\ln l + 2 \times$ number of parameters
<code>convergence</code>	result from <code>optim</code> ; if 0 <code>optim</code> thinks it converged
<code>count</code>	<code>optim</code> results of number of function evaluations
<code>reals</code>	dataframe of data and real S and p estimates for each animal-occasion excluding those that occurred before release
<code>vcv</code>	var-cov matrix of betas if <code>hessian=TRUE</code> was set

Author(s)

Jeff Laak

<code>splitCH</code>	<i>Split/collapse capture histories</i>
----------------------	---

Description

`splitCH` will split a character string vector of capture histories into a matrix. The `ch` can either be single character or comma separated string. The matrix is appended to the original data set (`data`) if one is specified. Will handle character and numeric values in `ch`. Results will differ depending on content of `ch`. `collapseCH` will collapse a capture history matrix back into a character vector. Argument can either be a capture history matrix (`chmat`) or a dataframe (`data`) that contains fields with a specified prefix.

Usage

```
splitCH(x="ch", data=NULL, prefix="Time")

collapseCH(chmat=NULL, data=NULL, prefix="Time", collapse="")
```

Arguments

<code>x</code>	A vector containing the character strings of capture histories or the column number or name in the data set <code>data</code>
<code>data</code>	A data frame containing column with value in <code>x</code> if <code>x</code> indicates a column in a data frame
<code>prefix</code>	first portion of field names for split <code>ch</code>
<code>chmat</code>	capture history matrix
<code>collapse</code>	in <code>collapseCH</code> the separator for <code>ch</code> string; defaults to <code>"</code> but <code>,</code> also useful if multi-characters are used

Value

A data frame if data specified and a matrix if vector ch is specified

Author(s)

Devin Johnson; Jeff Laake

Examples

```
data(dipper)
# following returns a matrix
chmat=splitCH(dipper$ch)
# following returns the original dataframe with the ch split into columns
newdipper=splitCH(data=dipper)
# following collapses chmat
ch=collapseCH(chmat)
# following finds fields in newdipper and creates ch
newdipper$ch=NULL
newdipper=collapseCH(data=newdipper)
```

valid.parameters

Determine validity of parameters for a model (internal use)

Description

Checks to make sure specified parameters are valid for a particular type of model.

Usage

```
valid.parameters(model, parameters)
```

Arguments

model type of c-r model ("CJS", "Burnham" etc)
parameters vector of parameter names (for example "Phi" or "p" or "S")

Value

Logical; TRUE if all parameters are acceptable and FALSE otherwise

Author(s)

Jeff Laake

See Also

[setup.parameters](#), [setup.model](#)

Values	<i>Summarize statistic values across parametric bootstraps of state predictions</i>
--------	---

Description

From bootstraps replicates of the state predictions, compute the value of a statistic summarizing some attribute. Computes the statistic value for the fitted model, standard error of the statistic value (std dev across bootstrap replicates), lower and upper 95 confidence intervals across replicates.

Usage

```
Values(pp, bs, f, index = NULL)
```

Arguments

pp	state prediction matrix at fitted model estimates
bs	list of bootstrap replicates of state prediction matrices
f	function to compute some desired value for each state prediction matrix
index	if NULL, f only produces a single value; otherwise index specifies which of the values should be summarized

Value

A dataframe containing the estimate, std error, 95

Author(s)

Jeff Laake

Index

* datasets

dipper, 22
mstrata, 43
Paradise_shelduck, 54
sealions, 65
skagit, 72

* models

crm, 16
crm.wrapper, 20

* utility

backward_prob, 3
boot_pred_states, 4
cjs.accumulate, 4
coef.crm, 6
compute_matrices, 7
compute_real, 7
convert.link.to.real, 9
create.fixed.matrix, 15
deriv_inverse.link, 21
dmat_hsmm2hmm, 23
function.wrapper, 25
global_decode, 25
hsmm2hmm, 26
inverse.link, 28
load_lib, 29
local_decode, 29
make.design.data, 30
merge_design.covariates, 31
omega, 54
Phi.mean, 56
predict.crm, 57
print.crm, 58
print.crmlist, 59
process.data, 61
resight.matrix, 64
set.fixed, 66
setup.model, 68
setup.parameters, 69
valid.parameters, 76

Values, 77

accumulate_data (process.data), 61
backward_prob, 3
boot_pred_states, 4
cjs.accumulate, 4
cjs_delta, 5
coef.crm, 6
collapseCH (splitCH), 75
compute_matrices, 7
compute_real, 7, 10, 22, 28
convert.link.to.real, 9
create.dm, 10, 18, 19, 34, 38, 40, 42, 45, 67, 74
create.dmdf, 10, 11, 19
create.dml (create.dm), 10
create.fixed.matrix, 15
create.links, 15
create.model.list (crm.wrapper), 20
crm, 6, 16, 21, 33–35, 37–43, 45, 46, 58, 59, 62, 63, 73, 74
crm.wrapper, 20
crmlist_fromfiles (crm.wrapper), 20
deriv_inverse.link
 (deriv_inverse.link), 21
deriv_inverse.link, 21, 28
dipper, 22, 62, 63
dmat_hsmm2hmm, 23
fix.parameters, 24
function.wrapper, 25
fx.aic (function.wrapper), 25
fx.par.count (function.wrapper), 25
global_decode, 25
hsmm2hmm, 26

initiate_pi, 27
inverse.link, 9, 10, 22, 28

load.model (crm.wrapper), 20
load_lib, 29
local_decode, 29

make.design.data, 11, 18, 19, 30, 32, 34, 38,
40, 42, 45, 62, 74
mcmc_mode (resight.matrix), 64
merge, 32
merge.design.covariates
 (merge_design.covariates), 31
merge_design.covariates, 31, 31
mixed.model (mixed.model.admb), 32
mixed.model.admb, 32
model.table (crm.wrapper), 20
mscjs_tmb, 33
msjs_tmb, 39
msjsu_tmb, 37
msld_tmb, 41
mstrata, 43
mvms_design_data, 53
mvmscjs_delta (cjs_delta), 5
mvmscjs_tmb, 44

naive.survival (resight.matrix), 64

omega, 54

p.boxplot (Phi.mean), 56
p.mean (Phi.mean), 56
Paradise_shelduck, 54
Phi.boxplot (Phi.mean), 56
Phi.mean, 56
predict.crm, 57
print.crm, 58
print.crmlist, 59
proc.form, 59
process.ch, 60
process.data, 11, 14, 18, 19, 23, 30–32, 61
ps (Paradise_shelduck), 54

reindex (mixed.model.admb), 32
rerun_crm (crm.wrapper), 20
resight.matrix, 64

scale.dm (set.scale), 67
scale.par (set.scale), 67
sealions, 65

set.fixed, 66
set.initial, 67
set.scale, 67
set_mvms, 71
setup.model, 68, 71, 76
setup.parameters, 11, 69, 69, 76
setup_tmb, 71
setupHMM (setup.model), 68
skagit, 72
smsld_tmb, 73
splitCH, 75

unscale.par (set.scale), 67

valid.parameters, 69–71, 76
Values, 77