Package 'unmarked'

January 9, 2024

Version 1.4.1

Date 2024-01-08

Type Package

Title Models for Data from Unmarked Animals

Depends R (>= 4.0)

Imports graphics, lattice, lme4, MASS, Matrix, methods, parallel, Rcpp (>= 0.8.0), stats, TMB (>= 1.7.18), utils

Suggests pbapply, knitr, rmarkdown, raster, shiny, testthat, terra

Description

Fits hierarchical models of animal abundance and occurrence to data collected using survey methods such as point counts, site occupancy sampling, distance sampling, removal sampling, and double observer sampling. Parameters governing the state and observation processes can be modeled as functions of covariates. References: Kellner et al. (2023) <doi:10.1111/2041-210X.14123>, Fiske and Chandler (2011) <doi:10.18637/jss.v043.i10>.

License GPL (>= 3)

LazyLoad yes

LazyData yes

Collate 'classes.R' 'unmarkedEstimate.R' 'mapInfo.R' 'unmarkedFrame.R' 'unmarkedFit.R' 'utils.R' 'getDesign.R' 'colext.R' 'distsamp.R' 'multinomPois.R' 'occu.R' 'occuRN.R' 'occuMulti.R' 'pcount.R' 'gmultmix.R' 'pcountOpen.R' 'gdistsamp.R' 'unmarkedFitList.R' 'unmarkedLinComb.R' 'ranef.R' 'boot.R' 'occuFP.R' 'gpcount.R' 'occuPEN.R' 'pcount.spHDS.R' 'occuMS.R' 'occuTTD.R' 'distsampOpen.R' 'multmixOpen.R' 'unmarkedCrossVal.R' 'piFun.R' 'vif.R' 'makePiFun.R' 'posteriorSamples.R' 'nmixTTD.R' 'gdistremoval.R' 'plotEffects.R' 'mixedModelTools.R' 'power.R' 'simulate.R' 'predict.R' 'goccu.R' 'occuCOP.R' 'RcppExports.R' 'zzz.R'

LinkingTo Rcpp, RcppArmadillo, TMB, RcppEigen

SystemRequirements GNU make

URL https://groups.google.com/d/forum/unmarked,
https://rbchan.github.io/unmarked/,
https://github.com/ianfiske/unmarked,
https://github.com/rbchan/unmarked

BugReports https://github.com/rbchan/unmarked/issues

VignetteBuilder knitr

NeedsCompilation yes

Author Richard Chandler [aut],
Ken Kellner [cre, aut],
Ian Fiske [aut],
David Miller [aut],
Andy Royle [aut],
Jeff Hostetler [aut],
Rebecca Hutchinson [aut],
Adam Smith [aut],
Lea Pautrel [aut],
Marc Kery [ctb],
Mike Meredith [ctb],
Auriel Fournier [ctb],
Ariel Muldoon [ctb],
Chris Baker [ctb]

Maintainer Ken Kellner <contact@kenkellner.com>

Repository CRAN

Date/Publication 2024-01-09 10:20:02 UTC

R topics documented:

`unmarked-package` *Models for Data from Unmarked Animals*

Description

Fits hierarchical models of animal occurrence and abundance to data collected on species that may be detected imperfectly. Models include single- and multi-season site occupancy models, binomial N-mixture models, and multinomial N-mixture models. The data can arise from survey methods such as occurrence sampling, temporally replicated counts, removal sampling, double observer sampling, and distance sampling. Parameters governing the state and observation processes can be

modeled as functions of covariates. General treatment of these models can be found in MacKenzie et al. (2006) and Royle and Dorazio (2008). The primary reference for the package is Fiske and Chandler (2011).

Details

Overview of Model-fitting Functions:

occu fits occurrence models with no linkage between abundance and detection (MacKenzie et al. 2002).

occuRN fits abundance models to presence/absence data by exploiting the link between detection probability and abundance (Royle and Nichols 2003).

occuFP fits occupancy models to data characterized by false negatives and false positive detections (e.g., Royle and Link [2006] and Miller et al. [2011]).

occuMulti fits multi-species occupancy model of Rota et al. [2016].

colext fits the mutli-season occupancy model of MacKenzie et al. (2003).

pcount fits N-mixture models (aka binomial mixture models) to repeated count data (Royle 2004a, Kery et al 2005).

distsamp fits the distance sampling model of Royle et al. (2004) to distance data recorded in discrete intervals.

gdistsamp fits the generalized distance sampling model described by Chandler et al. (2011) to distance data recorded in discrete intervals.

gpcount fits the generalized N-mixture model described by Chandler et al. (2011) to repeated count data collected using the robust design.

multinomPois fits the multinomial-Poisson model of Royle (2004b) to data collected using methods such as removal sampling or double observer sampling.

gmultmix fits a generalized form of the multinomial-mixture model of Royle (2004b) that allows for estimating availability and detection probability.

pcountOpen fits the open population model of Dail and Madsen (2011) to repeated count data. This is a genearlized form of the Royle (2004a) N-mixture model that includes parameters for recruitment and apparent survival.

Data: All data are passed to unmarked's estimation functions as a formal S4 class called an unmarkedFrame, which has child classes for each model type. This allows metadata (eg as distance interval cut points, measurement units, etc...) to be stored with the response and covariate data. See unmarkedFrame for a detailed description of unmarkedFrames and how to create them.

Model Specification: *unmarked*'s model-fitting functions allow specification of covariates for both the state process and the detection process. For two-level hierarchical models, (eg occu, occuRN, pcount, multinomPois, distsamp) covariates for the detection process (at the site or observation level) and the state process (at the site level) are specified with a double right-hand sided formula, in that order. Such a formula looks like

$x1 + x2 + \ldots + x_n \; x_1 + x_2 + \ldots + x_n$

where x_1 through x_n are additive covariates of the process of interest. Using two tildes in a single formula differs from standard R convention, but it is informative about the model being fit. The meaning of these covariates, or what they model, is full described in the help files for the individual functions and is not the same for all functions. For models with more than two processes (eg colext,

gmultmix, pcountOpen), single right-hand sided formulas (only one tilde) are used to model each parameter.

Utility Functions: *unmarked* contains several utility functions for organizing data into the form required by its model-fitting functions. csvToUMF converts an appropriately formated comma-separated values (.csv) file to a list containing the components required by model-fitting functions.

Author(s)

Ian Fiske, Richard Chandler, Andy Royle, Marc Kery, David Miller, and Rebecca Hutchinson

References

Chandler, R. B., J. A. Royle, and D. I. King. 2011. Inference about density and temporary emigration in unmarked populations. *Ecology* 92:1429-1435.

Dail, D. and L. Madsen. 2011. Models for estimating abundance from repeated counts of an open metapopulation. *Biometrics* 67:577-587.

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

Kery, M., Royle, J. A., and Schmid, H. 2005 Modeling avian abundance from replicated counts using binomial mixture models. *Ecological Applications* 15:1450–1461.

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

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

MacKenzie, D. I., J. D. Nichols, J. A. Royle, K. H. Pollock, L. L. Bailey, and J. E. Hines. 2006. *Occupancy Estimation and Modeling*. Amsterdam: Academic Press.

Miller, D.A., J.D. Nichols, B.T. McClintock, E.H.C. Grant, L.L. Bailey, and L.A. Weir. 2011. Improving occupancy estimation when two types of observational error occur: non-detection and species misidentification. *Ecology* 92:1422-1428.

Rota, C.T., et al. 2016. A multi-species occupancy model for two or more interacting species. Methods in Ecology and Evolution 7: 1164-1173.

Royle, J. A. 2004a. N-Mixture models for estimating population size from spatially replicated counts. *Biometrics* 60:108–105.

Royle, J. A. 2004b. Generalized estimators of avian abundance from count survey data. *Animal Biodiversity and Conservation* 27:375–386.

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

Royle, J. A., and R. M. Dorazio. 2006. Hierarchical models of animal abundance and occurrence. *Journal Of Agricultural Biological And Environmental Statistics* 11:249–263.

Royle, J.A., and W.A. Link. 2006. Generalized site occupancy models allowing for false positive and false negative errors. *Ecology* 87:835-841.

Royle, J. A. and R. M. Dorazio. 2008. *Hierarchical Modeling and Inference in Ecology*. Academic Press.

Royle, J. A. and J. D. Nichols. 2003. Estimating Abundance from Repeated Presence-Absence Data or Point Counts. *Ecology*, 84:777–790.

Sillett, S. and Chandler, R.B. and Royle, J.A. and Kery, M. and Morrison, S.A. In Press. Hierarchical distance sampling models to estimate population size and habitat-specific abundance of an island endemic. *Ecological Applications*

Examples

```
## An example site-occupancy analysis

# Simulate occupancy data
set.seed(344)
nSites <- 100
nReps <- 5
covariates <- data.frame(veght=rnorm(nSites),
    habitat=factor(c(rep('A', 50), rep('B', 50))))

psipars <- c(-1, 1, -1)
ppars <- c(1, -1, 0)
X <- model.matrix(~veght+habitat, covariates) # design matrix
psi <- plogis(X %*% psipars)
p <- plogis(X %*% ppars)

y <- matrix(NA, nSites, nReps)
z <- rbinom(nSites, 1, psi)       # true occupancy state
for(i in 1:nSites) {
    y[i,] <- rbinom(nReps, 1, z[i]*p[i])
    }

# Organize data and look at it
umf <- unmarkedFrameOccu(y = y, siteCovs = covariates)
head(umf)
summary(umf)


# Fit some models
fm1 <- occu(~1 ~1, umf)
fm2 <- occu(~veght+habitat ~veght+habitat, umf)
fm3 <- occu(~veght ~veght+habitat, umf)


# Model selection
fms <- fitList(m1=fm1, m2=fm2, m3=fm3)
modSel(fms)

# Empirical Bayes estimates of the number of sites occupied
sum(bup(ranef(fm3), stat="mode"))     # Sum of posterior modes
sum(z)                                # Actual
```

```
# Model-averaged prediction and plots

# psi in each habitat type
newdata1 <- data.frame(habitat=c('A', 'B'), veght=0)
Epsi1 <- predict(fms, type="state", newdata=newdata1)
with(Epsi1, {
    plot(1:2, Predicted, xaxt="n", xlim=c(0.5, 2.5), ylim=c(0, 0.5),
        xlab="Habitat",
        ylab=expression(paste("Probability of occurrence (", psi, ")")),
        cex.lab=1.2,
        pch=16, cex=1.5)
    axis(1, 1:2, c('A', 'B'))
    arrows(1:2, Predicted-SE, 1:2, Predicted+SE, angle=90, code=3, length=0.05)
    })


# psi and p as functions of vegetation height
newdata2 <- data.frame(habitat=factor('A', levels=c('A','B')),
    veght=seq(-2, 2, length=50))
Epsi2 <- predict(fms, type="state", newdata=newdata2, appendData=TRUE)
Ep <- predict(fms, type="det", newdata=newdata2, appendData=TRUE)

op <- par(mfrow=c(2, 1), mai=c(0.9, 0.8, 0.2, 0.2))
plot(Predicted~veght, Epsi2, type="l", lwd=2, ylim=c(0,1),
    xlab="Vegetation height (standardized)",
    ylab=expression(paste("Probability of occurrence (", psi, ")")))
    lines(lower ~ veght, Epsi2, col=gray(0.7))
    lines(upper ~ veght, Epsi2, col=gray(0.7))
plot(Predicted~veght, Ep, type="l", lwd=2, ylim=c(0,1),
    xlab="Vegetation height (standardized)",
    ylab=expression(paste("Detection probability (", italic(p), ")")))
lines(lower~veght, Ep, col=gray(0.7))
lines(upper~veght, Ep, col=gray(0.7))
par(op)
```

`backTransform-methods` *Methods for Function backTransform in Package 'unmarked'*

Description

Methods for function `backTransform` in Package 'unmarked'. This converts from link-scale to original-scale

Usage

```
## S4 method for signature 'unmarkedFit'
backTransform(obj, type)
```

```
## S4 method for signature 'unmarkedEstimate'
backTransform(obj)
```

Arguments

obj	Object of appropriate S4 class
type	one of names(obj), eg 'state' or 'det'

Methods

obj = "unmarkedEstimate" Typically done internally

obj = "unmarkedFit" Back-transform a parameter from a fitted model. Only possible if no covariates are present. Must specify argument type as one of the values returned by names(obj).

obj = "unmarkedLinComb" Back-transform a predicted value created by `linearComb`. This is done internally by `predict` but can be done explicitly by user.

Examples

```
## Not run:

data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
    obsCovs = mallard.obs)

(fm <- pcount(~ 1 ~ forest, mallardUMF))    # Fit a model
backTransform(fm, type="det")          # This works because there are no detection covariates
#backTransform(fm, type="state")         # This doesn't work because covariates are present
lc <- linearComb(fm, c(1, 0), type="state") # Estimate abundance on the log scale when forest=0
backTransform(lc)                           # Abundance on the original scale

## End(Not run)
```

---

birds *BBS Point Count and Occurrence Data from 2 Bird Species*

Description

Data frames for 2 species from the breeding bird survey (BBS). Each data frame has a row for each site and columns for each sampling event. There is a point count and occurrence–designated by `.bin`– version for each species.

Usage

```
data(birds)
```

Format

`catbird` A data frame of point count observations for the catbird.

`catbird.bin` A data frame of occurrence observations for the catbird.

`woodthrush` A data frame of point count observations for the wood thrush.

`woodthrush.bin` A data frame of point count observations for the wood thrush.

Source

Royle J. N-mixture models for estimating population size from spatially replicated counts. Biometrics. 2004. 60(1):108–115.

Examples

```
data(birds)
```

`coef-methods` *Methods for Function coef in Package 'unmarked'*

Description

Extract coefficients

Usage

```
## S4 method for signature 'unmarkedFit'
coef(object, type, altNames = TRUE, fixedOnly=TRUE)
## S4 method for signature 'unmarkedEstimate'
coef(object, altNames = TRUE, fixedOnly=TRUE, ...)
## S4 method for signature 'linCombOrBackTrans'
coef(object)
```

Arguments

`object`	Object of appropriate S4 class
`type`	Either 'state' or 'det'
`altNames`	Return specific names for parameter estimates?
`fixedOnly`	Return only fixed effect parameters?
`...`	Further arguments. Not currently used

Value

A named numeric vector of parameter estimates.

Methods

object = "linCombOrBackTrans" Object from linearComb

object = "unmarkedEstimate" unmarkedEstimate object

object = "unmarkedFit" Fitted model

`colext` *Fit the dynamic occupancy model of MacKenzie et. al (2003)*

Description

Estimate parameters of the colonization-extinction model, including covariate-dependent rates and detection process.

Usage

```
colext(psiformula= ~1, gammaformula =  ~ 1, epsilonformula = ~ 1,
    pformula = ~ 1, data, starts, method="BFGS", se=TRUE, ...)
```

Arguments

`psiformula`	Right-hand sided formula for the initial probability of occupancy at each site.
`gammaformula`	Right-hand sided formula for colonization probability.
`epsilonformula`	Right-hand sided formula for extinction probability.
`pformula`	Right-hand sided formula for detection probability.
`data`	unmarkedMultFrame object that supplies the data (see `unmarkedMultFrame`).
`starts`	optionally, initial values for parameters in the optimization.
`method`	Optimization method used by `optim`.
`se`	logical specifying whether or not to compute standard errors.
`...`	Additional arguments to optim, such as lower and upper bounds

Details

This function fits the colonization-extinction model of MacKenzie et al (2003). The colonization and extinction rates can be modeled with covariates that vary yearly at each site using a logit link. These covariates are supplied by special unmarkedMultFrame `yearlySiteCovs` slot. These parameters are specified using the `gammaformula` and `epsilonformula` arguments. The initial probability of occupancy is modeled by covariates specified in the `psiformula`.

The conditional detection rate can also be modeled as a function of covariates that vary at the secondary sampling period (ie., repeat visits). These covariates are specified by the first part of the `formula` argument and the data is supplied via the usual `obsCovs` slot.

The projected and smoothed trajectories (Weir et al 2009) can be obtained from the `smoothed.mean` and `projected.mean` slots (see examples).

Value

unmarkedFitColExt object describing model fit.

References

MacKenzie, D.I. et al. (2002) Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One. Ecology, 83(8), 2248-2255.

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

MacKenzie, D. I. et al. (2006) *Occupancy Estimation and Modeling*.Amsterdam: Academic Press.

Weir L. A., Fiske I. J., Royle J. (2009) Trends in Anuran Occupancy from Northeastern States of the North American Amphibian Monitoring Program. Herpetological Conservation and Biology. 4(3):389-402.

See Also

nonparboot, unmarkedMultFrame, and formatMult

Examples

```
# Fake data
R <- 4 # number of sites
J <- 3 # number of secondary sampling occasions
T <- 2 # number of primary periods

y <- matrix(c(
   1,1,0,  0,0,0,
   0,0,0,  0,0,0,
   1,1,1,  1,1,0,
   1,0,1,  0,0,1), nrow=R, ncol=J*T, byrow=TRUE)
y

site.covs <- data.frame(x1=1:4, x2=factor(c('A','B','A','B')))
site.covs

yearly.site.covs <- list(
   year = matrix(c(
      'year1', 'year2',
      'year1', 'year2',
      'year1', 'year2',
      'year1', 'year2'), nrow=R, ncol=T, byrow=TRUE)
      )
yearly.site.covs

obs.covs <- list(
   x4 = matrix(c(
      -1,0,1,  -1,1,1,
      -2,0,0,  0,0,2,
```

```
      -3,1,0,  1,1,2,
      0,0,0,   0,1,-1), nrow=R, ncol=J*T, byrow=TRUE),
   x5 = matrix(c(
      'a','b','c',  'a','b','c',
      'd','b','a',  'd','b','a',
      'a','a','c',  'd','b','a',
      'a','b','a',  'd','b','a'), nrow=R, ncol=J*T, byrow=TRUE))
obs.covs

umf <- unmarkedMultFrame(y=y, siteCovs=site.covs,
    yearlySiteCovs=yearly.site.covs, obsCovs=obs.covs,
    numPrimary=2)                 # organize data
umf                               # look at data
summary(umf)                      # summarize
fm <- colext(~1, ~1, ~1, ~1, umf)  # fit a model
fm



## Not run:
# Real data
data(frogs)
umf <- formatMult(masspcru)
obsCovs(umf) <- scale(obsCovs(umf))

## Use 1/4 of data just for run speed in example
umf <- umf[which((1:numSites(umf)) %% 4 == 0),]

## constant transition rates
(fm <- colext(psiformula = ~ 1,
gammaformula = ~ 1,
epsilonformula = ~ 1,
pformula = ~ JulianDate + I(JulianDate^2), umf, control = list(trace=1, maxit=1e4)))

## get the trajectory estimates
smoothed(fm)
projected(fm)

# Empirical Bayes estimates of number of sites occupied in each year
re <- ranef(fm)
modes <- colSums(bup(re, stat="mode"))
plot(1:7, modes, xlab="Year", ylab="Sites occupied", ylim=c(0, 70))

## Find bootstrap standard errors for smoothed trajectory
fm <- nonparboot(fm, B = 100)  # This takes a while!
fm@smoothed.mean.bsse

## try yearly transition rates
yearlySiteCovs(umf) <- data.frame(year = factor(rep(1:7, numSites(umf))))
(fm.yearly <- colext(psiformula = ~ 1,
gammaformula = ~ year,
epsilonformula = ~ year,
pformula = ~ JulianDate + I(JulianDate^2), umf,
```

```
control = list(trace=1, maxit=1e4)))

## End(Not run)
```

`computeMPLElambda` *Compute the penalty weight for the MPLE penalized likelihood method*

Description

This function computes the weight for the MPLE penalty of Moreno & Lele (2010).

Usage

```
computeMPLElambda(formula, data, knownOcc=numeric(0), starts,
method="BFGS",engine=c("C", "R"))
```

Arguments

`formula`	Double right-hand side formula describing covariates of detection and occupancy in that order.
`data`	An `unmarkedFrameOccu` object
`knownOcc`	Vector of sites that are known to be occupied. These should be supplied as row numbers of the y matrix, eg, c(3,8) if sites 3 and 8 were known to be occupied a priori.
`starts`	Vector of parameter starting values.
`method`	Optimization method used by `optim`.
`engine`	Either "C" or "R" to use fast C++ code or native R code during the optimization.

Details

See `occuPEN` for details and examples.

Value

The computed lambda.

Author(s)

Rebecca A. Hutchinson

References

Moreno, M. and S. R. Lele. 2010. Improved estimation of site occupancy using penalized likelihood. Ecology 91: 341-346.

See Also

unmarked, unmarkedFrameOccu, occu, occuPEN, occuPEN_CV, nonparboot

---

confint-methods *Methods for Function confint in Package 'unmarked'*

Description

Methods for function confint in Package 'unmarked'

Usage

```
## S4 method for signature 'unmarkedBackTrans'
confint(object, parm, level)
## S4 method for signature 'unmarkedEstimate'
confint(object, parm, level)
## S4 method for signature 'unmarkedLinComb'
confint(object, parm, level)
## S4 method for signature 'unmarkedFit'
confint(object, parm, level, type, method)
```

Arguments

object	Object of appropriate S4 class
parm	Name of parameter(s) of interest
level	Level of confidence
type	Either "state" or "det"
method	Either "normal" or "profile"

Value

A vector of lower and upper confidence intervals. These are asymtotic unless method='profile' is used on unmarkedFit objects in which case they are profile likelihood intervals.

See Also

unmarkedFit-class

`crossbill` *Detection/non-detection data on the European crossbill (*Loxia curvirostra*)*

Description

267 1-kmsq quadrats were surveyed 3 times per year during 1999-2007.

Usage

```
data(crossbill)
```

Format

A data frame with 267 observations on the following 58 variables.

`id` Plot ID

`ele` Elevation

`forest` Percent forest cover

`surveys` a numeric vector

`det991` Detection data for 1999, survey 1

`det992` Detection data for 1999, survey 2

`det993` Detection data for 1999, survey 3

`det001` Detection data for 2000, survey 1

`det002` a numeric vector

`det003` a numeric vector

`det011` a numeric vector

`det012` a numeric vector

`det013` a numeric vector

`det021` a numeric vector

`det022` a numeric vector

`det023` a numeric vector

`det031` a numeric vector

`det032` a numeric vector

`det033` a numeric vector

`det041` a numeric vector

`det042` a numeric vector

`det043` a numeric vector

`det051` a numeric vector

`det052` a numeric vector

`det053` a numeric vector

`det061` a numeric vector

`det062` a numeric vector

`det063` Detection data for 2006, survey 3

`det071` Detection data for 2007, survey 1

`det072` Detection data for 2007, survey 2

`det073` Detection data for 2007, survey 3

`date991` Day of the season for 1999, survey 1

`date992` Day of the season for 1999, survey 2

`date993` Day of the season for 1999, survey 3

`date001` Day of the season for 2000, survey 1

`date002` a numeric vector

`date003` a numeric vector

`date011` a numeric vector

`date012` a numeric vector

`date013` a numeric vector

`date021` a numeric vector

`date022` a numeric vector

`date023` a numeric vector

`date031` a numeric vector

`date032` a numeric vector

`date033` a numeric vector

`date041` a numeric vector

`date042` a numeric vector

`date043` a numeric vector

`date051` a numeric vector

`date052` a numeric vector

`date053` a numeric vector

`date061` a numeric vector

`date062` a numeric vector

`date063` a numeric vector

`date071` a numeric vector

`date072` a numeric vector

`date073` Day of the season for 2007, survey 3

Source

Schmid, H. N. Zbinden, and V. Keller. 2004. Uberwachung der Bestandsentwicklung haufiger Brutvogel in der Schweiz, Swiss Ornithological Institute Sempach Switzerland

See Also

Switzerland for corresponding covariate data defined for all 1-kmsq pixels in Switzerland. Useful for making species distribution maps.

Examples

```
data(crossbill)
str(crossbill)
```

---

crossVal *Cross-validation methods for fitted unmarked models and fit lists*

Description

Test predictive accuracy of fitted models using several cross-validation approaches. The dataset is divided by site only into folds or testing and training datasets (i.e., encounter histories within sites are never split up).

Usage

```
## S4 method for signature 'unmarkedFit'
crossVal(
  object, method=c("Kfold","holdout","leaveOneOut"),
  folds=10, holdoutPct=0.25, statistic=RMSE_MAE, parallel=FALSE, ncores, ...)
## S4 method for signature 'unmarkedFitList'
crossVal(
  object, method=c("Kfold","holdout","leaveOneOut"),
  folds=10, holdoutPct=0.25, statistic=RMSE_MAE, parallel=FALSE, ncores,
  sort = c("none", "increasing", "decreasing"), ...)
```

Arguments

object	A fitted model inheriting class unmarkedFit or a list of fitted models with class unmarkedFitList
method	Cross validation method to use as string. Valid options are "Kfold", "holdout", or "leaveOneOut"
folds	Number of folds to use for k-fold cross validation
holdoutPct	Proportion of dataset (value between 0-1) to use as the "holdout" or "test" set, for the holdout method
statistic	Function that calculates statistics for each fold. The function must take an unmarkedFit object as the first argument and return a named numeric vector with statistic value(s). The default function RMSE_MAE returns root-mean-square error and mean absolute error. See unmarked:::RMSE_MAE for an example of correct statistic function structure.

parallel	If TRUE, run folds in parallel. This may speed up cross-validation if the unmarked model takes a long time to fit or you have a large number of sites and are using leave-one-out cross-validation.
ncores	Number of parallel cores to use.
sort	If doing cross-validation on a fitList, you can optionally sort the resulting table(s) of statistic values for each model.
...	Other arguments passed to the statistic function.

Value

unmarkedCrossVal or unmarkedCrossValList object containing calculated statistic values for each fold.

Author(s)

Ken Kellner <contact@kenkellner.com>

See Also

fitList, unmarkedFit

Examples

```
## Not run:
#Get data
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
siteCovs(pferUMF) <- data.frame(sitevar1 = rnorm(numSites(pferUMF)))
obsCovs(pferUMF) <- data.frame(obsvar1 = rnorm(numSites(pferUMF) * obsNum(pferUMF)))

#Fit occupancy model
fm <- occu(~ obsvar1 ~ 1, pferUMF)

#k-fold cross validation with 10 folds
(kfold = crossVal(fm, method="Kfold", folds=10))

#holdout method with 25
(holdout = crossVal(fm,method='holdout', holdoutPct=0.25))

#Leave-one-out method
(leave = crossVal(fm, method='leaveOneOut'))

#Fit a second model and combine into a fitList
fm2 <- occu(~1 ~1, pferUMF)
fl <- fitList(fm2,fm)

#Cross-validation for all fits in fitList using holdout method
(cvlist <- crossVal(fl, method='holdout'))
```

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

---

cruz *Landscape data for Santa Cruz Island*

Description

Spatially-referenced elevation, forest cover, and vegetation data for Santa Cruz Island.

Usage

```
data(cruz)
```

Format

A data frame with 2787 observations on the following 5 variables.

`x` Easting (meters)

`y` Northing (meters)

`elevation` a numeric vector, FEET (multiply by 0.3048 to convert to meters)

`forest` a numeric vector, proportion cover

`chaparral` a numeric vector, proportion cover

Details

The resolution is 300x300 meters.

The Coordinate system is EPSG number 26911

NAD_1983_UTM_Zone_11N Projection: Transverse_Mercator False_Easting: 500000.000000 False_Northing: 0.000000 Central_Meridian: -117.000000 Scale_Factor: 0.999600 Latitude_Of_Origin: 0.000000 Linear Unit: Meter GCS_North_American_1983 Datum: D_North_American_1983

Source

Brian Cohen of the Nature Conservancy helped prepare the data

References

Sillett, S. and Chandler, R.B. and Royle, J.A. and Kery, M. and Morrison, S.A. In Press. Hierarchical distance sampling models to estimate population size and habitat-specific abundance of an island endemic. *Ecological Applications*

Examples

```
## Not run:
library(lattice)
data(cruz)
str(cruz)

levelplot(elevation ~ x + y, cruz, aspect="iso",
    col.regions=terrain.colors(100))

if(require(raster)) {
elev <- rasterFromXYZ(cruz[,1:3],
     crs="+proj=utm +zone=11 +ellps=GRS80 +datum=NAD83 +units=m +no_defs")
elev
plot(elev)
}

## End(Not run)
```

---

csvToUMF *Convert .CSV File to an unmarkedFrame*

Description

This function converts an appropriatedly formated comma-separated values file (.csv) to a format usable by *unmarked*'s fitting functions (see *Details*).

Usage

```
csvToUMF(filename, long=FALSE, type, species, ...)
```

Arguments

filename	string describing filename of file to read in
long	FALSE if file is in long format or TRUE if file is in long format (see *Details*)
species	if data is in long format with multiple species, then this can specify a particular species to extract if there is a column named "species".
type	specific type of unmarkedFrame.
...	further arguments to be passed to the unmarkedFrame constructor.

Details

This function provides a quick way to take a .csv file with headers named as described below and provides the data required and returns of data in the format required by the model-fitting functions in unmarked. The .csv file can be in one of 2 formats: long or wide. See the first 2 lines of the *examples* for what these formats look like.

The .csv file is formatted as follows:

- col 1 is site labels.
- if data is in long format, col 2 is date of observation.
- next J columns are the observations (y) - counts or 0/1's.
- next is a series of columns for the site variables (one column per variable). The column header is the variable name.
- next is a series of columns for the observation-level variables. These are in sets of J columns for each variable, e.g., var1-1 var1-2 var1-3 var2-1 var2-2 var2-3, etc. The column header of the first variable in each group must indicate the variable name.

Value

an unmarkedFrame object

Author(s)

Ian Fiske <ianfiske@gmail.com>

Examples

```
# examine a correctly formatted long .csv
head(read.csv(system.file("csv","frog2001pcru.csv", package="unmarked")))

# examine a correctly formatted wide .csv
head(read.csv(system.file("csv","widewt.csv", package="unmarked")))

# convert them!
dat1 <- csvToUMF(system.file("csv","frog2001pcru.csv", package="unmarked"),
                 long = TRUE, type = "unmarkedFrameOccu")
dat2 <- csvToUMF(system.file("csv","frog2001pfer.csv", package="unmarked"),
                 long = TRUE, type = "unmarkedFrameOccu")
dat3 <- csvToUMF(system.file("csv","widewt.csv", package="unmarked"),
                 long = FALSE, type = "unmarkedFrameOccu")
```

`detFuns` *Distance-sampling detection functions and associated density functions*

Description

These functions represent the currently available detection functions used for modeling line and point transect data with distsamp. Detection functions begin with "g", and density functions begin with a "d".

Usage

```
gxhn(x, sigma)
gxexp(x, rate)
gxhaz(x, shape, scale)

dxhn(x, sigma)
dxexp(x, rate)
dxhaz(x, shape, scale)
drhn(r, sigma)
drexp(r, rate)
drhaz(r, shape, scale)
```

Arguments

x	Perpendicular distance
r	Radial distance
sigma	Shape parameter of half-normal detection function
rate	Shape parameter of negative-exponential detection function
shape	Shape parameter of hazard-rate detection function
scale	Scale parameter of hazard-rate detection function

See Also

distsamp for example of using these for plotting detection function

Examples

```
# Detection probabilities at 25m for range of half-normal sigma values.
round(gxhn(25, 10:15), 2)

# Plot negative exponential distributions
plot(function(x) gxexp(x, rate=10), 0, 50, xlab="distance",
    ylab="Detection probability")
plot(function(x) gxexp(x, rate=20), 0, 50, add=TRUE, lty=2)
plot(function(x) gxexp(x, rate=30), 0, 50, add=TRUE, lty=3)

# Plot half-normal probability density functions for line- and point-transects
par(mfrow=c(2, 1))
plot(function(x) dxhn(x, 20), 0, 50, xlab="distance",
    ylab="Probability density", main="Line-transect")
plot(function(x) drhn(x, 20), 0, 50, xlab="distance",
    ylab="Probability density", main="Point-transect")
```

distsamp *Fit the hierarchical distance sampling model of Royle et al. (2004)*

Description

Fit the hierarchical distance sampling model of Royle et al. (2004) to line or point transect data recorded in discrete distance intervals.

Usage

```
distsamp(formula, data, keyfun=c("halfnorm", "exp",
  "hazard", "uniform"), output=c("density", "abund"),
  unitsOut=c("ha", "kmsq"), starts, method="BFGS", se=TRUE,
  engine=c("C", "R", "TMB"), rel.tol=0.001, ...)
```

Arguments

formula	Double right-hand formula describing detection covariates followed by abundance covariates. ~1 ~1 would be a null model.
data	object of class unmarkedFrameDS, containing response matrix, covariates, distance interval cut points, survey type ("line" or "point"), transect lengths (for survey = "line"), and units ("m" or "km") for cut points and transect lengths. See example for set up.
keyfun	One of the following detection functions: "halfnorm", "hazard", "exp", or "uniform." See details.
output	Model either "density" or "abund"
unitsOut	Units of density. Either "ha" or "kmsq" for hectares and square kilometers, respectively.
starts	Vector of starting values for parameters.
method	Optimization method used by optim.
se	logical specifying whether or not to compute standard errors.
engine	Use code written in C++ or R
rel.tol	Requested relative accuracy of the integral, see integrate
...	Additional arguments to optim, such as lower and upper bounds

Details

Unlike conventional distance sampling, which uses the 'conditional on detection' likelihood formulation, this model is based upon the unconditional likelihood and allows for modeling both abundance and detection function parameters.

The latent transect-level abundance distribution $f(N|\theta)$ assumed to be Poisson with mean λ (but see gdistsamp for alternatives).

The detection process is modeled as multinomial: $y_{ij} \sim Multinomial(N_i, \pi_{ij})$, where π_{ij} is the multinomial cell probability for transect i in distance class j. These are computed based upon a detection function $g(x|\sigma)$, such as the half-normal, negative exponential, or hazard rate.

Parameters λ and σ can be vectors affected by transect-specific covariates using the log link.

Value

unmarkedFitDS object (child class of unmarkedFit-class) describing the model fit.

Note

You cannot use obsCovs.

Author(s)

Richard Chandler <rbchan@uga.edu>

References

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

Sillett, S. and Chandler, R.B. and Royle, J.A. and Kery, M. and Morrison, S.A. In Press. Hierarchical distance sampling models to estimate population size and habitat-specific abundance of an island endemic. *Ecological Applications*

See Also

unmarkedFrameDS, unmarkedFit-class fitList, formatDistData, parboot, sight2perpdist, detFuns, gdistsamp, ranef. Also look at vignette("distsamp").

Examples

```
## Line transect examples

data(linetran)

ltUMF <- with(linetran, {
   unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
   siteCovs = data.frame(Length, area, habitat),
   dist.breaks = c(0, 5, 10, 15, 20),
   tlength = linetran$Length * 1000, survey = "line", unitsIn = "m")
   })

ltUMF
summary(ltUMF)
hist(ltUMF)

# Half-normal detection function. Density output (log scale). No covariates.
(fm1 <- distsamp(~ 1 ~ 1, ltUMF))

# Some methods to use on fitted model
```

```
summary(fm1)
backTransform(fm1, type="state")                # animals / ha
exp(coef(fm1, type="state", altNames=TRUE))     # same
backTransform(fm1, type="det")                  # half-normal SD
hist(fm1, xlab="Distance (m)") # Only works when there are no det covars
# Empirical Bayes estimates of posterior distribution for N_i
plot(ranef(fm1, K=50))

# Effective strip half-width
(eshw <- integrate(gxhn, 0, 20, sigma=10.9)$value)

# Detection probability
eshw / 20 # 20 is strip-width


# Halfnormal. Covariates affecting both density and and detection.
(fm2 <- distsamp(~area + habitat ~ habitat, ltUMF))

# Hazard-rate detection function.
(fm3 <- distsamp(~ 1 ~ 1, ltUMF, keyfun="hazard"))

# Plot detection function.
fmhz.shape <- exp(coef(fm3, type="det"))
fmhz.scale <- exp(coef(fm3, type="scale"))
plot(function(x) gxhaz(x, shape=fmhz.shape, scale=fmhz.scale), 0, 25,
xlab="Distance (m)", ylab="Detection probability")



## Point transect examples

# Analysis of the Island Scrub-jay data.
# See Sillett et al. (In press)

data(issj)
str(issj)

jayumf <- unmarkedFrameDS(y=as.matrix(issj[,1:3]),
 siteCovs=data.frame(scale(issj[,c("elevation","forest","chaparral")])),
 dist.breaks=c(0,100,200,300), unitsIn="m", survey="point")

(fm1jay <- distsamp(~chaparral ~chaparral, jayumf))




## Not run:

data(pointtran)

ptUMF <- with(pointtran, {
unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4, dc5),
siteCovs = data.frame(area, habitat),
```

```
dist.breaks = seq(0, 25, by=5), survey = "point", unitsIn = "m")
})

# Half-normal.
(fmp1 <- distsamp(~ 1 ~ 1, ptUMF))
hist(fmp1, ylim=c(0, 0.07), xlab="Distance (m)")

# effective radius
sig <- exp(coef(fmp1, type="det"))
ea <- 2*pi * integrate(grhn, 0, 25, sigma=sig)$value # effective area
sqrt(ea / pi) # effective radius

# detection probability
ea / (pi*25^2)


## End(Not run)
```

`distsampOpen` *Open population model for distance sampling data*

Description

Fit the model of Dail and Madsen (2011) and Hostetler and Chandler (2015) with a distance sampling observation model (Sollmann et al. 2015).

Usage

```
distsampOpen(lambdaformula, gammaformula, omegaformula, pformula,
    data, keyfun=c("halfnorm", "exp", "hazard", "uniform"),
    output=c("abund", "density"), unitsOut=c("ha", "kmsq"),
    mixture=c("P", "NB", "ZIP"), K,
    dynamics=c("constant", "autoreg", "notrend", "trend", "ricker", "gompertz"),
    fix=c("none", "gamma", "omega"), immigration=FALSE, iotaformula = ~1,
    starts, method="BFGS", se=TRUE, ...)
```

Arguments

`lambdaformula`	Right-hand sided formula for initial abundance
`gammaformula`	Right-hand sided formula for recruitment rate (when dynamics is "constant", "autoreg", or "notrend") or population growth rate (when dynamics is "trend", "ricker", or "gompertz")
`omegaformula`	Right-hand sided formula for apparent survival probability (when dynamics is "constant", "autoreg", or "notrend") or equilibrium abundance (when dynamics is "ricker" or "gompertz")
`pformula`	A right-hand side formula describing the detection function covariates
`data`	An object of class `unmarkedFrameDSO`

keyfun	One of the following detection functions: "halfnorm", "hazard", "exp", or "uniform"
output	Model either "density" or "abund"
unitsOut	Units of density. Either "ha" or "kmsq" for hectares and square kilometers, respectively
mixture	String specifying mixture: "P", "NB", or "ZIP" for the Poisson, negative binomial, or zero-inflated Poisson distributions respectively
K	Integer defining upper bound of discrete integration. This should be higher than the maximum observed count and high enough that it does not affect the parameter estimates. However, the higher the value the slower the computation
dynamics	Character string describing the type of population dynamics. "constant" indicates that there is no relationship between omega and gamma. "autoreg" is an auto-regressive model in which recruitment is modeled as gamma*N[i,t-1]. "notrend" model gamma as lambda*(1-omega) such that there is no temporal trend. "trend" is a model for exponential growth, N[i,t] = N[i,t-1]*gamma, where gamma in this case is finite rate of increase (normally referred to as lambda). "ricker" and "gompertz" are models for density-dependent population growth. "ricker" is the Ricker-logistic model, N[i,t] = N[i,t-1]*exp(gamma*(1-N[i,t-1]/omega)), where gamma is the maximum instantaneous population growth rate (normally referred to as r) and omega is the equilibrium abundance (normally referred to as K). "gompertz" is a modified version of the Gompertz-logistic model, N[i,t] = N[i,t-1]*exp(gamma*(1-log(N[i,t-1]+1)/log(omega+1))), where the interpretations of gamma and omega are similar to in the Ricker model
fix	If "omega", omega is fixed at 1. If "gamma", gamma is fixed at 0
immigration	Logical specifying whether or not to include an immigration term (iota) in population dynamics
iotaformula	Right-hand sided formula for average number of immigrants to a site per time step
starts	Vector of starting values
method	Optimization method used by optim
se	Logical specifying whether or not to compute standard errors
...	Additional arguments to optim, such as lower and upper bounds

Details

These models generalize distance sampling models (Buckland et al. 2001) by relaxing the closure assumption (Dail and Madsen 2011, Hostetler and Chandler 2015, Sollmann et al. 2015).

The models include two or three additional parameters: gamma, either the recruitment rate (births and immigrations), the finite rate of increase, or the maximum instantaneous rate of increase; omega, either the apparent survival rate (deaths and emigrations) or the equilibrium abundance (carrying capacity); and iota, the number of immigrants per site and year. Estimates of population size at each time period can be derived from these parameters, and thus so can trend estimates. Or, trend can be estimated directly using dynamics="trend".

When immigration is set to FALSE (the default), iota is not modeled. When immigration is set to TRUE and dynamics is set to "autoreg", the model will separately estimate birth rate (gamma) and

number of immigrants (iota). When immigration is set to TRUE and dynamics is set to "trend", "ricker", or "gompertz", the model will separately estimate local contributions to population growth (gamma and omega) and number of immigrants (iota).

The latent abundance distribution, $f(N|\theta)$ can be set as a Poisson, negative binomial, or zero-inflated Poisson random variable, depending on the setting of the `mixture` argument, `mixture = "P"`, `mixture = "NB"`, `mixture = "ZIP"` respectively. For the first two distributions, the mean of N_i is λ_i. If $N_i \sim NB$, then an additional parameter, α, describes dispersion (lower α implies higher variance). For the ZIP distribution, the mean is $\lambda_i(1-\psi)$, where psi is the zero-inflation parameter.

For "constant", "autoreg", or "notrend" dynamics, the latent abundance state following the initial sampling period arises from a Markovian process in which survivors are modeled as $S_{it} \sim Binomial(N_{it-1}, \omega_{it})$, and recruits follow $G_{it} \sim Poisson(\gamma_{it})$. Alternative population dynamics can be specified using the `dynamics` and `immigration` arguments.

λ_i, γ_{it}, and ι_{it} are modeled using the the log link. p_{ijt} is modeled using the logit link. ω_{it} is either modeled using the logit link (for "constant", "autoreg", or "notrend" dynamics) or the log link (for "ricker" or "gompertz" dynamics). For "trend" dynamics, ω_{it} is not modeled.

For the distance sampling detection process, half-normal (`"halfnorm"`), exponential (`"exp"`), hazard (`"hazard"`), and uniform (`"uniform"`) key functions are available.

Value

An object of class unmarkedFitDSO

Warning

This function can be extremely slow, especially if there are covariates of gamma or omega. Consider testing the timing on a small subset of the data, perhaps with se=FALSE. Finding the lowest value of K that does not affect estimates will also help with speed.

Note

When gamma or omega are modeled using year-specific covariates, the covariate data for the final year will be ignored; however, they must be supplied.

If the time gap between primary periods is not constant, an M by T matrix of integers should be supplied to `unmarkedFrameDSO` using the `primaryPeriod` argument.

Secondary sampling periods are optional, but can greatly improve the precision of the estimates.

Optimization may fail if the initial value of the intercept for the detection parameter (sigma) is too small or large relative to transect width. By default, this parameter is initialized at log(average band width). You may have to adjust this starting value.

Author(s)

Richard Chandler, Jeff Hostetler, Andy Royle, Ken Kellner

References

Buckland, S.T., Anderson, D.R., Burnham, K.P., Laake, J.L., Borchers, D.L. and Thomas, L. (2001) *Introduction to Distance Sampling: Estimating Abundance of Biological Populations*. Oxford University Press, Oxford, UK.

Dail, D. and L. Madsen (2011) Models for Estimating Abundance from Repeated Counts of an Open Metapopulation. *Biometrics*. 67: 577-587.

Hostetler, J. A. and R. B. Chandler (2015) Improved State-space Models for Inference about Spatial and Temporal Variation in Abundance from Count Data. *Ecology* 96: 1713-1723.

Sollmann, R., Gardner, B., Chandler, R.B., Royle, J.A. and Sillett, T.S. (2015) An open-population hierarchical distance sampling model. *Ecology* 96: 325-331.

See Also

distsamp, gdistsamp, unmarkedFrameDSO

Examples

```
## Not run:

#Generate some data
set.seed(123)
lambda=4; gamma=0.5; omega=0.8; sigma=25;
M=100; T=10; J=4
y <- array(NA, c(M, J, T))
N <- matrix(NA, M, T)
S <- G <- matrix(NA, M, T-1)
db <- c(0, 25, 50, 75, 100)

#Half-normal, line transect
g <- function(x, sig) exp(-x^2/(2*sig^2))

cp <- u <- a <- numeric(J)
L <-  1
a[1] <- L*db[2]
cp[1] <- integrate(g, db[1], db[2], sig=sigma)$value
for(j in 2:J) {
  a[j] <-  db[j+1]  - sum(a[1:j])
  cp[j] <- integrate(g, db[j], db[j+1], sig=sigma)$value
}
u <- a / sum(a)
cp <- cp / a * u
cp[j+1] <- 1-sum(cp)

for(i in 1:M) {
  N[i,1] <- rpois(1, lambda)
  y[i,1:J,1] <- rmultinom(1, N[i,1], cp)[1:J]

  for(t in 1:(T-1)) {
      S[i,t] <- rbinom(1, N[i,t], omega)
      G[i,t] <- rpois(1, gamma)
      N[i,t+1] <- S[i,t] + G[i,t]
      y[i,1:J,t+1] <- rmultinom(1, N[i,t+1], cp)[1:J]
      }
}
y <- matrix(y, M)
```

```
#Make a covariate
sc <- data.frame(x1 = rnorm(M))

umf <- unmarkedFrameDSO(y = y, siteCovs=sc, numPrimary=T, dist.breaks=db,
                        survey="line", unitsIn="m", tlength=rep(1, M))

(fit <- distsampOpen(~x1, ~1, ~1, ~1, data = umf, K=50, keyfun="halfnorm"))

#Compare to truth
cf <- coef(fit)
data.frame(model=c(exp(cf[1]), cf[2], exp(cf[3]), plogis(cf[4]), exp(cf[5])),
           truth=c(lambda, 0, gamma, omega, sigma))

#Predict
head(predict(fit, type='lambda'))

#Check fit with parametric bootstrap
pb <- parboot(fit, nsims=15)
plot(pb)

# Empirical Bayes estimates of abundance for each site / year
re <- ranef(fit)
plot(re, layout=c(10,5), xlim=c(-1, 10))


## End(Not run)
```

`fitList` *constructor of unmarkedFitList objects*

Description

Organize models for model selection or model-averaged prediction.

Usage

```
fitList(..., fits, autoNames=c("object", "formula"))
```

Arguments

`...`	Fitted models. Preferrably named.
`fits`	An alternative way of providing the models. A (preferrably named) list of fitted models.
`autoNames`	Option to change the names `unmarked` assigns to models if you don't name them yourself. If `autoNames="object"`, models in the `fitList` will be named based on their R object names. If `autoNames="formula"`, the models will instead be named based on their formulas. This is not possible for some model types.

Note

Two requirements exist to conduct AIC-based model-selection and model-averaging in unmarked. First, the data objects (ie, unmarkedFrames) must be identical among fitted models. Second, the response matrix must be identical among fitted models after missing values have been removed. This means that if a response value was removed in one model due to missingness, it needs to be removed from all models.

Author(s)

Richard Chandler <rbchan@uga.edu>

Examples

```
data(linetran)
(dbreaksLine <- c(0, 5, 10, 15, 20))
lengths <- linetran$Length * 1000

ltUMF <- with(linetran, {
unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
siteCovs = data.frame(Length, area, habitat), dist.breaks = dbreaksLine,
tlength = lengths, survey = "line", unitsIn = "m")
})

fm1 <- distsamp(~ 1 ~1, ltUMF)
fm2 <- distsamp(~ area ~1, ltUMF)
fm3 <- distsamp( ~ 1 ~area, ltUMF)

## Two methods of creating an unmarkedFitList using fitList()

# Method 1
fmList <- fitList(Null=fm1, .area=fm2, area.=fm3)

# Method 2. Note that the arugment name "fits" must be included in call.
models <- list(Null=fm1, .area=fm2, area.=fm3)
fmList <- fitList(fits = models)

# Extract coefficients and standard errors
coef(fmList)
SE(fmList)

# Model-averaged prediction
predict(fmList, type="state")

# Model selection
modSel(fmList, nullmod="Null")
```

`fitted-methods` *Methods for Function fitted in Package 'unmarked'*

Description

Extracted fitted values from a fitted model.

Usage

```
## S4 method for signature 'unmarkedFit'
fitted(object, na.rm = FALSE)
## S4 method for signature 'unmarkedFitColExt'
fitted(object, na.rm = FALSE)
## S4 method for signature 'unmarkedFitOccu'
fitted(object, na.rm = FALSE)
## S4 method for signature 'unmarkedFitOccuRN'
fitted(object, K, na.rm = FALSE)
## S4 method for signature 'unmarkedFitPCount'
fitted(object, K, na.rm = FALSE)
## S4 method for signature 'unmarkedFitDS'
fitted(object, na.rm = FALSE)
```

Arguments

object	A fitted model of appropriate S4 class
K	Integer specifying upper bound of integration.
na.rm	Logical. Should missing values be removed from data?

Value

Returns a matrix of expected values

Methods

object = "unmarkedFit" A fitted model

object = "unmarkedFitColExt" A model fit by colext

object = "unmarkedFitOccu" A model fit by occu

object = "unmarkedFitOccuRN" A model fit by occuRN

object = "unmarkedFitPCount" A model fit by pcount

object = "unmarkedFitDS" A model fit by distsamp

---

formatDistData *Bin distance data*

Description

Convert individual-level distance data to the transect-level format required by distsamp or gdistsamp

Usage

```
formatDistData(distData, distCol, transectNameCol, dist.breaks,
                      occasionCol, effortMatrix)
```

Arguments

distData	data.frame where each row is a detected individual. Must have at least 2 columns. One for distances and the other for transect names.
distCol	character, name of the column in distData that contains the distances. The distances should be numeric.
transectNameCol	character, column name containing transect names. The transect column should be a factor.
dist.breaks	numeric vector of distance interval cutpoints. Length must equal J+1.
occasionCol	optional character. If transects were visited more than once, this can be used to format data for gdistsamp. It is the name of the column in distData that contains the occasion numbers. The occasion column should be a factor.
effortMatrix	optional matrix of 1 and 0s that is M * T in size and will allow for the insertion of NAs where the matrix = 0, indicating that a survey was not completed. When not supplied a matrix of all 1s is created since it is assumed all surveys were completed.

Details

This function creates a site (M) by distance interval (J) response matrix from a data.frame containing the detection distances for each individual and the transect names. Alternatively, if each transect was surveyed T times, the resulting matrix is M x JT, which is the format required by gdistsamp, seeunmarkedFrameGDS.

Value

An M x J or M x JT matrix containing the binned distance data. Transect names will become rownames and colnames will describe the distance intervals.

Note

It is important that the factor containing transect names includes levels for all the transects surveyed, not just those with >=1 detection. Likewise, if transects were visited more than once, the factor containing the occasion numbers should include levels for all occasions. See the example for how to add levels to a factor.

See Also

distsamp, unmarkedFrame

Examples

```
# Create a data.frame containing distances of animals detected
# along 4 transects.
dat <- data.frame(transect=gl(4,5, labels=letters[1:4]),
                  distance=rpois(20, 10))
dat

# Look at your transect names.
levels(dat$transect)

# Suppose that you also surveyed a transect named "e" where no animals were
# detected. You must add it to the levels of dat$transect
levels(dat$transect) <- c(levels(dat$transect), "e")
levels(dat$transect)

# Distance cut points defining distance intervals
cp <- c(0, 8, 10, 12, 14, 18)

# Create formated response matrix
yDat <- formatDistData(dat, "distance", "transect", cp)
yDat

# Now you could merge yDat with transect-level covariates and
# then use unmarkedFrameDS to prepare data for distsamp


## Example for data from multiple occasions

dat2 <- data.frame(distance=1:100, site=gl(5, 20),
                   visit=factor(rep(1:4, each=5)))
cutpt <- seq(0, 100, by=25)
y2 <- formatDistData(dat2, "distance", "site", cutpt, "visit")
umf <- unmarkedFrameGDS(y=y2, numPrimary=4, survey="point",
                        dist.breaks=cutpt, unitsIn="m")
 ## Example for datda from multiple occasions with effortMatrix

 dat3 <-  data.frame(distance=1:100, site=gl(5, 20), visit=factor(rep(1:4, each=5)))
 cutpt <- seq(0, 100, by=25)

 effortMatrix <- matrix(ncol=4, nrow=5, rbinom(20,1,0.8))

 y3 <- formatDistData(dat2, "distance", "site", cutpt, "visit", effortMatrix)
```

---

formatMult *Create unmarkedMultFrame from Long Format Data Frame*

Description

This convenience function converts multi-year data in long format to unmarkedMultFrame Object. See Details for more information.

Usage

```
formatMult(df.in)
```

Arguments

df.in	a data.frame appropriately formatted (see Details).

Details

df.in is a data frame with columns formatted as follows:

Column 1 = year number
Column 2 = site name or number
Column 3 = julian date or chronological sample number during year
Column 4 = observations (y)
Column 5 – Final Column = covariates

Note that if the data is already in wide format, it may be easier to create an unmarkedMultFrame object directly with a call to unmarkedMultFrame.

Value

unmarkedMultFrame object

---

formatWideLong *Convert between wide and long data formats.*

Description

Convert a data.frame between wide and long formats.

Usage

```
formatWide(dfin, sep = ".", obsToY, type, ...)
formatLong(dfin, species = NULL, type, ...)
```

Arguments

dfin	A data.frame to be reformatted.
sep	A seperator of column names in wide format.
obsToY	Optional matrix specifying relationship between covariate column structure and response matrix structure.
type	Type of unmarkedFrame to create?
species	Character name of species response column
...	Further arguments to the unmarkedFrame* constructor functions

Details

Note that not all possible unmarkedFrame* classes have been tested with these functions. Multinomial data sets (e.g., removal, double-observer, capture-recapture) are almost certainly easier to enter directly to the constructor function and are not supported by formatLong or formatWide.

In order for these functions to work, the columns of dfin need to be in the correct order. formatLong requires that the columns are in the following scheme:

1. site name or number.
2. date or observation number.
3. response variable (detections, counts, etc).
4. The remaining columns are observation-level covariates.

formatWide requires particular names for the columns. The column order for formatWide is

1. (optional) site name, named "site".
2. response, named "y.1", "y.2", ..., "y.J".
3. columns of site-level covariates, each with a relevant name per column.
4. groups of columns of observation-level covariates, each group having the name form "someObsCov.1", "someObsCov.2", ..., "someObsCov.J".

Value

A data.frame

See Also

csvToUMF

---

frogs *2001 Delaware North American Amphibian Monitoring Program Data*

Description

frogs contains NAAMP data for Pseudacris feriarum (pfer) and Pseudacris crucifer (pcru) in 2001.

Usage

```
data(frogs)
```

Format

pcru.y matrix of observed calling indices for pcru

pcru.bin matrix of detections for pcru

pcru.data array of covariates measured at the observation-level for pcru

pfer.y matrix of observed calling indices for pfer

pfer.bin matrix of detections for pfer

pfer.data array of covariates measured at the observation-level for pfer

Details

The rows of pcru.y, pcru.bin, pfer.y, and pfer.bin correspond to sites and columns correspond to visits to each site. The first 2 dimensions of pfer.data and pcru.data are matrices of covariates that correspond to the observation matrices (sites $\times$ observation), with the 3rd dimension corresponding to separate covariates.

Source

https://www.pwrc.usgs.gov/naamp/

References

Mossman MJ, Weir LA. North American Amphibian Monitoring Program (NAAMP). Amphibian Declines: the conservation status of United States species. University of California Press, Berkeley, California, USA. 2005:307-313.

Examples

```
data(frogs)
str(pcru.data)
```

---

gdistremoval *Fit the combined distance and removal model of Amundson et al. (2014).*

Description

Fit the model of Amundson et al. (2014) to point count datasets containing both distance and time of observation data. The Amundson et al. (2014) model is extended to account for temporary emigration by estimating an additional availability probability if multiple counts at a site are available. Abundance can be modeled as a Poisson, negative binomial, or Zero-inflated Poisson. Multiple distance sampling key functions are also available.

Usage

```
gdistremoval(lambdaformula=~1, phiformula=~1, removalformula=~1,
  distanceformula=~1, data, keyfun=c("halfnorm", "exp", "hazard", "uniform"),
 output=c("abund", "density"), unitsOut=c("ha", "kmsq"), mixture=c('P', 'NB', 'ZIP'),
  K, starts, method = "BFGS", se = TRUE, engine=c("C","TMB"), threads=1, ...)
```

Arguments

`lambdaformula`	A right-hand side formula describing the abundance covariates
`phiformula`	A right-hand side formula describing the availability covariates
`removalformula`	A right-hand side formula describing removal probability covariates
`distanceformula`	A right-hand side formula describing the detection function covariates

data	An object of class unmarkedFrameGDR
keyfun	One of the following detection functions: "halfnorm", "hazard", "exp", or "uniform"
output	Model either "abund" or "density"
unitsOut	Units of density. Either "ha" or "kmsq" for hectares and square kilometers, respectively
mixture	Either "P", "NB", or "ZIP" for the Poisson, negative binomial, and Zero-inflated Poisson models of abundance
K	An integer value specifying the upper bound used in the integration
starts	A numeric vector of starting values for the model parameters
method	Optimization method used by optim
se	logical specifying whether or not to compute standard errors
engine	Either "C" to use C++ code or "TMB" to use TMB for optimization
threads	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If threads=1 (the default), OpenMP is disabled
...	Additional arguments to optim, such as lower and upper bounds

Value

An object of class unmarkedFitGDR

Author(s)

Ken Kellner <contact@kenkellner.com>

References

Amundson, C.L., Royle, J.A. and Handel, C.M., 2014. A hierarchical model combining distance sampling and time removal to estimate detection probability during avian point counts. The Auk 131: 476-494.

See Also

unmarkedFrameGDR, gdistsamp, gmultmix

gdistsamp *Fit the generalized distance sampling model of Chandler et al. (2011).*

Description

Extends the distance sampling model of Royle et al. (2004) to estimate the probability of being available for detection. Also allows abundance to be modeled using the negative binomial distribution.

Usage

```
gdistsamp(lambdaformula, phiformula, pformula, data, keyfun =
c("halfnorm", "exp", "hazard", "uniform"), output = c("abund",
"density"), unitsOut = c("ha", "kmsq"), mixture = c("P", "NB", "ZIP"), K,
starts, method = "BFGS", se = TRUE, engine=c("C","R"), rel.tol=1e-4, threads=1, ...)
```

Arguments

lambdaformula	A right-hand side formula describing the abundance covariates.
phiformula	A right-hand side formula describing the availability covariates.
pformula	A right-hand side formula describing the detection function covariates.
data	An object of class unmarkedFrameGDS
keyfun	One of the following detection functions: "halfnorm", "hazard", "exp", or "uniform." See details.
output	Model either "density" or "abund"
unitsOut	Units of density. Either "ha" or "kmsq" for hectares and square kilometers, respectively.
mixture	Either "P", "NB", or "ZIP" for the Poisson, negative binomial, or zero-inflated Poisson models of abundance.
K	An integer value specifying the upper bound used in the integration.
starts	A numeric vector of starting values for the model parameters.
method	Optimization method used by optim.
se	logical specifying whether or not to compute standard errors.
engine	Either "C" to use fast C++ code or "R" to use native R code during the optimization.
rel.tol	relative accuracy for the integration of the detection function. See integrate. You might try adjusting this if you get an error message related to the integral. Alternatively, try providing different starting values.
threads	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If threads=1 (the default), OpenMP is disabled.
...	Additional arguments to optim, such as lower and upper bounds

Details

This model extends the model of Royle et al. (2004) by estimating the probability of being available for detection ϕ. This effectively relaxes the assumption that $g(0) = 1$. In other words, inividuals at a distance of 0 are not assumed to be detected with certainty. To estimate this additional parameter, replicate distance sampling data must be collected at each transect. Thus the data are collected at i = 1, 2, ..., R transects on t = 1, 2, ..., T occassions. As with the model of Royle et al. (2004), the detections must be binned into distance classes. These data must be formatted in a matrix with R rows, and JT columns where J is the number of distance classses. See `unmarkedFrameGDS` for more information.

Value

An object of class unmarkedFitGDS.

Note

If you aren't interested in estimating phi, but you want to use the negative binomial distribution, simply set numPrimary=1 when formatting the data.

Note

You cannot use obsCovs, but you can use yearlySiteCovs (a confusing name since this model isn't for multi-year data. It's just a hold-over from the colext methods of formatting data upon which it is based.)

Author(s)

Richard Chandler <rbchan@uga.edu>

References

Royle, J. A., D. K. Dawson, and S. Bates. 2004. Modeling abundance effects in distance sampling. *Ecology* 85:1591-1597.

Chandler, R. B, J. A. Royle, and D. I. King. 2011. Inference about density and temporary emigration in unmarked populations. *Ecology* 92:1429–1435.

See Also

`distsamp`

Examples

```
# Simulate some line-transect data

set.seed(36837)

R <- 50 # number of transects
T <- 5  # number of replicates
```

```
strip.width <- 50
transect.length <- 100
breaks <- seq(0, 50, by=10)

lambda <- 5 # Abundance
phi <- 0.6  # Availability
sigma <- 30 # Half-normal shape parameter

J <- length(breaks)-1
y <- array(0, c(R, J, T))
for(i in 1:R) {
    M <- rpois(1, lambda) # Individuals within the 1-ha strip
    for(t in 1:T) {
        # Distances from point
        d <- runif(M, 0, strip.width)
        # Detection process
        if(length(d)) {
            cp <- phi*exp(-d^2 / (2 * sigma^2)) # half-normal w/ g(0)<1
            d <- d[rbinom(length(d), 1, cp) == 1]
            y[i,,t] <- table(cut(d, breaks, include.lowest=TRUE))
            }
        }
    }
y <- matrix(y, nrow=R) # convert array to matrix

# Organize data
umf <- unmarkedFrameGDS(y = y, survey="line", unitsIn="m",
    dist.breaks=breaks, tlength=rep(transect.length, R), numPrimary=T)
summary(umf)


# Fit the model
m1 <- gdistsamp(~1, ~1, ~1, umf, output="density", K=50)

summary(m1)


backTransform(m1, type="lambda")
backTransform(m1, type="phi")
backTransform(m1, type="det")

## Not run:
# Empirical Bayes estimates of abundance at each site
re <- ranef(m1)
plot(re, layout=c(10,5), xlim=c(-1, 20))

## End(Not run)
```

---

getB-methods *Methods for Function getB in Package 'unmarked'*

Description

Methods for function getB in Package 'unmarked'. These methods return a matrix of probabilities detections were certain for occupancy models that account for false positives.

---

getFP-methods *Methods for Function getFP in Package 'unmarked'*

Description

Methods for function getFP in Package 'unmarked'. These methods return a matrix of false positive detection probabilities.

---

getP-methods *Methods for Function getP in Package 'unmarked'*

Description

Methods for function getP in Package 'unmarked'. These methods return a matrix of the back-transformed detection parameter (p the detection probability or λ the detection rate, depending on the model). The matrix is of dimension MxJ, with M the number of sites and J the number of sampling periods; or of dimension MxJT for models with multiple primary periods T.

Methods

signature(object = "unmarkedFit") A fitted model object

signature(object = "unmarkedFitDS") A fitted model object

signature(object = "unmarkedFitMPois") A fitted model object

signature(object = "unmarkedFitGMM") A fitted model object

signature(object = "unmarkedFitOccuCOP") With unmarkedFitOccuCOP the object of a model fitted with occuCOP. Returns a matrix of λ the detection rate.

gf *Green frog count index data*

Description

Multinomial calling index data.

Usage

```
data(gf)
```

Format

A list with 2 components

gf.data 220 x 3 matrix of count indices

gf.obs list of covariates

References

Royle, J. Andrew, and William A. Link. 2005. A General Class of Multinomial Mixture Models for Anuran Calling Survey Data. Ecology 86, no. 9: 2505–2512.

Examples

```
data(gf)
str(gf.data)
str(gf.obs)
```

gmultmix *Generalized multinomial N-mixture model*

Description

A three level hierarchical model for designs involving repeated counts that yield multinomial outcomes. Possible data collection methods include repeated removal sampling and double observer sampling. The three model parameters are abundance, availability, and detection probability.

Usage

```
gmultmix(lambdaformula, phiformula, pformula, data, mixture = c("P", "NB", "ZIP"), K,
         starts, method = "BFGS", se = TRUE, engine=c("C","R"), threads=1, ...)
```

Arguments

`lambdaformula`	Righthand side (RHS) formula describing abundance covariates
`phiformula`	RHS formula describing availability covariates
`pformula`	RHS formula describing detection covariates
`data`	An object of class unmarkedFrameGMM
`mixture`	Either "P", "NB", or "ZIP" for the Poisson, negative binomial, or zero-inflated Poisson models of abundance
`K`	The upper bound of integration
`starts`	Starting values
`method`	Optimization method used by `optim`
`se`	Logical. Should standard errors be calculated?
`engine`	Either "C" to use fast C++ code or "R" to use native R code during the optimization.
`threads`	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If `threads=1` (the default), OpenMP is disabled.
`...`	Additional arguments to optim, such as lower and upper bounds

Details

The latent transect-level super-population abundance distribution $f(M|\theta)$ can be set as a Poisson, negative binomial, or zero-inflated Poisson random variable, depending on the setting of the `mixture` argument. `mixture = "P"`, `mixture = "NB"`, and `mixture = "ZIP"` select the Poisson, negative binomial, and zero-inflated Poisson distributions respectively. The mean of M_i is λ_i. If $M_i \sim NB$, then an additional parameter, α, describes dispersion (lower α implies higher variance). If $M_i \sim ZIP$, then an additional zero-inflation parameter ψ is estimated.

The number of individuals available for detection at time j is a modeled as binomial: $N_{ij} \sim Binomial(M_i, \phi_{\mathbf{ij}})$.

The detection process is modeled as multinomial: $\mathbf{y_{it}} \sim Multinomial(N_{it}, \pi_{it})$, where π_{ijt} is the multinomial cell probability for plot i at time t on occasion j.

Cell probabilities are computed via a user-defined function related to the sampling design. Alternatively, the default functions `removalPiFun` or `doublePiFun` can be used for equal-interval removal sampling or double observer sampling. Note that the function for computing cell probabilites is specified when setting up the data using `unmarkedFrameGMM`.

Parameters λ, ϕ and p can be modeled as linear functions of covariates using the log, logit and logit links respectively.

Value

An object of class unmarkedFitGMM.

Note

In the case where availability for detection is due to random temporary emigration, population density at time j, D(i,j), can be estimated by N(i,j)/plotArea.

This model is also applicable to sampling designs in which the local population size is closed during the J repeated counts, and availability is related to factors such as the probability of vocalizing. In this case, density can be estimated by M(i)/plotArea.

If availability is a function of both temporary emigration and other processess such as song rate, then density cannot be directly estimated, but inference about the super-population size, M(i), is possible.

Three types of covariates can be supplied, site-level, site-by-year-level, and observation-level. These must be formatted correctly when organizing the data with unmarkedFrameGPC

Author(s)

Richard Chandler <rbchan@uga.edu> and Andy Royle

References

Royle, J. A. (2004) Generalized estimators of avian abundance from count survey data. *Animal Biodiversity and Conservation* 27, pp. 375–386.

Chandler, R. B., J. A. Royle, and D. I. King. 2011. Inference about density and temporary emigration in unmarked populations. Ecology 92:1429-1435.

See Also

unmarkedFrameGMM for setting up the data and metadata. multinomPois for surveys where no secondary sampling periods were used. Example functions to calculate multinomial cell probabilities are described piFuns

Examples

```
# Simulate data using the multinomial-Poisson model with a
# repeated constant-interval removal design.

n <- 100  # number of sites
T <- 4    # number of primary periods
J <- 3    # number of secondary periods

lam <- 3
phi <- 0.5
p <- 0.3

#set.seed(26)
y <- array(NA, c(n, T, J))
M <- rpois(n, lam)          # Local population size
N <- matrix(NA, n, T)       # Individuals available for detection

for(i in 1:n) {
```

```
    N[i,] <- rbinom(T, M[i], phi)
    y[i,,1] <- rbinom(T, N[i,], p)    # Observe some
    Nleft1 <- N[i,] - y[i,,1]         # Remove them
    y[i,,2] <- rbinom(T, Nleft1, p)   # ...
    Nleft2 <- Nleft1 - y[i,,2]
    y[i,,3] <- rbinom(T, Nleft2, p)
    }

y.ijt <- cbind(y[,1,], y[,2,], y[,3,], y[,4,])


umf1 <- unmarkedFrameGMM(y=y.ijt, numPrimary=T, type="removal")

(m1 <- gmultmix(~1, ~1, ~1, data=umf1, K=30))

backTransform(m1, type="lambda")        # Individuals per plot
backTransform(m1, type="phi")           # Probability of being avilable
(p <- backTransform(m1, type="det"))    # Probability of detection
p <- coef(p)

# Multinomial cell probabilities under removal design
c(p, (1-p) * p, (1-p)^2 * p)

# Or more generally:
head(getP(m1))

# Empirical Bayes estimates of super-population size
re <- ranef(m1)
plot(re, layout=c(5,5), xlim=c(-1,20), subset=site%in%1:25)
```

---

goccu *Fit multi-scale occupancy models*

Description

Fit multi-scale occupancy models as described in Nichols et al. (2008) to repeated presence-absence data collected using the robust design. This model allows for inference about occupancy, availability, and detection probability.

Usage

```
goccu(psiformula, phiformula, pformula, data, linkPsi = c("logit", "cloglog"),
      starts, method = "BFGS", se = TRUE, ...)
```

Arguments

psiformula	Right-hand sided formula describing occupancy covariates

phiformula	Right-hand sided formula describing availability covariates
pformula	Right-hand sided formula for detection probability covariates
data	An object of class unmarkedFrameGOccu or unmarkedMultFrame
linkPsi	Link function for the occupancy model. Options are "logit" for the standard occupancy model or "cloglog" for the complimentary log-log link, which relates occupancy to site-level abundance.
starts	Starting values
method	Optimization method used by optim
se	Logical. Should standard errors be calculated?
...	Additional arguments to optim, such as lower and upper bounds

Details

Primary periods could represent spatial or temporal sampling replicates. For example, you could have several spatial sub-units within each site, where each sub-unit was then sampled repeatedly. This is a frequent design for eDNA studies. Or, you could have multiple primary periods of sampling at each site (conducted at different times within a season), each of which contains several secondary sampling periods. In both cases the robust design structure can be used to estimate an availability probability in addition to detection probability. See Kery and Royle (2015) 10.10 for more details.

Value

An object of class unmarkedFitGOccu

Author(s)

Ken Kellner <contact@kenkellner.com>

References

Kery, M., & Royle, J. A. (2015). Applied hierarchical modeling in ecology: Volume 1: Prelude and static models. Elsevier Science.

Nichols, J. D., Bailey, L. L., O'Connell Jr, A. F., Talancy, N. W., Campbell Grant, E. H., Gilbert, A. T., Annand E. M., Husband, T. P., & Hines, J. E. (2008). Multi-scale occupancy estimation and modelling using multiple detection methods. Journal of Applied Ecology, 45(5), 1321-1329.

See Also

occu, colext, unmarkedMultFrame, unmarkedFrameGOccu

Examples

```
set.seed(123)
M <- 100
T <- 5
J <- 4
```

```
psi <- 0.5
phi <- 0.3
p <- 0.4

z <- rbinom(M, 1, psi)
zmat <- matrix(z, nrow=M, ncol=T)

zz <- rbinom(M*T, 1, zmat*phi)
zz <- matrix(zz, nrow=M, ncol=T)

zzmat <- zz[,rep(1:T, each=J)]
y <- rbinom(M*T*J, 1, zzmat*p)
y <- matrix(y, M, J*T)
umf <- unmarkedMultFrame(y=y, numPrimary=T)

## Not run:
  mod <- goccu(psiformula = ~1, phiformula = ~1, pformula = ~1, umf)
  plogis(coef(mod))

## End(Not run)
```

---

gpcount *Generalized binomial N-mixture model for repeated count data*

Description

Fit the model of Chandler et al. (2011) to repeated count data collected using the robust design. This model allows for inference about population size, availability, and detection probability.

Usage

```
gpcount(lambdaformula, phiformula, pformula, data,
mixture = c("P", "NB", "ZIP"), K, starts, method = "BFGS", se = TRUE,
engine = c("C", "R"), threads=1, ...)
```

Arguments

lambdaformula	Right-hand sided formula describing covariates of abundance.
phiformula	Right-hand sided formula describing availability covariates
pformula	Right-hand sided formula for detection probability covariates
data	An object of class unmarkedFrameGPC
mixture	Either "P", "NB", or "ZIP" for Poisson, negative binomial, or zero-inflated Poisson distributions
K	The maximum possible value of M, the super-population size.
starts	Starting values

method	Optimization method used by optim
se	Logical. Should standard errors be calculated?
engine	Either "C" or "R" for the C++ or R versions of the likelihood. The C++ code is faster, but harder to debug.
threads	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If threads=1 (the default), OpenMP is disabled.
...	Additional arguments to optim, such as lower and upper bounds

Details

The latent transect-level super-population abundance distribution $f(M|\theta)$ can be set as either a Poisson, negative binomial, or zero-inflated Poisson random variable, depending on the setting of the mixture argument. The expected value of M_i is λ_i. If $M_i \sim NB$, then an additional parameter, α, describes dispersion (lower α implies higher variance). If $M_i \sim ZIP$, then an additional zero-inflation parameter ψ is estimated.

The number of individuals available for detection at time j is a modeled as binomial: $N_{ij} \sim Binomial(M_i, \phi_{\mathbf{ij}})$.

The detection process is also modeled as binomial: $y_{ikj} \sim Binomial(N_{ij}, p_{ikj})$.

Parameters λ, ϕ and p can be modeled as linear functions of covariates using the log, logit and logit links respectively.

Value

An object of class unmarkedFitGPC

Note

In the case where availability for detection is due to random temporary emigration, population density at time j, D(i,j), can be estimated by N(i,j)/plotArea.

This model is also applicable to sampling designs in which the local population size is closed during the J repeated counts, and availability is related to factors such as the probability of vocalizing. In this case, density can be estimated by M(i)/plotArea.

If availability is a function of both temporary emigration and other processess such as song rate, then density cannot be directly estimated, but inference about the super-population size, M(i), is possible.

Three types of covariates can be supplied, site-level, site-by-year-level, and observation-level. These must be formatted correctly when organizing the data with unmarkedFrameGPC

Author(s)

Richard Chandler <rbchan@uga.edu>

References

Royle, J. A. 2004. N-Mixture models for estimating population size from spatially replicated counts. *Biometrics* 60:108–105.

Chandler, R. B., J. A. Royle, and D. I. King. 2011. Inference about density and temporary emigration in unmarked populations. Ecology 92:1429-1435.

See Also

gmultmix, gdistsamp, unmarkedFrameGPC

Examples

```
set.seed(54)

nSites <- 20
nVisits <- 4
nReps <- 3

lambda <- 5
phi <- 0.7
p <- 0.5

M <- rpois(nSites, lambda) # super-population size

N <- matrix(NA, nSites, nVisits)
y <- array(NA, c(nSites, nReps, nVisits))
for(i in 1:nVisits) {
    N[,i] <- rbinom(nSites, M, phi) # population available during vist j
}
colMeans(N)

for(i in 1:nSites) {
    for(j in 1:nVisits) {
        y[i,,j] <- rbinom(nReps, N[i,j], p)
    }
}

ym <- matrix(y, nSites)
ym[1,] <- NA
ym[2, 1:nReps] <- NA
ym[3, (nReps+1):(nReps+nReps)] <- NA
umf <- unmarkedFrameGPC(y=ym, numPrimary=nVisits)

## Not run:
fmu <- gpcount(~1, ~1, ~1, umf, K=40, control=list(trace=TRUE, REPORT=1))

backTransform(fmu, type="lambda")
backTransform(fmu, type="phi")
backTransform(fmu, type="det")

## End(Not run)
```

imputeMissing *A function to impute missing entries in continuous obsCovs*

Description

This function uses an ad-hoc averaging approach to impute missing entries in obsCovs. The missing entry is replaced by an average of the average for the site and the average for the visit number.

Usage

```
imputeMissing(umf, whichCovs = seq(length=ncol(obsCovs(umf))))
```

Arguments

umf	The data set who's obsCovs are being imputed.
whichCovs	An integer vector giving the indices of the covariates to be imputed. This defaults to all covariates in obsCovs.

Value

A version of umf that has the requested obsCovs imputed.

Author(s)

Ian Fiske

Examples

```
data(frogs)
pcru.obscovs <- data.frame(MinAfterSunset=as.vector(t(pcru.data[,,1])),
     Wind=as.vector(t(pcru.data[,,2])),
     Sky=as.vector(t(pcru.data[,,3])),
     Temperature=as.vector(t(pcru.data[,,4])))
pcruUMF <- unmarkedFrameOccu(y = pcru.bin, obsCovs = pcru.obscovs)
pcruUMF.i1 <- imputeMissing(pcruUMF)
pcruUMF.i2 <- imputeMissing(pcruUMF, whichCovs = 2)
```

issj *Distance-sampling data for the Island Scrub Jay (*Aphelocoma insularis*)*

Description

Data were collected at 307 survey locations ("point transects") on Santa Cruz Island, California during the Fall of 2008. The distance data are binned into 3 distance intervals [0-100], (100-200], and (200-300]. The coordinates of the survey locations as well as 3 habitat covariates are also included.

Usage

```
data(issj)
```

Format

A data frame with 307 observations on the following 8 variables.

`issj[0-100]` Number of individuals detected within 100m

`issj(100-200]` Detections in the interval (100-200m]

`issj(200-300]` Detections in the interval (200-300m]

`x` Easting (meters)

`y` Northing (meters)

`elevation` Elevation in meters

`forest` Forest cover

`chaparral` Chaparral cover

References

Sillett, S. and Chandler, R.B. and Royle, J.A. and Kery, M. and Morrison, S.A. In Press. Hierarchical distance sampling models to estimate population size and habitat-specific abundance of an island endemic. *Ecological Applications*

See Also

Island-wide covariates are also available `cruz`

Examples

```
data(issj)
str(issj)
head(issj)

umf <- unmarkedFrameDS(y=as.matrix(issj[,1:3]), siteCovs=issj[,6:8],
    dist.breaks=c(0,100,200,300), unitsIn="m", survey="point")
summary(umf)
```

`jay` *European Jay data from the Swiss Breeding Bird Survey 2002*

Description

The Swiss breeding bird survey ("Monitoring Haufige Brutvogel" MHB) has monitored the populations of 150 common species since 1999. The MHB sample consists of 267 1-km squares that are laid out as a grid across Switzerland. Fieldwork is conducted by about 200 skilled birdwatchers, most of them volunteers. Avian populations are monitored using a simplified territory mapping protocol, where each square is surveyed up to three times during the breeding season (only twice above the tree line). Surveys are conducted along a transect that does not change over the years.

The list `jay` has the data for European Jay territories for 238 sites surveyed in 2002.

Usage

```
data("jay")
```

Format

`jay` is a list with 3 elements:

caphist a data frame with rows for 238 sites and columns for each of the observable detection histories. For the sites visited 3 times, these are `"100"`, `"010"`, `"001"`, `"110"`, `"101"`, `"011"`, `"111"`. Sites visited twice have `"10x"`, `"01x"`, `"11x"`.

Each row gives the number of territories with the corresponding detection history, with NA for the detection histories not applicable: sites visited 3 times have NAs in the last 3 columns while those visited twice have NAs in the first 7 columns.

sitescovs a data frame with rows for 238 sites, and the following columns:

1. elev : the mean elevation of the quadrat, m.
2. length : the length of the route walked in the quadrat, km.
3. forest : percentage forest cover.

covinfo a data frame with rows for 238 sites, and the following columns:

1. x, y : the coordinates of the site.
2. date1, date2, date3 : the Julian date of the visit, with 1 April = 1. Sites visited twice have NA in the 3rd column.
3. dur1, dur2, dur3 : the duration of the survey, mins. For 10 visits the duration is not available, so there are additional NAs in these columns.

Note

In previous versions, `jay` had additional information not required for the analysis, and a data frame with essentially the same information as the `Switzerland` data set.

Source

Swiss Ornithological Institute

References

Royle, J.A., Kery, M., Gauthier, R., Schmid, H. (2007) Hierarchical spatial models of abundance and occurrence from imperfect survey data. *Ecological Monographs*, 77, 465-481.

Kery & Royle (2016) *Applied Hierarachical Modeling in Ecology* Section 7.9

Examples

```
data(jay)
str(jay)

# Carry out a simple analysis, without covariates:
# Create a customised piFun (see ?piFun for details)
crPiFun <- function(p) {
   p1 <- p[,1] # Extract the columns of the p matrix, one for
   p2 <- p[,2] #   each of J = 3 sample occasions
   p3 <- p[,3]
   cbind(      # define multinomial cell probabilities:
      "100" = p1 * (1-p2) * (1-p3),
      "010" = (1-p1) * p2 * (1-p3),
      "001" = (1-p1) * (1-p2) * p3,
      "110" = p1 * p2 * (1-p3),
      "101" = p1 * (1-p2) * p3,
      "011" = (1-p1) * p2 * p3,
      "111" = p1 * p2 * p3,
      "10x" = p1*(1-p2),
      "01x" = (1-p1)*p2,
      "11x" = p1*p2)
}
# Build the unmarkedFrame object
mhb.umf <- unmarkedFrameMPois(y=as.matrix(jay$caphist),
  obsToY=matrix(1, 3, 10), piFun="crPiFun")
# Fit a model
( fm1 <- multinomPois(~1 ~1, mhb.umf) )
```

`lambda2psi` *Convert Poisson mean (lambda) to probability of occurrence (psi).*

Description

Abundance and occurrence are fundamentally related.

Usage

```
lambda2psi(lambda)
```

Arguments

`lambda`	Numeric vector with values >= 0

Value

A vector of psi values of the same length as lambda.

See Also

pcount, multinomPois, distsamp

Examples

```
lambda2psi(0:5)
```

---

linearComb-methods *Methods for Function linearComb in Package 'unmarked'*

Description

Methods for function linearComb in Package 'unmarked'

Methods

obj = "unmarkedEstimate", coefficients = "matrixOrVector" Typically called internally

obj = "unmarkedFit", coefficients = "matrixOrVector" Returns linear combinations of parameters from a fitted model. Coefficients are supplied through coefficients. The required argument type specifies which model estimate to use. You can use names(fittedmodel) to view possible values for the type argument.

Examples

```
data(ovendata)
ovenFrame <- unmarkedFrameMPois(ovendata.list$data,
siteCovs=as.data.frame(scale(ovendata.list$covariates[,-1])), type = "removal")
fm <- multinomPois(~ 1 ~ ufc + trba, ovenFrame)
linearComb(fm, c(1, 0.5, 0.5), type = "state")
linearComb(fm, matrix(c(1, 0.5, 0.5, 1, 0, 0, 1, 0, 0.5), 3, 3,
  byrow=TRUE), type="state")
```

`linetran` *Simulated line transect data*

Description

Response matrix of animals detected in four distance classes plus transect lengths and two covariates.

Usage

```
data(linetran)
```

Format

A data frame with 12 observations on the following 7 variables.

`dc1` Counts in distance class 1 [0-5 m)

`dc2` Counts in distance class 2 [5-10 m)

`dc3` Counts in distance class 3 [10-15 m)

`dc4` Counts in distance class 4 [15-20 m)

`Length` Transect lengths in km

`area` Numeric covariate

`habitat` a factor with levels `A` and `B`

Examples

```
data(linetran)
linetran

# Format for distsamp()
ltUMF <- with(linetran, {
        unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
        siteCovs = data.frame(Length, area, habitat),
        dist.breaks = c(0, 5, 10, 15, 20),
        tlength = linetran$Length * 1000, survey = "line", unitsIn = "m")
        })
```

`makePiFuns` *Create functions to compute multinomial cell probabilities*

Description

These are factory functions that generate piFuns with the required defaults, which are enclosed within the environment of the piFun. See the main entry for piFuns.

Usage

```
makeRemPiFun(times)
makeCrPiFun(nOcc)
makeCrPiFunMb(nOcc)
makeCrPiFunMh(nOcc)
```

Arguments

`times`	a vector of times for each interval, `length(times)` is the number of survey occasions; can be all 1's if times are the same.
`nOcc`	the number of survey occasions

Details

`makeRemPiFun` produces a piFun for a removal model with the required number of occasions and potentially varying time intervals. The input to the piFun must be probabilities *per unit time*. This is a generalisation of the piFun in the Examples section of piFuns.

`makeCrPiFun` produces a piFun for a standard capture-recapture model, M0, Mt or Mx. Probabilities of detection may vary across occasions. See Kery & Royle (2016) section 7.8.1.

`makeCrPiFunMb` produces a piFun for a capture-recapture model with a behavioral response after the first capture, Mb. Probabilities of detection are constant across occasions. The first column is the probability of detection for animals not caught before, column #2 is for animals after the first capture. The remaining columns are ignored. See Kery & Royle (2016) section 7.8.2.

`makeCrPiFunMh` produces a piFun for a capture-recapture model with individual heterogeneity in detection probability, Mh, using a logit-normal distribution. Probabilities of detection are constant across occasions. The first column is the mean of the logit-normal on the probability scale. Cell p[1, 2] is a value in [0, 1] which controls the spread of the distribution. The remaining cells are ignored. See Kery & Royle (2016) section 7.8.3.

Value

A piFun with the appropriate defaults.

References

Kery, M., Royle, J. A. (2016) *Applied Hierarchical Modeling in Ecology* Vol 1.

Examples

```
# Generate piFuns and check their behaviour:

# makeRemPiFun
# ============
( pRem <- matrix(0.4, nrow=5, ncol=3) )
myPi <- makeRemPiFun(times=c(2,3,5))
myPi(pRem)
ls(environment(myPi))  # See what's in the environment
environment(myPi)$times

( pRem <- matrix(runif(15), 5, 3) )
myPi(pRem)

myPi <- makeRemPiFun(c(5,3,2))
environment(myPi)$times
myPi(pRem)

# More than 3 occasions
myPi <- makeRemPiFun(c(1,2,3,5))
try(myPi(pRem))  # Error
( pRem <- matrix(runif(20), 5, 4) )
myPi(pRem)
# Probability of escaping detection
1 - rowSums(myPi(pRem))

# makeCrPiFun
# ===========
p <- matrix(0.4, 2, 3)
myPi <- makeCrPiFun(3)
myPi(p)
myPi  # Look at the function
ls(environment(myPi))
environment(myPi)$histories

p <- matrix(runif(6, 0.1, 0.9), 2, 3)  # different p's everywhere
myPi(p)

p <- matrix(runif(4*5, 0.1, 0.9), 4, 5)  # > 3 occasions
try(myPi(p))  # Error
myPi <- makeCrPiFun(5)
( tmp <- myPi(p) )
1 - rowSums(tmp) # Probability of non-capture

# makeCrPiFunMb
# =============
( pMb <- cbind(rep(0.7, 5), 0.3, NA) )
myPi <- makeCrPiFunMb(3)
myPi(pMb)

( pMb <- matrix(runif(15), 5, 3) )  # col #3 will be ignored
myPi(pMb)
```

```
# with > 3 occasions
( pMb <- matrix(runif(15), 3, 5) )
try(myPi(pMb))
myPi <- makeCrPiFunMb(5)
myPi(pMb)

# makeCrPiFunMh
# =============
pMh <- cbind(rep(0.4, 5), NA, NA)
pMh[1, 2] <- 0.3
pMh
myPi <- makeCrPiFunMh(3)
myPi(pMh)
pMh <- cbind(runif(5), NA, NA)
pMh[1, 2] <- 0.3
pMh
myPi(pMh)

# with > 3 occasions
pMh <- cbind(runif(5), NA, NA, NA, NA)
pMh[1, 2] <- 0.3
pMh
try(myPi(pMh))
myPi <- makeCrPiFunMh(5)
1 - rowSums(myPi(pMh))  # Probability of non-detection
```

`mallard` *Mallard count data*

Description

Mallard repeated count data and covariates

Usage

```
data(mallard)
```

Format

A list with 3 components

mallard.y response matrix

mallard.site site-specific covariates

mallard.obs survey-specific covariates

References

Kery, M., Royle, J. A., and Schmid, H. (2005) Modeling Avaian Abundance from Replicated Counts Using Binomial Mixture Models. *Ecological Applications* 15(4), pp. 1450–1461.

Examples

```
data(mallard)
str(mallard.y)
str(mallard.site)
str(mallard.obs)
```

---

masspcru *Massachusetts North American Amphibian Monitoring Program Data*

Description

masspcru contains NAAMP data for Pseudacris crucifer (pcru) in Massachusetts from 2001 to 2007 in the raw long format.

Usage

```
data(masspcru)
```

Format

Data frame with

SurveyYear Year of data collection.

RouteNumStopNum Stop number.

JulianDate Day of year.

Pcru Observed calling index.

MinAfterSunset Minutes after sunset of the observation.

Temperature Temperature measured during observation.

Details

These data come from the North American Amphibian Monitoring Program. Please see the reference below for more details.

Source

https://www.pwrc.usgs.gov/naamp/

References

Mossman MJ, Weir LA. North American Amphibian Monitoring Program (NAAMP). Amphibian Declines: the conservation status of United States species. University of California Press, Berkeley, California, USA. 2005:307-313.

Examples

```
data(masspcru)
str(masspcru)
```

`MesoCarnivores` *Occupancy data for coyote, red fox, and bobcat*

Description

Occupancy data and site covariates for coyote, red fox, and bobcat from 1437 camera trap sites sampled 3 times. Each sampling period represents one week. This data is a simplified form of the dataset used by Rota et al. (2016).

Usage

```
data(MesoCarnivores)
```

Format

A list with four elements:

`bobcat` A 1437x3 occupancy matrix for bobcat

`coyote` A 1437x3 occupancy matrix for coyote

`redfox` A 1437x3 occupancy matrix for red fox

`sitecovs` A data frame containing covariates for the 1437 sites, with the following columns:

- `Dist_5km` Proportion of disturbed land in 5 km radius
- `HDens_5km` Housing density in 5 km radius
- `Latitude` Latitude / 100
- `Longitude` Longitude / 100
- `People_site` Number of photos of people at site / 1000
- `Trail` 1 if camera was on trail, 0 if not

Source

Used with permission of Roland Kays and Arielle Parsons at North Carolina State University and the North Carolina Museum of Natural Sciences.

References

Rota, C.T., et al. 2016. A multi-species occupancy model for two or more interacting species. Methods in Ecology and Evolution 7: 1164-1173.

`modSel` *Model selection results from an unmarkedFitList*

Description

Model selection results from an unmarkedFitList

Arguments

`object`	an object of class "unmarkedFitList" created by the function `fitList`.
`nullmod`	optional character naming which model in the `fitList` contains results from the null model. Only used in calculation of Nagelkerke's R-squared index.

Value

A S4 object with the following slots

`Full`	data.frame with formula, estimates, standard errors and model selection information. Converge is optim convergence code. CondNum is model condition number. n is the number of sites. delta is delta AIC. cumltvWt is cumulative AIC weight. Rsq is Nagelkerke's (1991) R-squared index, which is only returned when the nullmod argument is specified.
`Names`	matrix referencing column names of estimates (row 1) and standard errors (row 2).

Note

Two requirements exist to conduct AIC-based model-selection and model-averaging in unmarked. First, the data objects (ie, unmarkedFrames) must be identical among fitted models. Second, the response matrix must be identical among fitted models after missing values have been removed. This means that if a response value was removed in one model due to missingness, it needs to be removed from all models.

Author(s)

Richard Chandler <rbchan@uga.edu>

References

Nagelkerke, N.J.D. (2004) A Note on a General Definition of the Coefficient of Determination. *Biometrika* 78, pp. 691-692.

Examples

```
data(linetran)
(dbreaksLine <- c(0, 5, 10, 15, 20))
lengths <- linetran$Length * 1000

ltUMF <- with(linetran, {
unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
siteCovs = data.frame(Length, area, habitat), dist.breaks = dbreaksLine,
tlength = lengths, survey = "line", unitsIn = "m")
})

fm1 <- distsamp(~ 1 ~1, ltUMF)
fm2 <- distsamp(~ area ~1, ltUMF)
fm3 <- distsamp( ~ 1 ~area, ltUMF)

fl <- fitList(Null=fm1, A.=fm2, .A=fm3)
fl

ms <- modSel(fl, nullmod="Null")
ms

coef(ms)                            # Estimates only
SE(ms)                              # Standard errors only
(toExport <- as(ms, "data.frame"))  # Everything
```

`multinomPois` *Multinomial-Poisson Mixtures Model*

Description

Fit the multinomial-Poisson mixture model to data collected using survey methods such as removal sampling or double observer sampling.

Usage

```
multinomPois(formula, data, starts, method = "BFGS",
   se = TRUE, engine=c("C","R","TMB"), ...)
```

Arguments

`formula`	double right-hand side formula for detection and abundance covariates, in that order.
`data`	unmarkedFrame supplying data.
`starts`	vector of starting values.
`method`	Optimization method used by `optim`.
`se`	logical specifying whether or not to compute standard errors.

engine	Either "C" to use fast C++ code or "R" to use native R code during the optimization.
...	Additional arguments to optim, such as lower and upper bounds

Details

This function takes advantage of the closed form of the integrated likelihood when a latent Poisson distribution is assumed for abundance at each site and a multinomial distribution is taken for the observation state. Many common sampling methods can be framed in this context. For example, double-observer point counts and removal sampling can be analyzed with this function by specifying the proper multinomial cell probablilities. This is done with by supplying the appropriate function (piFun) argument. removalPiFun and doublePiFun are supplied as example cell probability functions.

Value

unmarkedFit object describing the model fit.

Author(s)

Ian Fiske

References

Royle, J. A. (2004). Generalized estimators of avian abundance from count survey data. Animal Biodiversity and Conservation, 27(1), 375-386.

Royle, J. A., & Dorazio, R. M. (2006). Hierarchical Models of Animal Abundance and Occurrence. Journal Of Agricultural Biological And Environmental Statistics, 11(3), 249.

See Also

piFuns, unmarkedFrameMPois

Examples

```
# Simulate independent double observer data
nSites <- 50
lambda <- 10
p1 <- 0.5
p2 <- 0.3
cp <- c(p1*(1-p2), p2*(1-p1), p1*p2)
set.seed(9023)
N <- rpois(nSites, lambda)
y <- matrix(NA, nSites, 3)
for(i in 1:nSites) {
  y[i,] <- rmultinom(1, N[i], c(cp, 1-sum(cp)))[1:3]
}

# Fit model
```

```
observer <- matrix(c('A','B'), nSites, 2, byrow=TRUE)
umf <- unmarkedFrameMPois(y=y, obsCovs=list(observer=observer),
    type="double")
fm <- multinomPois(~observer-1 ~1, umf)

# Estimates of fixed effects
e <- coef(fm)
exp(e[1])
plogis(e[2:3])

# Estimates of random effects
re <- ranef(fm, K=20)
#ltheme <- canonical.theme(color = FALSE)
#lattice.options(default.theme = ltheme)
plot(re, layout=c(10,5))



## Real data
data(ovendata)
ovenFrame <- unmarkedFrameMPois(ovendata.list$data,
    siteCovs=as.data.frame(scale(ovendata.list$covariates[,-1])),
    type = "removal")
(fm1 <- multinomPois(~ 1 ~ ufc + trba, ovenFrame))

# Detection probability for a single pass
backTransform(fm1, type="det")

# Detection probability after 4 removal passes
rowSums(getP(fm1))

# Empirical Bayes estimates of abundance at first 25 sites
# Very low uncertainty because p is very high
plot(ranef(fm1, K=10), layout=c(10,7), xlim=c(-1, 10))
```

`multmixOpen` *Open population multinomial N-mixture model*

Description

Fit the model of Dail and Madsen (2011) and Hostetler and Chandler (2015) for designs involving repeated counts that yield multinomial outcomes. Possible data collection methods include repeated removal sampling and double observer sampling.

Usage

```
multmixOpen(lambdaformula, gammaformula, omegaformula, pformula,
    data, mixture=c("P", "NB", "ZIP"), K,
    dynamics=c("constant", "autoreg", "notrend", "trend", "ricker", "gompertz"),
```

```
fix=c("none", "gamma", "omega"), immigration=FALSE, iotaformula = ~1,
starts, method="BFGS", se=TRUE, ...)
```

Arguments

lambdaformula	Right-hand sided formula for initial abundance
gammaformula	Right-hand sided formula for recruitment rate (when dynamics is "constant", "autoreg", or "notrend") or population growth rate (when dynamics is "trend", "ricker", or "gompertz")
omegaformula	Right-hand sided formula for apparent survival probability (when dynamics is "constant", "autoreg", or "notrend") or equilibrium abundance (when dynamics is "ricker" or "gompertz")
pformula	A right-hand side formula describing the detection function covariates
data	An object of class unmarkedFrameMMO
mixture	String specifying mixture: "P", "NB", or "ZIP" for the Poisson, negative binomial, or zero-inflated Poisson distributions respectively
K	Integer defining upper bound of discrete integration. This should be higher than the maximum observed count and high enough that it does not affect the parameter estimates. However, the higher the value the slower the computation
dynamics	Character string describing the type of population dynamics. "constant" indicates that there is no relationship between omega and gamma. "autoreg" is an auto-regressive model in which recruitment is modeled as gamma*N[i,t-1]. "notrend" model gamma as lambda*(1-omega) such that there is no temporal trend. "trend" is a model for exponential growth, N[i,t] = N[i,t-1]*gamma, where gamma in this case is finite rate of increase (normally referred to as lambda). "ricker" and "gompertz" are models for density-dependent population growth. "ricker" is the Ricker-logistic model, N[i,t] = N[i,t-1]*exp(gamma*(1-N[i,t-1]/omega)), where gamma is the maximum instantaneous population growth rate (normally referred to as r) and omega is the equilibrium abundance (normally referred to as K). "gompertz" is a modified version of the Gompertz-logistic model, N[i,t] = N[i,t-1]*exp(gamma*(1-log(N[i,t-1]+1)/log(omega+1))), where the interpretations of gamma and omega are similar to in the Ricker model
fix	If "omega", omega is fixed at 1. If "gamma", gamma is fixed at 0
immigration	Logical specifying whether or not to include an immigration term (iota) in population dynamics
iotaformula	Right-hand sided formula for average number of immigrants to a site per time step
starts	Vector of starting values
method	Optimization method used by optim
se	Logical specifying whether or not to compute standard errors
...	Additional arguments to optim, such as lower and upper bounds

Details

These models generalize multinomial N-mixture models (Royle et al. 2004) by relaxing the closure assumption (Dail and Madsen 2011, Hostetler and Chandler 2015, Sollmann et al. 2015).

The models include two or three additional parameters: gamma, either the recruitment rate (births and immigrations), the finite rate of increase, or the maximum instantaneous rate of increase; omega, either the apparent survival rate (deaths and emigrations) or the equilibrium abundance (carrying capacity); and iota, the number of immigrants per site and year. Estimates of population size at each time period can be derived from these parameters, and thus so can trend estimates. Or, trend can be estimated directly using dynamics="trend".

When immigration is set to FALSE (the default), iota is not modeled. When immigration is set to TRUE and dynamics is set to "autoreg", the model will separately estimate birth rate (gamma) and number of immigrants (iota). When immigration is set to TRUE and dynamics is set to "trend", "ricker", or "gompertz", the model will separately estimate local contributions to population growth (gamma and omega) and number of immigrants (iota).

The latent abundance distribution, $f(N|\theta)$ can be set as a Poisson, negative binomial, or zero-inflated Poisson random variable, depending on the setting of the `mixture` argument, `mixture = "P"`, `mixture = "NB"`, `mixture = "ZIP"` respectively. For the first two distributions, the mean of N_i is λ_i. If $N_i \sim NB$, then an additional parameter, α, describes dispersion (lower α implies higher variance). For the ZIP distribution, the mean is $\lambda_i(1-\psi)$, where psi is the zero-inflation parameter.

For "constant", "autoreg", or "notrend" dynamics, the latent abundance state following the initial sampling period arises from a Markovian process in which survivors are modeled as $S_{it} \sim Binomial(N_{it-1}, \omega_{it})$, and recruits follow $G_{it} \sim Poisson(\gamma_{it})$. Alternative population dynamics can be specified using the `dynamics` and `immigration` arguments.

λ_i, γ_{it}, and ι_{it} are modeled using the the log link. p_{ijt} is modeled using the logit link. ω_{it} is either modeled using the logit link (for "constant", "autoreg", or "notrend" dynamics) or the log link (for "ricker" or "gompertz" dynamics). For "trend" dynamics, ω_{it} is not modeled.

The detection process is modeled as multinomial: $\mathbf{y_{it}} \sim Multinomial(N_{it}, \pi_{it})$, where π_{ijt} is the multinomial cell probability for plot i at time t on occasion j.

Options for the detection process include equal-interval removal sampling (`"removal"`), double observer sampling (`"double"`), or dependent double-observer sampling (`"depDouble"`). This option is specified when setting up the data using `unmarkedFrameMMO`. Note that unlike the related functions `multinomPois` and `gmultmix`, custom functions for the detection process (i.e., `piFuns`) are not supported. To request additional options contact the author.

Value

An object of class unmarkedFitMMO

Warning

This function can be extremely slow, especially if there are covariates of gamma or omega. Consider testing the timing on a small subset of the data, perhaps with se=FALSE. Finding the lowest value of K that does not affect estimates will also help with speed.

Note

When gamma or omega are modeled using year-specific covariates, the covariate data for the final year will be ignored; however, they must be supplied.

If the time gap between primary periods is not constant, an M by T matrix of integers should be supplied to unmarkedFrameMMO using the primaryPeriod argument.

Secondary sampling periods are optional, but can greatly improve the precision of the estimates.

Author(s)

Ken Kellner <contact@kenkellner.com>, Richard Chandler

References

Dail, D. and L. Madsen (2011) Models for Estimating Abundance from Repeated Counts of an Open Metapopulation. *Biometrics*. 67: 577-587.

Hostetler, J. A. and R. B. Chandler (2015) Improved State-space Models for Inference about Spatial and Temporal Variation in Abundance from Count Data. *Ecology* 96: 1713-1723.

Royle, J. A. (2004). Generalized estimators of avian abundance from count survey data. *Animal Biodiversity and Conservation* 27(1), 375-386.

See Also

multinomPois, gmultmix, unmarkedFrameMMO

Examples

```
#Generate some data
set.seed(123)
lambda=4; gamma=0.5; omega=0.8; p=0.5
M <- 100; T <- 5
y <- array(NA, c(M, 3, T))
N <- matrix(NA, M, T)
S <- G <- matrix(NA, M, T-1)

for(i in 1:M) {
  N[i,1] <- rpois(1, lambda)
  y[i,1,1] <- rbinom(1, N[i,1], p)    # Observe some
  Nleft1 <- N[i,1] - y[i,1,1]         # Remove them
  y[i,2,1] <- rbinom(1, Nleft1, p)   # ...
  Nleft2 <- Nleft1 - y[i,2,1]
  y[i,3,1] <- rbinom(1, Nleft2, p)

  for(t in 1:(T-1)) {
    S[i,t] <- rbinom(1, N[i,t], omega)
    G[i,t] <- rpois(1, gamma)
    N[i,t+1] <- S[i,t] + G[i,t]
    y[i,1,t+1] <- rbinom(1, N[i,t+1], p)    # Observe some
    Nleft1 <- N[i,t+1] - y[i,1,t+1]         # Remove them
```

```
    y[i,2,t+1] <- rbinom(1, Nleft1, p)   # ...
    Nleft2 <- Nleft1 - y[i,2,t+1]
    y[i,3,t+1] <- rbinom(1, Nleft2, p)
  }
}
y=matrix(y, M)

#Create some random covariate data
sc <- data.frame(x1=rnorm(100))

## Not run:
#Create unmarked frame
umf <- unmarkedFrameMMO(y=y, numPrimary=5, siteCovs=sc, type="removal")

#Fit model
(fit <- multmixOpen(~x1, ~1, ~1, ~1, K=30, data=umf))

#Compare to truth
cf <- coef(fit)
data.frame(model=c(exp(cf[1]), cf[2], exp(cf[3]), plogis(cf[4]), plogis(cf[5])),
           truth=c(lambda, 0, gamma, omega, p))

#Predict
head(predict(fit, type='lambda'))

#Check fit with parametric bootstrap
pb <- parboot(fit, nsims=15)
plot(pb)

# Empirical Bayes estimates of abundance for each site / year
re <- ranef(fit)
plot(re, layout=c(10,5), xlim=c(-1, 10))

## End(Not run)
```

---

nmixTTD *Fit N-mixture Time-to-detection Models*

Description

Fit N-mixture models with time-to-detection data.

Usage

```
nmixTTD(stateformula= ~1, detformula = ~1, data, K=100,
    mixture = c("P","NB"), ttdDist = c("exp", "weibull"), starts, method="BFGS",
     se=TRUE, engine = c("C", "R"), threads = 1, ...)
```

Arguments

stateformula	Right-hand sided formula for the abundance at each site.
detformula	Right-hand sided formula for mean time-to-detection.
data	unmarkedFrameOccuTTD object that supplies the data (see unmarkedFrameOccuTTD). Note that only single-season models are supported by nmixTTD.
K	The upper summation index used to numerically integrate out the latent abundance. This should be set high enough so that it does not affect the parameter estimates. Computation time will increase with K.
mixture	String specifying mixture distribution: "P" for Poisson or "NB" for negative binomial.
ttdDist	Distribution to use for time-to-detection; either "exp" for the exponential, or "weibull" for the Weibull, which adds an additional shape parameter k.
starts	optionally, initial values for parameters in the optimization.
method	Optimization method used by optim.
se	logical specifying whether or not to compute standard errors.
engine	Either "C" or "R" to use fast C++ code or native R code during the optimization.
threads	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If threads=1 (the default), OpenMP is disabled.
...	Additional arguments to optim, such as lower and upper bounds

Details

This model extends time-to-detection (TTD) occupancy models to estimate site abundance using data from single or repeated visits. Latent abundance can be modeled as Poisson (mixture="P") or negative binomial (mixture="NB"). Time-to-detection can be modeled as an exponential (ttdDist="exp") or Weibull (ttdDist="weibull") random variable with rate parameter λ and, for the Weibull, an additional shape parameter k. Note that occuTTD puts covariates on λ and not $1/\lambda$, i.e., the expected time between events.

Assuming that there are N independent individuals at a site, and all individuals have the same individual detection rate, the expected detection rate across all individuals λ is equal to the the individual-level detection rate r multipled by the number of individuals present N.

In the case where there are no detections before the maximum sample time at a site (surveyLength) is reached, we are not sure if the site has $N = 0$ or if we just didn't wait long enough for a detection. We therefore must censor (C the exponential or Weibull distribution at the maximum survey length, $Tmax$. Thus, assuming true abundance at site i is N_i, and an exponential distribution for the TTD y_i (parameterized with the rate), then:

$$y_i \sim Exponential(r_i * N_i)C(Tmax)$$

Note that when $N_i = 0$, the exponential rate $lambda = 0$ and the scale is therefore $1/0 = Inf$, and thus the value will be censored at $Tmax$.

Because in unmarked values of NA are typically used to indicate missing values that were a result of the sampling structure (e.g., lost data), we indicate a censored y_i in nmixTTD instead by setting $y_i = Tmax_i$ in the y matrix provided to unmarkedFrameOccuTTD. You can provide either a single value of $Tmax$ to the surveyLength argument of unmarkedFrameOccuTTD, or provide a matrix, potentially with a unique value of $Tmax$ for each value of y. Note that in the latter case the value of y that will be interpreted by nmixTTD as a censored observation (i.e., $Tmax$) will differ between observations!

Value

unmarkedFitNmixTTD object describing model fit.

Author(s)

Ken Kellner <contact@kenkellner.com>

References

Strebel, N., Fiss, C., Kellner, K. F., Larkin, J. L., Kery, M., & Cohen, J (2021). Estimating abundance based on time-to-detection data. Methods in Ecology and Evolution 12: 909-920.

See Also

unmarked, unmarkedFrameOccuTTD

Examples

```
## Not run:

# Simulate data
M = 1000 # Number of sites
nrep <- 3 # Number of visits per site
Tmax = 5 # Max duration of a visit
alpha1 = -1 # Covariate on rate
beta1 = 1 # Covariate on density
mu.lambda = 1 # Rate at alpha1 = 0
mu.dens = 1 # Density at beta1 = 0

covDet <- matrix(rnorm(M*nrep),nrow = M,ncol = nrep) #Detection covariate
covDens <- rnorm(M) #Abundance/density covariate
dens <- exp(log(mu.dens) + beta1 * covDens)
sum(N <- rpois(M, dens)) # Realized density per site
lambda <- exp(log(mu.lambda) + alpha1 * covDet) # per-individual detection rate
ttd <- NULL
for(i in 1:nrep) {
  ttd <- cbind(ttd,rexp(M, N*lambda[,i]))  # Simulate time to first detection per visit
}
ttd[N == 0,] <- 5 # Not observed where N = 0; ttd set to Tmax
ttd[ttd >= Tmax] <- 5 # Crop at Tmax

#Build unmarked frame
```

```
umf <- unmarkedFrameOccuTTD(y = ttd, surveyLength=5,
                            siteCovs = data.frame(covDens=covDens),
                            obsCovs = data.frame(covDet=as.vector(t(covDet))))

#Fit model
fit <- nmixTTD(~covDens, ~covDet, data=umf, K=max(N)+10)

#Compare to truth
cbind(coef(fit), c(log(mu.dens), beta1, log(mu.lambda), alpha1))

#Predict abundance/density values
head(predict(fit, type='state'))


## End(Not run)
```

---

nonparboot-methods *Nonparametric bootstrapping in unmarked*

Description

Call nonparboot on an unmarkedFit to obtain non-parametric bootstrap samples. These can then be used by vcov in order to get bootstrap estimates of standard errors.

Details

Calling nonparboot on an unmarkedFit returns the original unmarkedFit, with the bootstrap samples added on. Then subsequent calls to vcov with the argument method="nonparboot" will use these bootstrap samples. Additionally, standard errors of derived estimates from either linearComb or backTransform can be instructed to use bootstrap samples by providing the argument method = "nonparboot".

For occu and occuRN both sites and occassions are re-sampled. For all other fitting functions, only sites are re-sampled.

Methods

signature(object = "unmarkedFit") Obtain nonparametric bootstrap samples for a general unmarkedFit.

signature(object = "unmarkedFitColExt") Obtain nonparametric bootstrap samples for colext fits.

signature(object = "unmarkedFitDS") Obtain nonparametric bootstrap samples for a distsamp fits.

signature(object = "unmarkedFitMPois") Obtain nonparametric bootstrap samples for a distsamp fits.

signature(object = "unmarkedFitOccu") Obtain nonparametric bootstrap samples for a occu fits.

`signature(object = "unmarkedFitOccuPEN")` Obtain nonparametric bootstrap samples for an occuPEN fit.

`signature(object = "unmarkedFitOccuPEN_CV")` Obtain nonparametric bootstrap samples for occuPEN_CV fit.

`signature(object = "unmarkedFitOccuRN")` Obtain nonparametric bootstrap samples for a occuRN fits.

`signature(object = "unmarkedFitPCount")` Obtain nonparametric bootstrap samples for a pcount fits.

Examples

```
data(ovendata)
ovenFrame <- unmarkedFrameMPois(ovendata.list$data,
siteCovs=as.data.frame(scale(ovendata.list$covariates[,-1])), type = "removal")
(fm <- multinomPois(~ 1 ~ ufc + trba, ovenFrame))
fm <- nonparboot(fm, B = 20) # should use larger B in real life.
vcov(fm, method = "hessian")
vcov(fm, method = "nonparboot")
avg.abundance <- backTransform(linearComb(fm, type = "state", coefficients = c(1, 0, 0)))

## Bootstrap sample information propagates through to derived quantities.
vcov(avg.abundance, method = "hessian")
vcov(avg.abundance, method = "nonparboot")
SE(avg.abundance, method = "nonparboot")
```

---

occu *Fit the MacKenzie et al. (2002) Occupancy Model*

Description

This function fits the single season occupancy model of MacKenzie et al (2002).

Usage

```
occu(formula, data, knownOcc=numeric(0), linkPsi=c("logit", "cloglog"),
            starts, method="BFGS", se=TRUE, engine=c("C", "R", "TMB"),
            threads = 1, ...)
```

Arguments

`formula`	Double right-hand side formula describing covariates of detection and occupancy in that order.
`data`	An `unmarkedFrameOccu` object
`knownOcc`	Vector of sites that are known to be occupied. These should be supplied as row numbers of the y matrix, eg, c(3,8) if sites 3 and 8 were known to be occupied a priori.

linkPsi	Link function for the occupancy model. Options are "logit" for the standard occupancy model or "cloglog" for the complimentary log-log link, which relates occupancy to site-level abundance. See details.
starts	Vector of parameter starting values.
method	Optimization method used by optim.
se	Logical specifying whether or not to compute standard errors.
engine	Code to use for optimization. Either "C" for fast C++ code, "R" for native R code, or "TMB" for Template Model Builder. "TMB" is used automatically if your formula contains random effects.
threads	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If threads=1 (the default), OpenMP is disabled.
...	Additional arguments to optim, such as lower and upper bounds

Details

See unmarkedFrame and unmarkedFrameOccu for a description of how to supply data to the data argument.

occu fits the standard occupancy model based on zero-inflated binomial models (MacKenzie et al. 2006, Royle and Dorazio 2008). The occupancy state process (z_i) of site i is modeled as

$$z_i \sim Bernoulli(\psi_i)$$

The observation process is modeled as

$$y_{ij}|z_i \sim Bernoulli(z_i p_{ij})$$

By default, covariates of ψ_i and p_{ij} are modeled using the logit link according to the formula argument. The formula is a double right-hand sided formula like ~ detform ~ occform where detform is a formula for the detection process and occform is a formula for the partially observed occupancy state. See formula for details on constructing model formulae in R.

When linkPsi = "cloglog", the complimentary log-log link function is used for psi instead of the logit link. The cloglog link relates occupancy probability to the intensity parameter of an underlying Poisson process (Kery and Royle 2016). Thus, if abundance at a site is can be modeled as N_i $Poisson(\lambda_i)$, where $log(\lambda_i) = \alpha + \beta * x$, then presence/absence data at the site can be modeled as Z_i $Binomial(\psi_i)$ where $cloglog(\psi_i) = \alpha + \beta * x$.

Value

unmarkedFitOccu object describing the model fit.

Author(s)

Ian Fiske

References

Kery, Marc, and J. Andrew Royle. 2016. *Applied Hierarchical Modeling in Ecology*, Volume 1. Academic Press.

MacKenzie, D. I., J. D. Nichols, G. B. Lachman, S. Droege, J. Andrew Royle, and C. A. Langtimm. 2002. Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One. Ecology 83: 2248-2255.

MacKenzie, D. I. et al. 2006. *Occupancy Estimation and Modeling*. Amsterdam: Academic Press.

Royle, J. A. and R. Dorazio. 2008. *Hierarchical Modeling and Inference in Ecology*. Academic Press.

See Also

unmarked, unmarkedFrameOccu, modSel, parboot

Examples

```
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)
plot(pferUMF, panels=4)
# add some fake covariates for illustration
siteCovs(pferUMF) <- data.frame(sitevar1 = rnorm(numSites(pferUMF)))

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

(fm <- occu(~ obsvar1 ~ 1, pferUMF))

confint(fm, type='det', method = 'normal')
confint(fm, type='det', method = 'profile')

# estimate detection effect at obsvars=0.5
(lc <- linearComb(fm['det'],c(1,0.5)))

# transform this to probability (0 to 1) scale and get confidence limits
(btlc <- backTransform(lc))
confint(btlc, level = 0.9)

# Empirical Bayes estimates of proportion of sites occupied
re <- ranef(fm)
sum(bup(re, stat="mode"))
```

---

occuCOP *Fit the occupancy model using count dta*

Description

This function fits a single season occupancy model using count data.

Usage

```
occuCOP(data,
        psiformula = ~1, lambdaformula = ~1,
        psistarts, lambdastarts, starts,
        method = "BFGS", se = TRUE,
        engine = c("C", "R"), na.rm = TRUE,
        return.negloglik = NULL, L1 = FALSE, ...)
```

Arguments

data	An unmarkedFrameOccuCOP object created with the unmarkedFrameOccuCOP function.
psiformula	Formula describing the occupancy covariates.
lambdaformula	Formula describing the detection covariates.
psistarts	Vector of starting values for likelihood maximisation with optim for occupancy probability ψ. These values must be logit-transformed (with qlogis) (see details). By default, optimisation will start at 0, corresponding to an occupancy probability of 0.5 (plogis(0) is 0.5).
lambdastarts	Vector of starting values for likelihood maximisation with optim for detection rate λ. These values must be log-transformed (with log) (see details). By default, optimisation will start at 0, corresponding to detection rate of 1 (exp(0) is 1).
starts	Vector of starting values for likelihood maximisation with optim. If psistarts and lambdastarts are provided, starts = c(psistarts, lambdastarts).
method	Optimisation method used by optim.
se	Logical specifying whether to compute (se=TRUE) standard errors or not (se=FALSE).
engine	Code to use for optimisation. Either "C" for fast C++ code, or "R" for native R code.
na.rm	Logical specifying whether to fit the model (na.rm=TRUE) or not (na.rm=FALSE) if there are NAs in the unmarkedFrameOccuCOP object.
return.negloglik	A list of vectors of parameters (c(psiparams, lambdaparams)). If specified, the function will not maximise likelihood but return the negative log-likelihood for the those parameters in the nll column of a dataframe. See an example below.
L1	Logical specifying whether the length of observations (L) are purposefully set to 1 (L1=TRUE) or not (L1=FALSE).
...	Additional arguments to pass to optim, such as lower and upper bounds or a list of control parameters.

Details

See unmarkedFrameOccuCOP for a description of how to supply data to the data argument. See unmarkedFrame for a more general documentation of unmarkedFrame objects for the different models implemented in **unmarked**.

The COP occupancy model:

`occuCOP` fits a single season occupancy model using count data, as described in Pautrel et al. (2023).

The **occupancy sub-model** is:

$$z_i \sim \text{Bernoulli}(\psi_i)$$

- With z_i the occupany state of site i. $z_i = 1$ if site i is occupied by the species, *i.e.* if the species is present in site i. $z_i = 0$ if site i is not occupied.
- With ψ_i the occupancy probability of site i.

The **observation sub-model** is:

$$N_{ij}|z_i = 1 \sim \text{Poisson}(\lambda_{ij} L_{ij}) N_{ij}|z_i = 0 \sim 0$$

- With N_{ij} the count of detection events in site i during observation j.
- With λ_{ij} the detection rate in site i during observation j (*for example, 1 detection per day.*).
- With L_{ij} the length of observation j in site i (*for example, 7 days.*).

What we call "observation" (j) here can be a sampling occasion, a transect, a discretised session. Consequently, the unit of λ_{ij} and L_{ij} can be either a time-unit (day, hour, ...) or a space-unit (kilometer, meter, ...).

The transformation of parameters ψ and λ: In order to perform unconstrained optimisation, parameters are transformed.

The occupancy probability (ψ) is transformed with the logit function (`psi_transformed = qlogis(psi)`). It can be back-transformed with the "inverse logit" function (`psi = plogis(psi_transformed)`).

The detection rate (λ) is transformed with the log function (`lambda_transformed = log(lambda)`). It can be back-transformed with the exponential function (`lambda = exp(lambda_transformed)`).

Value

`unmarkedFitOccuCOP` object describing the model fit. See the `unmarkedFit` classes.

Author(s)

Léa Pautrel

References

Pautrel, L., Moulherat, S., Gimenez, O. & Etienne, M.-P. Submitted. *Analysing biodiversity observation data collected in continuous time: Should we use discrete or continuous-time occupancy models?* Preprint at doi:10.1101/2023.11.17.567350.

See Also

`unmarked`, `unmarkedFrameOccuCOP`, `unmarkedFit-class`

Examples

```
set.seed(123)
options(max.print = 50)

# We simulate data in 100 sites with 3 observations of 7 days per site.
nSites <- 100
nObs <- 3

# For an occupancy covariate, we associate each site to a land-use category.
landuse <- sample(factor(c("Forest", "Grassland", "City"), ordered = TRUE),
                  size = nSites, replace = TRUE)
simul_psi <- ifelse(landuse == "Forest", 0.8,
                    ifelse(landuse == "Grassland", 0.4, 0.1))
z <- rbinom(n = nSites, size = 1, prob = simul_psi)

# For a detection covariate, we create a fake wind variable.
wind <- matrix(rexp(n = nSites * nObs), nrow = nSites, ncol = nObs)
simul_lambda <- wind / 5
L = matrix(7, nrow = nSites, ncol = nObs)

# We now simulate count detection data
y <- matrix(rpois(n = nSites * nObs, lambda = simul_lambda * L),
            nrow = nSites, ncol = nObs) * z

# We create our unmarkedFrameOccuCOP object
umf <- unmarkedFrameOccuCOP(
  y = y,
  L = L,
  siteCovs = data.frame("landuse" = landuse),
  obsCovs = list("wind" = wind)
)
print(umf)

# We fit our model without covariates
fitNull <- occuCOP(data = umf)
print(fitNull)

# We fit our model with covariates
fitCov <- occuCOP(data = umf, psiformula = ~ landuse, lambdaformula = ~ wind)
print(fitCov)

# We back-transform the parameter's estimates
## Back-transformed occupancy probability with no covariates
backTransform(fitNull, "psi")

## Back-transformed occupancy probability depending on habitat use
predict(fitCov,
        "psi",
        newdata = data.frame("landuse" = c("Forest", "Grassland", "City")),
        appendData = TRUE)

## Back-transformed detection rate with no covariates
```

```
backTransform(fitNull, "lambda")

## Back-transformed detection rate depending on wind
predict(fitCov,
        "lambda",
        appendData = TRUE)

## This is not easily readable. We can show the results in a clearer way, by:
##  - adding the site and observation
##  - printing only the wind covariate used to get the predicted lambda
cbind(
  data.frame(
    "site" = rep(1:nSites, each = nObs),
    "observation" = rep(1:nObs, times = nSites),
    "wind" = getData(fitCov)@obsCovs
  ),
  predict(fitCov, "lambda", appendData = FALSE)
)

# We can choose the initial parameters when fitting our model.
# For psi, intituively, the initial value can be the proportion of sites
#          in which we have observations.
(psi_init <- mean(rowSums(y) > 0))

# For lambda, the initial value can be the mean count of detection events
#             in sites in which there was at least one observation.
(lambda_init <- mean(y[rowSums(y) > 0, ]))

# We have to transform them.
occuCOP(
  data = umf,
  psiformula = ~ 1,
  lambdaformula = ~ 1,
  psistarts = qlogis(psi_init),
  lambdastarts = log(lambda_init)
)

# If we have covariates, we need to have the right length for the start vectors.
# psi ~ landuse --> 3 param to estimate: Intercept, landuseForest, landuseGrassland
# lambda ~ wind --> 2 param to estimate: Intercept, wind
occuCOP(
  data = umf,
  psiformula = ~ landuse,
  lambdaformula = ~ wind,
  psistarts = rep(qlogis(psi_init), 3),
  lambdastarts = rep(log(lambda_init), 2)
)

# And with covariates, we could have chosen better initial values, such as the
# proportion of sites in which we have observations per land-use category.
(psi_init_covs <- c(
  "City" = mean(rowSums(y[landuse == "City", ]) > 0),
  "Forest" = mean(rowSums(y[landuse == "Forest", ]) > 0),
```

```
  "Grassland" = mean(rowSums(y[landuse == "Grassland", ]) > 0)
))
occuCOP(
  data = umf,
  psiformula = ~ landuse,
  lambdaformula = ~ wind,
  psistarts = qlogis(psi_init_covs))

# We can fit our model with a different optimisation algorithm.
occuCOP(data = umf, method = "Nelder-Mead")

# We can run our model with a C++ or with a R likelihood function.
## They give the same result.
occuCOP(data = umf, engine = "C", psistarts = 0, lambdastarts = 0)
occuCOP(data = umf, engine = "R", psistarts = 0, lambdastarts = 0)

## The C++ (the default) is faster.
system.time(occuCOP(data = umf, engine = "C", psistarts = 0, lambdastarts = 0))
system.time(occuCOP(data = umf, engine = "R", psistarts = 0, lambdastarts = 0))

## However, if you want to understand how the likelihood is calculated,
## you can easily access the R likelihood function.
print(occuCOP(data = umf, engine = "R", psistarts = 0, lambdastarts = 0)@nllFun)

# Finally, if you do not want to fit your model but only get the likelihood,
# you can get the negative log-likelihood for a given set of parameters.
occuCOP(data = umf, return.negloglik = list(
  c("psi" = qlogis(0.25), "lambda" = log(2)),
  c("psi" = qlogis(0.5), "lambda" = log(1)),
  c("psi" = qlogis(0.75), "lambda" = log(0.5))
))
```

`occuFP` *Fit occupancy models when false positive detections occur (e.g., Royle and Link [2006] and Miller et al. [2011])*

Description

This function fits the single season occupancy model while allowing for false positive detections.

Usage

```
occuFP(detformula = ~ 1, FPformula = ~ 1, Bformula = ~ 1,
stateformula = ~ 1, data, starts, method="BFGS", se = TRUE, engine = "R", ...)
```

Arguments

`detformula`	formula describing covariates of detection.
`FPformula`	formula describing covariates of false positive detection probability.

Bformula	formula describing covariates of probability detections are certain.
stateformula	formula describing covariates of occupancy.
data	An unmarkedFrameOccuFP object
starts	Vector of parameter starting values.
method	Optimization method used by optim.
se	Logical specifying whether or not to compute standard errors.
engine	Currently only choice is R.
...	Additional arguments to optim, such as lower and upper bounds

Details

See unmarkedFrame and unmarkedFrameOccuFP for a description of how to supply data to the data argument.

occuFP fits an extension of the standard single-season occupancy model (MacKenzie et al. 2002), which allows false positive detections. The occupancy status of a site is the same way as with the occu function, where stateformula is used to specify factors that lead to differences in occupancy probabilities among sites.

The observation process differs in that both false negative and false positive errors are modeled for observations. The function allows data to be of 3 types. These types are specified using in unmarkedFrameOccuFP as type. Occassions are specified to belong to 1 of the 3 data types and all or a subset of the data types can be combined in the same model.

For type 1 data, the detection process is assumed to fit the assumptions of the standard MacKenzie model where false negative probabilities are estimated but false positive detections are assumed not to occur. If all of your data is of this type you should use occu to analyze data. The detection parameter p, which is modeled using the detformula is the only observation parameter for these data.

For type 2 data, both false negative and false positive detection probabilities are estimated. If all data is of this type the likelihood follows Royle and Link (2006). Both p (the true positive detection probability) and fp (the false positive detection probability described by fpformula) are estimated for occassions when this data type occurs

For type 3 data, observations are assumed to include both certain detections (false positives assumed not to occur) and uncertain detections that may include false positive detections. When only this data type occurs, the estimator is the same as the multiple detection state model described in Miller et al. (2011). Three observation parameters occur for this data type: p - true positive detection probability, fp - false positive detection probability, and b - the probability a true positive detection was designated as certain.

When both type 1 and type 2 data occur, the estimator is equivalent to the multiple detection method model described in Miller et al. (2011). The frog data example in the same paper uses an analysis where type 1 (dipnet surveys) and type 3 (call surveys) data were used.

Data in the y matrix of the unmarked frame should be all 0s and 1s for type 1 and type 2 data. For type 3 data, uncertain detections are given a value of 1 and certain detections a value of 2.

Value

unmarkedFitOccuFP object describing the model fit.

Author(s)

David Miller

References

MacKenzie, D. I., J. D. Nichols, G. B. Lachman, S. Droege, J. Andrew Royle, and C. A. Langtimm. 2002. Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One. Ecology 83: 2248-2255.

Miller, D.A., J.D. Nichols, B.T. McClintock, E.H.C. Grant, L.L. Bailey, and L.A. Weir. 2011. Improving occupancy estimation when two types of observational error occur: non-detection and species misidentification. Ecology 92:1422-1428.

Royle, J.A., and W.A. Link. 2006. Generalized site occupancy models allowing for false positive and false negative errors. Ecology 87:835-841.

See Also

unmarked, unmarkedFrameOccuFP, modSel, parboot

Examples

```
n = 100
o = 10
o1 = 5
y = matrix(0,n,o)
p = .7
r = .5
fp = 0.05
y[1:(n*.5),(o-o1+1):o] <- rbinom((n*o1*.5),1,p)
y[1:(n*.5),1:(o-o1)] <- rbinom((o-o1)*n*.5,1,r)
y[(n*.5+1):n,(o-o1+1):o] <- rbinom((n*o1*.5),1,fp)
type <- c((o-o1),o1,0)  ### vector with the number of each data type
site <- c(rep(1,n*.5*.8),rep(0,n*.5*.2),rep(1,n*.5*.2),rep(0,n*.8*.5))
occ <- matrix(c(rep(0,n*(o-o1)),rep(1,n*o1)),n,o)
site <- data.frame(habitat = site)
occ <- list(METH = occ)

umf1 <- unmarkedFrameOccuFP(y,site,occ, type = type)

m1 <- occuFP(detformula = ~ METH, FPformula = ~1,
             stateformula = ~ habitat, data = umf1)
predict(m1, type = 'fp')
coef(m1)
confint(m1, type = 'det')
```

occuMS *Fit Single-Season and Dynamic Multi-State Occupancy Models*

Description

This function fits single-season and dynamic multi-state occupancy models with both the multinomial and conditional binomial parameterizations.

Usage

```
occuMS(detformulas, psiformulas, phiformulas=NULL, data,
    parameterization=c("multinomial","condbinom"),
    starts, method="BFGS", se=TRUE, engine=c("C","R"), silent=FALSE, ...)
```

Arguments

detformulas	Character vector of formulas for detection probabilities. See details for a description of how to order these formulas.
psiformulas	Character vector of formulas for occupancy probabilities. See details for a description of how to order these formulas.
phiformulas	Character vector of formulas for state transition probabilities. Only used if you are fitting a dynamic model. See details for a description of how to order these formulas.
data	An unmarkedFrameOccuMS object
parameterization	Either "multinomial" for the multinomial parameterization (MacKenzie et al. 2009) which allows an arbitrary number of occupancy states, or "condbinom" for the conditional binomial parameterization (Nichols et al. 2007) which requires exactly 3 occupancy states. See details.
starts	Vector of parameter starting values.
method	Optimization method used by optim.
se	Logical specifying whether or not to compute standard errors.
engine	Either "C" to use fast C++ code or "R" to use native R code during the optimization.
silent	Boolean; if TRUE, suppress warnings.
...	Additional arguments to optim, such as lower and upper bounds

Details

Traditional occupancy models fit data with exactly two states: detection and non-detection (MacKenzie et al. 2002). The occuMS function fits models to occupancy data for which there are greater than 2 states (Nichols et al 2007, MacKenzie et al. 2009). For example, detections may be further divided into multiple biologically relevant categories, e.g. breeding vs. non-breeding, or some/many

individuals present. As with detection status, classification of these additional occupancy states is likely to be imperfect.

Multiple parameterizations for multi-state occupancy models have been proposed. The `occuMS` function fits two at present: the "conditional binomial" parameterization of Nichols et al. (2007), and the more general "multinomial" parameterization of MacKenzie et al. (2009). Both single-season and dynamic models are possible with `occuMS` (MacKenzie et al. 2009).

The conditional binomial parameterization (`parameterization = 'condbinom'`) models occupancy and the presence or absence of an additional biological state of interest given the species is present (typically breeding status). Thus, there should be exactly 3 occupancy states in the data: 0 (non-detection); 1 (detection, no evidence of breeding); or 2 (detection, evidence of breeding).

Two state parameters are estimated: ψ, the probability of occupancy, and R, the probability of successful reproduction given an occupied state (although this could be some other binary biological condition). Covariates (in `siteCovs`) can be supplied for either or both of these parameters with the `stateformulas` argument, which takes a character vector of R-style formulas with length = 2, with formulas in the order (ψ, R). For example, to fit a model where ψ varies with a landcover covariate and R is constant, `stateformulas = c('~landcover','~1')`.

There are three detection parameters associated with the conditional binomial parameterization: p_1, the probability of detecting the species given true state 1; p_2, the probability of detecting the species given true state 2; and δ, the probability of detecting state 2 (i.e., breeding), given that the species has been detected. See MacKenzie et al. (2009), pages 825-826 for more details. As with occupancy, covariates (in `obsCovs`) can be supplied for these detection probabilities with the `detformulas` argument, which takes a character vector of formulas with length = 3 in the order (p_1, p_2, δ). So, to fit a model where p_1 varies with temperature and the other two parameters are constant, `detformulas = c('~temp','~1','~1')`.

The multinomial parameterization (`parameterization = "multinomial"`) is more general, allowing an arbitrary number of occupancy states S. S - 1 occupancy probabilities ψ are estimated. Thus, if there are S = 4 occupancy states (0, 1, 2, 3), `occuMS` estimates ψ_1, ψ_2, and ψ_3 (the probability of state 0 can be obtained by subtracting the others from 1). Covariates can be supplied for each occupancy probability with a character vector with length $S - 1$, e.g. `stateformulas = c('~landcover','~1','~1')` where ψ_1 varies with landcover and ψ_2 and ψ_3 are constant.

The number of detection probabilities estimated quickly expands as S increases, equal to $S \times (S - 1)/2$. In the simplest case (when S = 3), there are 3 detection probabilities: p_{11}, the probability of detecting state 1 given true state 1; p_{12}, the probability of detecting state 1 given true state 2; and p_{22}, the probability of detecting state 2 given true state 2. Covariates can be supplied for any or all of these detection probabilities with the `detformulas` argument, which takes a character vector of formulas with length = 3 in the order (p_{11}, p_{12}, p_{22}). So, to fit a model where p_{11} varies with temperature and the other two detection probabilities are constant, `detformulas = c('~temp','~1','~1')`. If there were S = 4 occupancy states, there are 6 estimated detection probabilities and the order is (p_{11}, p_{12}, p_{13}, p_{22}, p_{23}, p_{33}), and so on. See MacKenzie et al. (2009) for a more detailed explanation.

Dynamic (multi-season) models can be fit as well for both parameterizations (MacKenzie et al. 2009). In a standard dynamic occupancy model, additional parameters for probabilities of colonization (i.e., state 0 -> 1) and extinction (1 -> 0) are estimated. In a multi-state context, we must estimate a transition probability matrix (ϕ) between all possible states. You can provide formulas for some of the probabilities in this matrix using the `phiformulas` argument. The approach differs depending on parameterization.

For the conditional binomial parameterization, phiformulas is a character vector of length 6. The first three elements are formulas for the probability a site is occupied at time t given that it was previously in states 0, 1, or 2 at time $t-1$ (phi0, phi1, phi2). Elements 4-6 are formulas for the probability of reproduction (or other biological state) given state 0, 1, or 2 at time $t-1$ (R0, R1, R2). See umf@phiOrder$cond_binom for a reminder of the correct order, where umf is your unmarkedFrameOccuMS.

For the multinomial parameterization, phiformulas can be used to provide formulas for some transitions between different occupancy states. You can't give formulas for the probabilities of remaining in the same state between seasons to keep the model identifiable. Thus, if there are 3 possible states (0, 1, 2), phiformulas should contain 6 formulas for the following transitions: p(0->1), p(0->2), p(1->0), p(1->2), p(2->0), p(2->1), in that order (and similar for more than 3 states). The remaining probabilities of staying in the same state between seasons can be obtained via subtraction. See umf@phiOrder$multinomial for the correct order matching the number of states in your dataset.

See unmarkedFrame and unmarkedFrameOccuMS for a description of how to supply data to the data argument.

Value

unmarkedFitOccuMS object describing the model fit.

Author(s)

Ken Kellner <contact@kenkellner.com>

References

MacKenzie, D. I., J. D. Nichols, G. B. Lachman, S. Droege, J. Andrew Royle, and C. A. Langtimm. 2002. Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One. Ecology 83: 2248-2255.

MacKenzie, D. I., Nichols, J. D., Seamans, M. E., and R. J. Gutierrez, 2009. Modeling species occurrence dynamics with multiple states and imperfect detection. Ecology 90: 823-835.

Nichols, J. D., Hines, J. E., Mackenzie, D. I., Seamans, M. E., and R. J. Gutierrez. 2007. Occupancy estimation and modeling with multiple states and state uncertainty. Ecology 88: 1395-1400.

See Also

unmarked, unmarkedFrameOccuMS

Examples

```
## Not run:

#Simulate data

#Parameters
N <- 500; J <- 5; S <- 3
site_covs <- matrix(rnorm(N*2),ncol=2)
```

```
obs_covs <- matrix(rnorm(N*J*2),ncol=2)
a1 <- -0.5; b1 <- 1; a2 <- -0.6; b2 <- -0.7

#################################
## Multinomial parameterization ##
#################################

p11 <- -0.4; p12 <- -1.09; p22 <- -0.84
truth <- c(a1,b1,a2,b2,p11,0,p12,p22)

#State process
lp <- matrix(NA,ncol=S,nrow=N)
for (n in 1:N){
  lp[n,2] <- exp(a1+b1*site_covs[n,1])
  lp[n,3] <- exp(a2+b2*site_covs[n,2])
  lp[n,1] <- 1
}
psi_mat <- lp/rowSums(lp)

z <- rep(NA,N)
for (n in 1:N){
  z[n] <- sample(0:2, 1, replace=T, prob=psi_mat[n,])
}

probs_raw <- matrix(c(1,0,0,1,exp(p11),0,1,exp(p12),exp(p22)),nrow=3,byrow=T)
probs_raw <- probs_raw/rowSums(probs_raw)

y <- matrix(0,nrow=N,ncol=J)
for (n in 1:N){

  probs <- switch(z[n]+1,
                  probs_raw[1,],
                  probs_raw[2,],
                  probs_raw[3,])
  if(z[n]>0){
    y[n,] <- sample(0:2, J, replace=T, probs)
  }
}

#Construct unmarkedFrame
umf <- unmarkedFrameOccuMS(y=y,siteCovs=as.data.frame(site_covs),
                           obsCovs=as.data.frame(obs_covs))

#Formulas

#3 states, so detformulas is a character vector of formulas of
#length 3 in following order:
#1) p[11]: prob of detecting state 1 given true state 1
#2) p[12]: prob of detecting state 1 given true state 2
#3) p[22]: prob of detecting state 2 given true state 2
detformulas <- c('~V1','~1','~1')
#If you had 4 states, it would be p[11],p[12],p[13],p[22],p[23],p[33] and so on
```

```
#3 states, so stateformulas is a character vector of length 2 in following order:
#1) psi[1]: probability of state 1
#2) psi[2]: probability of state 2
#You can get probability of state 0 (unoccupied) as 1 - psi[1] - psi[2]
stateformulas <- c('~V1','~V2')

#Fit model
fit <- occuMS(detformulas, stateformulas, data=umf,
              parameterization="multinomial")

#Look at results
fit
#Compare with truth
cbind(truth=truth,estimate=coef(fit))

#Generate predicted values
lapply(predict(fit,type='psi'),head)
lapply(predict(fit,type='det'),head)

#Fit a null model
detformulas <- rep('~1',3)
stateformulas <- rep('~1',2)
fit_null <- occuMS(detformulas, stateformulas, data=umf,
                   parameterization="multinomial")

#Compare fits
modSel(fitList(fit,fit_null))

##########################################
## Conditional binomial parameterization ##
##########################################

p11 <- 0.4; p12 <- 0.6; p22 <- 0.8
truth_cb <- c(a1,b1,a2,b2,qlogis(p11),0,qlogis(c(p12,p22)))

#Simulate data

#State process
psi_mat <- matrix(NA,ncol=S,nrow=N)
for (n in 1:N){
  psi_mat[n,2] <- plogis(a1+b1*site_covs[n,1])
  psi_mat[n,3] <- plogis(a2+b2*site_covs[n,2])
}
psi_bin <- matrix(NA,nrow=nrow(psi_mat),ncol=ncol(psi_mat))
psi_bin[,1] <- 1-psi_mat[,2]
psi_bin[,2] <- (1-psi_mat[,3])*psi_mat[,2]
psi_bin[,3] <- psi_mat[,2]*psi_mat[,3]
z <- rep(NA,N)
for (n in 1:N){
  z[n] <- sample(0:2, 1, replace=T, prob=psi_bin[n,])
}

#Detection process
```

```
y_cb <- matrix(0,nrow=N,ncol=J)
for (n in 1:N){
  #p11 = p1; p12 = p2; p22 = delta
  probs <- switch(z[n]+1,
                  c(1,0,0),
                  c(1-p11,p11,0),
                  c(1-p12,p12*(1-p22),p12*p22))
  if(z[n]>0){
    y_cb[n,] <- sample(0:2, J, replace=T, probs)
  }
}

#Build unmarked frame
umf2 <- unmarkedFrameOccuMS(y=y_cb,siteCovs=as.data.frame(site_covs),
                            obsCovs=as.data.frame(obs_covs))

#Formulas

#detformulas is a character vector of formulas of length 3 in following order:
#1) p[1]: prob of detecting species given true state 1
#2) p[2]: prob of detecting species given true state 2
#3) delta: prob of detecting state 2 (eg breeding) given species was detected
detformulas <- c('~V1','~1','~1')

#stateformulas is a character vector of length 2 in following order:
#1) psi: probability of occupancy
#2) R: probability state 2 (eg breeding) given occupancyc
stateformulas <- c('~V1','~V2')

#Fit model
fit_cb <- occuMS(detformulas, stateformulas, data=umf2,
                 parameterization='condbinom')

#Look at results
fit_cb
#Compare with truth
cbind(truth=truth_cb,estimate=coef(fit_cb))

#Generate predicted values
lapply(predict(fit_cb,type='psi'),head)
lapply(predict(fit_cb,type='det'),head)


##################################
## Dynamic (multi-season) model ##
##################################

#Simulate data-------------------------------------------------
N <- 500 #Number of sites
T <- 3 #Number of primary periods
J <- 5 #Number of secondary periods
S <- 3 #Number of occupancy states (0,1,2)
```

```
#Generate covariates
site_covs <- as.data.frame(matrix(rnorm(N*2),ncol=2))
yearly_site_covs <- as.data.frame(matrix(rnorm(N*T*2),ncol=2))
obs_covs <- as.data.frame(matrix(rnorm(N*J*T*2),ncol=2))

#True parameter values
b <- c(
  #Occupancy parameters
  a1=-0.5, b1=1, a2=-0.6, b2=-0.7,
  #Transition prob (phi) parameters
  phi01=0.7, phi01_cov=-0.5, phi02=-0.5, phi10=1.2,
  phi12=0.3, phi12_cov=1.1, phi20=-0.3, phi21=1.4, phi21_cov=0,
  #Detection prob parameters
  p11=-0.4, p11_cov=0, p12=-1.09, p22=-0.84
)

#Generate occupancy probs (multinomial parameterization)
lp <- matrix(1, ncol=S, nrow=N)
lp[,2] <- exp(b[1]+b[2]*site_covs[,1])
lp[,3] <- exp(b[3]+b[4]*site_covs[,2])
psi <- lp/rowSums(lp)

#True occupancy state matrix
z <- matrix(NA, nrow=N, ncol=T)

#Initial occupancy
for (n in 1:N){
  z[n,1] <- sample(0:(S-1), 1, prob=psi[n,])
}

#Raw phi probs
phi_raw <- matrix(NA, nrow=N*T, ncol=S^2-S)
phi_raw[,1] <- exp(b[5]+b[6]*yearly_site_covs[,1]) #p[0->1]
phi_raw[,2] <- exp(b[7]) #p[0->2]
phi_raw[,3] <- exp(b[8]) #p[1->0]
phi_raw[,4] <- exp(b[9]+b[10]*yearly_site_covs[,2]) #p[1->2]
phi_raw[,5] <- exp(b[11]) #p[2->0]
phi_raw[,6] <- exp(b[12]+b[13]*yearly_site_covs[,1])

#Generate states in times 2..T
px <- 1
for (n in 1:N){
  for (t in 2:T){
    phi_mat <- matrix(c(1, phi_raw[px,1], phi_raw[px,2],  # phi|z=0
                        phi_raw[px,3], 1, phi_raw[px,4],  # phi|z=1
                        phi_raw[px,5], phi_raw[px,6], 1), # phi|z=2
                      nrow=S, byrow=T)
    phi_mat <- phi_mat/rowSums(phi_mat)
    z[n, t] <- sample(0:(S-1), 1, prob=phi_mat[z[n,(t-1)]+1,])
    px <- px + 1
    if(t==T) px <- px + 1 #skip last datapoint for each site
  }
}
```

```
#Raw p probs
p_mat <- matrix(c(1, 0, 0, #p|z=0
                  1, exp(b[14]), 0, #p|z=1
                  1, exp(b[16]), exp(b[17])), #p|z=2
                nrow=S, byrow=T)
p_mat <- p_mat/rowSums(p_mat)

#Simulate observation data
y <- matrix(0, nrow=N, ncol=J*T)
for (n in 1:N){
  yx <- 1
  for (t in 1:T){
    if(z[n,t]==0){
      yx <- yx + J
      next
    }
    for (j in 1:J){
      y[n, yx] <- sample(0:(S-1), 1, prob=p_mat[z[n,t]+1,])
      yx <- yx+1
    }
  }
}
#-----------------------------------------------------------------

#Model fitting

#Build UMF
umf <- unmarkedFrameOccuMS(y=y, siteCovs=site_covs,
                           obsCovs=obs_covs,
                           yearlySiteCovs=yearly_site_covs,
                           numPrimary=3)
summary(umf)

#Formulas
#Initial occupancy
psiformulas <- c('~V1','~V2') #on psi[1] and psi[2]

#Transition probs
#Guide to order:
umf@phiOrder$multinomial
phiformulas <- c('~V1','~1','~1','~V2','~1','~V1')

#Detection probability
detformulas <- c('~V1','~1','~1') #on p[1|1], p[1|2], p[2|2]

#Fit model
(fit <- occuMS(detformulas=detformulas, psiformulas=psiformulas,
               phiformulas=phiformulas, data=umf))

#Compare with truth
compare <- cbind(b,coef(fit),
                 coef(fit)-1.96*SE(fit),coef(fit)+1.96*SE(fit))
```

```
colnames(compare) <- c('truth','estimate','lower','upper')
round(compare,3)

#Estimated phi matrix for site 1
phi_est <- predict(fit, 'phi', se.fit=F)
phi_est <- sapply(phi_est, function(x) x$Predicted[1])
phi_est_mat <- matrix(NA, nrow=S, ncol=S)
phi_est_mat[c(4,7,2,8,3,6)] <- phi_est
diag(phi_est_mat) <- 1 - rowSums(phi_est_mat,na.rm=T)

#Actual phi matrix for site 1
phi_act_mat <- diag(S)
phi_act_mat[c(4,7,2,8,3,6)] <- phi_raw[1,]
phi_act_mat <- phi_act_mat/rowSums(phi_act_mat)

#Compare
cat('Estimated phi\n')
phi_est_mat
cat('Actual phi\n')
phi_act_mat

#Rough check of model fit
fit_sim <- simulate(fit, nsim=20)
hist(sapply(fit_sim,mean),col='gray')
abline(v=mean(umf@y),col='red',lwd=2)
#line should fall near middle of histogram


## End(Not run)
```

`occuMulti` *Fit the Rota et al. (2016) Multi-species Occupancy Model*

Description

This function fits the multispecies occupancy model of Rota et al (2016).

Usage

```
occuMulti(detformulas, stateformulas, data, maxOrder, penalty=0, boot=30,
       starts, method="BFGS", se=TRUE, engine=c("C","R"), silent=FALSE, ...)
```

Arguments

`detformulas`	Character vector of formulas for the detection models, one per species.
`stateformulas`	Character vector of formulas for the natural parameters. To fix a natural parameter at 0, specify the corresponding formula as `"0"` or `"~0"`.
`data`	An `unmarkedFrameOccuMulti` object

maxOrder	Optional; specify maximum interaction order. Defaults to number of species (all possible interactions). Reducing this value may speed up optimization if you aren't interested in higher-order interactions.
penalty	Penalty term for likelihood. The total penalty is calculated as `penalty * 0.5 * sum(paramvals^2)`. Defaults to 0 (no penalty).
boot	Number of bootstrap samples to use to generate the variance-covariance matrix when `penalty > 0`.
starts	Vector of parameter starting values.
method	Optimization method used by optim.
se	Logical specifying whether or not to compute standard errors.
engine	Either "C" to use fast C++ code or "R" to use native R code during the optimization.
silent	Boolean; if TRUE, suppress warnings.
...	Additional arguments to optim, such as lower and upper bounds

Details

See unmarkedFrame and unmarkedFrameOccuMulti for a description of how to supply data to the data argument.

occuMulti fits the multispecies occupancy model from Rota et al. (2016), for two or more interacting species. The model generalizes the standard single-species occupancy model from MacKenzie et al. (2002). The latent occupancy state at site i for a set of s potentially interacting species is a vector $\mathbf{Z}_i$ of length s containing a sequence of the values 0 or 1. For example, when $s = 2$, the possible states are [11], [10], [01], or [00], corresponding to both species present, only species 1 or species 2 present, or both species absent, respectively. The latent state modeled as a multivariate Bernoulli random variable:

$$\mathbf{Z}_i \sim \text{MVB}(\boldsymbol{\psi}_i)$$

where $\boldsymbol{\psi}_i$ is a vector of length 2^s containing the probability of each possible combination of 0s and 1s, such that $\sum \boldsymbol{\psi}_i = 1$.

For $s = 2$, the corresponding natural parameters f are

$$f_1 = \log\left(\frac{\psi_{10}}{\psi_{00}}\right)$$

$$f_2 = \log\left(\frac{\psi_{01}}{\psi_{00}}\right)$$

$$f_{12} = \log\left(\frac{\psi_{11}\psi_{00}}{\psi_{10}\psi_{01}}\right)$$

The natural parameters can then be modeled as linear functions of covariates. Covariates for each f must be specified with the stateformulas argument, which takes a character vector of individual formulas of length equal to the number of natural parameters (which in turn depends on the number of species in the model).

The observation process is similar to the standard single-species occupancy model, except that the observations $\mathbf{y}_{ij}$ at site i on occasion j are vectors of length s and there are independent values of detection probability p for each species s:

$$\mathbf{y}_{ij}|\mathbf{Z}_i \sim \text{MVB}(\mathbf{Z}_i p_{sij})$$

Independent detection models (potentially containing different covariates) must be provided for each species with the `detformulas` argument, which takes a character vector of individual formulas with length equal to the number of species s.

If you are having problems with separation or boundary estimates (indicated by very large parameter estimates and SEs), use of penalized likelihood may help: see Clipp et al. (2021). `occuMulti` supports use of the Bayes-inspired penalty of Hutchinson et al. (2015). You can set the penalty value manually using the `penalty` argument, or identify the optimal penalty using K-fold cross validation with the `optimizePenalty` function. See example below.

Value

unmarkedFitOccuMulti object describing the model fit.

Author(s)

Ken Kellner <contact@kenkellner.com>

References

Clipp, H. L., Evans, A., Kessinger, B. E., Kellner, K. F., and C. T. Rota. 2021. A penalized likelihood for multi-species occupancy models improves predictions of species interactions. Ecology.

Hutchinson, R. A., J. V. Valente, S. C. Emerson, M. G. Betts, and T. G. Dietterich. 2015. Penalized Likelihood Methods Improve Parameter Estimates in Occupancy Models. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.12368

MacKenzie, D. I., J. D. Nichols, G. B. Lachman, S. Droege, J. Andrew Royle, and C. A. Langtimm. 2002. Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One. Ecology 83: 2248-2255.

Rota, C.T., et al. 2016. A multi-species occupancy model for two or more interacting species. Methods in Ecology and Evolution 7: 1164-1173.

See Also

`unmarked`, `unmarkedFrameOccuMulti`

Examples

```
## Not run:
#Simulate 3 species data
N <- 1000
nspecies <- 3
J <- 5
```

```
occ_covs <- as.data.frame(matrix(rnorm(N * 10),ncol=10))
names(occ_covs) <- paste('occ_cov',1:10,sep='')

det_covs <- list()
for (i in 1:nspecies){
  det_covs[[i]] <- matrix(rnorm(N*J),nrow=N)
}
names(det_covs) <- paste('det_cov',1:nspecies,sep='')

#True vals
beta <- c(0.5,0.2,0.4,0.5,-0.1,-0.3,0.2,0.1,-1,0.1)
f1 <- beta[1] + beta[2]*occ_covs$occ_cov1
f2 <- beta[3] + beta[4]*occ_covs$occ_cov2
f3 <- beta[5] + beta[6]*occ_covs$occ_cov3
f4 <- beta[7]
f5 <- beta[8]
f6 <- beta[9]
f7 <- beta[10]
f <- cbind(f1,f2,f3,f4,f5,f6,f7)
z <- expand.grid(rep(list(1:0),nspecies))[,nspecies:1]
colnames(z) <- paste('sp',1:nspecies,sep='')
dm <- model.matrix(as.formula(paste0("~.^",nspecies,"-1")),z)

psi <- exp(f %*% t(dm))
psi <- psi/rowSums(psi)

#True state
ztruth <- matrix(NA,nrow=N,ncol=nspecies)
for (i in 1:N){
  ztruth[i,] <- as.matrix(z[sample(8,1,prob=psi[i,]),])
}

p_true <- c(0.6,0.7,0.5)

# fake y data
y <- list()

for (i in 1:nspecies){
  y[[i]] <- matrix(NA,N,J)
  for (j in 1:N){
    for (k in 1:J){
      y[[i]][j,k] <- rbinom(1,1,ztruth[j,i]*p_true[i])
    }
  }
}
names(y) <- c('coyote','tiger','bear')

#Create the unmarked data object
data = unmarkedFrameOccuMulti(y=y,siteCovs=occ_covs,obsCovs=det_covs)

#Summary of data object
summary(data)
plot(data)
```

```
# Look at f parameter design matrix
data@fDesign

# Formulas for state and detection processes

# Length should match number/order of columns in fDesign
occFormulas <- c('~occ_cov1','~occ_cov2','~occ_cov3','~1','~1','~1','~1')

#Length should match number/order of species in data@ylist
detFormulas <- c('~1','~1','~1')

fit <- occuMulti(detFormulas,occFormulas,data)

#Look at output
fit

plot(fit)

#Compare with known values
cbind(c(beta,log(p_true/(1-p_true))),fit@opt$par)

#predict method
lapply(predict(fit,'state'),head)
lapply(predict(fit,'det'),head)

#marginal occupancy
head(predict(fit,'state',species=2))
head(predict(fit,'state',species='bear'))
head(predict(fit,'det',species='coyote'))

#probability of co-occurrence of two or more species
head(predict(fit, 'state', species=c('coyote','tiger')))

#conditional occupancy
head(predict(fit,'state',species=2,cond=3)) #tiger | bear present
head(predict(fit,'state',species='tiger',cond='bear')) #tiger | bear present
head(predict(fit,'state',species='tiger',cond='-bear')) #bear absent
head(predict(fit,'state',species='tiger',cond=c('coyote','-bear')))

#residuals (by species)
lapply(residuals(fit),head)

#ranef (by species)
ranef(fit, species='coyote')

#parametric bootstrap
bt <- parboot(fit,nsim=30)

#update model
occFormulas <- c('~occ_cov1','~occ_cov2','~occ_cov2+occ_cov3','~1','~1','~1','~1')
fit2 <- update(fit,stateformulas=occFormulas)
```

```
#List of fitted models
fl <- fitList(fit,fit2)
coef(fl)

#Model selection
modSel(fl)

#Fit model while forcing some natural parameters to be 0
#For example: fit model with no species interactions
occFormulas <- c('~occ_cov1','~occ_cov2','~occ_cov2+occ_cov3','0','0','0','0')
fit3 <- occuMulti(detFormulas,occFormulas,data)

#Alternatively, you can force all interaction parameters above a certain
#order to be zero with maxOrder. This will be faster.
occFormulas <- c('~occ_cov1','~occ_cov2','~occ_cov2+occ_cov3')
fit4 <- occuMulti(detFormulas,occFormulas,data,maxOrder=1)

#Add Bayes penalty term to likelihood. This is useful if your parameter
#estimates are very large, eg because of separation.
fit5 <- occuMulti(detFormulas, occFormulas, data, penalty=1)

#Find optimal penalty term value from a range of possible values using
#K-fold cross validation, and re-fit the model
fit_opt <- optimizePenalty(fit5, penalties=c(0,1,2))

## End(Not run)
```

---

occuPEN *Fit the MacKenzie et al. (2002) Occupancy Model with the penalized likelihood methods of Hutchinson et al. (2015)*

Description

This function fits the occupancy model of MacKenzie et al (2002) with the penalized methods of Hutchinson et al (2015).

Usage

```
occuPEN(formula, data, knownOcc=numeric(0), starts, method="BFGS",
    engine=c("C", "R"), lambda=0, pen.type = c("Bayes","Ridge","MPLE"), ...)
```

Arguments

formula	Double right-hand side formula describing covariates of detection and occupancy in that order.
data	An unmarkedFrameOccu object

knownOcc	Vector of sites that are known to be occupied. These should be supplied as row numbers of the y matrix, eg, c(3,8) if sites 3 and 8 were known to be occupied a priori.
starts	Vector of parameter starting values.
method	Optimization method used by optim.
engine	Either "C" or "R" to use fast C++ code or native R code during the optimization.
lambda	Penalty weight parameter.
pen.type	Which form of penalty to use.
...	Additional arguments to optim, such as lower and upper bounds

Details

See unmarkedFrame and unmarkedFrameOccu for a description of how to supply data to the data argument.

occuPEN fits the standard occupancy model based on zero-inflated binomial models (MacKenzie et al. 2006, Royle and Dorazio 2008) using the penalized likelihood methods described in Hutchinson et al. (2015). See occu for model details. occuPEN returns parameter estimates that maximize a penalized likelihood in which the penalty is specified by the pen.type argument. The penalty function is weighted by lambda.

The MPLE method includes an equation for computing lambda (Moreno & Lele, 2010). If the value supplied does not equal match the one computed with this equation, the supplied value is used anyway (with a warning).

Value

unmarkedFitOccuPEN object describing the model fit.

Author(s)

Rebecca A. Hutchinson

References

Hutchinson, R. A., J. V. Valente, S. C. Emerson, M. G. Betts, and T. G. Dietterich. 2015. Penalized Likelihood Methods Improve Parameter Estimates in Occupancy Models. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.12368

MacKenzie, D. I., J. D. Nichols, G. B. Lachman, S. Droege, J. Andrew Royle, and C. A. Langtimm. 2002. Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One. Ecology 83: 2248-2255.

MacKenzie, D. I. et al. 2006. *Occupancy Estimation and Modeling*. Amsterdam: Academic Press.

Moreno, M. and S. R. Lele. 2010. Improved estimation of site occupancy using penalized likelihood. Ecology 91: 341-346.

Royle, J. A. and R. Dorazio. 2008. *Hierarchical Modeling and Inference in Ecology*. Academic Press.

See Also

unmarked, unmarkedFrameOccu, occu, computeMPLElambda, occuPEN_CV, nonparboot

Examples

```
# Simulate occupancy data
set.seed(344)
nSites <- 100
nReps <- 2
covariates <- data.frame(veght=rnorm(nSites),
    habitat=factor(c(rep('A', nSites/2), rep('B', nSites/2))))

psipars <- c(-1, 1, -1)
ppars <- c(1, -1, 0)
X <- model.matrix(~veght+habitat, covariates) # design matrix
psi <- plogis(X %*% psipars)
p <- plogis(X %*% ppars)

y <- matrix(NA, nSites, nReps)
z <- rbinom(nSites, 1, psi)       # true occupancy state
for(i in 1:nSites) {
    y[i,] <- rbinom(nReps, 1, z[i]*p[i])
    }

# Organize data and look at it
umf <- unmarkedFrameOccu(y = y, siteCovs = covariates)
obsCovs(umf) <- covariates
head(umf)
summary(umf)


# Fit some models
fmMLE <- occu(~veght+habitat ~veght+habitat, umf)
fm1pen <- occuPEN(~veght+habitat ~veght+habitat, umf,lambda=0.33,pen.type="Ridge")
fm2pen <- occuPEN(~veght+habitat ~veght+habitat, umf,lambda=1,pen.type="Bayes")

# MPLE:
fm3pen <- occuPEN(~veght+habitat ~veght+habitat, umf,lambda=0.5,pen.type="MPLE")
MPLElambda = computeMPLElambda(~veght+habitat ~veght+habitat, umf)
fm4pen <- occuPEN(~veght+habitat ~veght+habitat, umf,lambda=MPLElambda,pen.type="MPLE")

# nonparametric bootstrap for uncertainty analysis:
fm1pen <- nonparboot(fm1pen,B=20) # should use more samples
vcov(fm1pen,method="nonparboot")
```

occuPEN_CV *Fit the MacKenzie et al. (2002) Occupancy Model with the penalized likelihood methods of Hutchinson et al. (2015) using cross-validation*

Description

This function fits the occupancy model of MacKenzie et al (2002) with the penalized methods of Hutchinson et al (2015) using k-fold cross-validation to choose the penalty weight.

Usage

```
occuPEN_CV(formula, data, knownOcc=numeric(0), starts, method="BFGS",
    engine=c("C", "R"), lambdaVec=c(0,2^seq(-4,4)),
    pen.type = c("Bayes","Ridge"), k = 5, foldAssignments = NA,
    ...)
```

Arguments

formula	Double right-hand side formula describing covariates of detection and occupancy in that order.
data	An unmarkedFrameOccu object
knownOcc	Vector of sites that are known to be occupied. These should be supplied as row numbers of the y matrix, eg, c(3,8) if sites 3 and 8 were known to be occupied a priori.
starts	Vector of parameter starting values.
method	Optimization method used by optim.
engine	Either "C" or "R" to use fast C++ code or native R code during the optimization.
lambdaVec	Vector of values to try for lambda.
pen.type	Which form of penalty to use.
k	Number of folds for k-fold cross-validation.
foldAssignments	Vector containing the number of the fold that each site falls into. Length of the vector should be equal to the number of sites, and the vector should contain k unique values. E.g. for 9 sites and 3 folds, c(1,2,3,1,2,3,1,2,3) or c(1,1,1,2,2,2,3,3,3).
...	Additional arguments to optim, such as lower and upper bounds

Details

See unmarkedFrame and unmarkedFrameOccu for a description of how to supply data to the data argument.

This function wraps k-fold cross-validation around occuPEN_CV for the "Bayes" and "Ridge" penalties of Hutchinson et al. (2015). The user may specify the number of folds (k), the values to try (lambdaVec), and the assignments of sites to folds (foldAssignments). If foldAssignments is not provided, the assignments are done pseudo-randomly, and the function attempts to put some sites with and without positive detections in each fold. This randomness introduces variability into the results of this function across runs; to eliminate the randomness, supply foldAssignments.

Value

unmarkedFitOccuPEN_CV object describing the model fit.

Author(s)

Rebecca A. Hutchinson

References

Hutchinson, R. A., J. V. Valente, S. C. Emerson, M. G. Betts, and T. G. Dietterich. 2015. Penalized Likelihood Methods Improve Parameter Estimates in Occupancy Models. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.12368

MacKenzie, D. I., J. D. Nichols, G. B. Lachman, S. Droege, J. Andrew Royle, and C. A. Langtimm. 2002. Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One. Ecology 83: 2248-2255.

See Also

unmarked, unmarkedFrameOccu, occu, occuPEN, nonparboot

Examples

```
# Simulate occupancy data
set.seed(646)
nSites <- 60
nReps <- 2
covariates <- data.frame(veght=rnorm(nSites),
    habitat=factor(c(rep('A', 30), rep('B', 30))))

psipars <- c(-1, 1, -1)
ppars <- c(1, -1, 0)
X <- model.matrix(~veght+habitat, covariates) # design matrix
psi <- plogis(X %*% psipars)
p <- plogis(X %*% ppars)

y <- matrix(NA, nSites, nReps)
z <- rbinom(nSites, 1, psi)       # true occupancy state
for(i in 1:nSites) {
    y[i,] <- rbinom(nReps, 1, z[i]*p[i])
    }

# Organize data and look at it
umf <- unmarkedFrameOccu(y = y, siteCovs = covariates)
obsCovs(umf) <- covariates
head(umf)
summary(umf)

## Not run:

# Fit some models
```

```
fmMLE <- occu(~veght+habitat ~veght+habitat, umf)
fmMLE@estimates

fm1penCV <- occuPEN_CV(~veght+habitat ~veght+habitat,
 umf,pen.type="Ridge", foldAssignments=rep(1:5,ceiling(nSites/5))[1:nSites])
fm1penCV@lambdaVec
fm1penCV@chosenLambda
fm1penCV@estimates

fm2penCV <- occuPEN_CV(~veght+habitat ~veght+habitat,
umf,pen.type="Bayes",foldAssignments=rep(1:5,ceiling(nSites/5))[1:nSites])
fm2penCV@lambdaVec
fm2penCV@chosenLambda
fm2penCV@estimates

# nonparametric bootstrap for uncertainty analysis:
# bootstrap is wrapped around the cross-validation
fm2penCV <- nonparboot(fm2penCV,B=10) # should use more samples
vcov(fm2penCV,method="nonparboot")

# Mean squared error of parameters:
mean((c(psipars,ppars)-c(fmMLE[1]@estimates,fmMLE[2]@estimates))^2)
mean((c(psipars,ppars)-c(fm1penCV[1]@estimates,fm1penCV[2]@estimates))^2)
mean((c(psipars,ppars)-c(fm2penCV[1]@estimates,fm2penCV[2]@estimates))^2)

## End(Not run)
```

---

occuRN *Fit the occupancy model of Royle and Nichols (2003)*

Description

Fit the occupancy model of Royle and Nichols (2003), which relates probability of detection of the species to the number of individuals available for detection at each site. Probability of occupancy is a derived parameter: the probability that at least one individual is available for detection at the site.

Usage

```
occuRN(formula, data, K=25, starts, method="BFGS", se=TRUE,
              engine=c("C","R"), threads=1, ...)
```

Arguments

formula	double right-hand side formula describing covariates of detection and abundance, in that order.
data	Object of class unmarkedFrameOccu supplying data to the model.

K	the upper summation index used to numerically integrate out the latent abundance. This should be set high enough so that it does not affect the parameter estimates. Computation time will increase with K.
starts	initial values for the optimization.
method	Optimization method used by optim.
se	logical specifying whether or not to compute standard errors.
engine	Either "C" to use fast C++ code or "R" to use native R code during the optimization.
threads	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If threads=1 (the default), OpenMP is disabled.
...	Additional arguments to optim, such as lower and upper bounds

Details

This function fits the latent abundance mixture model described in Royle and Nichols (2003).

The number of animals available for detection at site i is modelled as Poisson:

$$N_i \sim Poisson(\lambda_i)$$

We assume that all individuals at site i during sample j have identical detection probabilities, r_{ij}, and that detections are independent. The species will be recorded if at least one individual is detected. Thus, the detection probability for the species is linked to the detection probability for an individual by

$$p_{ij} = 1 - (1 - r_{ij})^{N_i}$$

Note that if $N_i = 0$, then $p_{ij} = 0$, and increasing values of N_i lead to higher values of p_{ij} The equation for the detection history is then:

$$y_{ij} \sim Bernoulli(p_{ij})$$

Covariates of λ_i are modelled with the log link and covariates of r_{ij} are modelled with the logit link.

Value

unmarkedFit object describing the model fit.

Author(s)

Ian Fiske

References

Royle, J. A. and Nichols, J. D. (2003) Estimating Abundance from Repeated Presence-Absence Data or Point Counts. *Ecology*, 84(3) pp. 777–790.

Examples

```
## Not run:

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

# Empirical Bayes estimates of abundance at each site
re <- ranef(fm.wood.rn)
plot(re)



## End(Not run)
```

---

occuTTD *Fit Single-Season and Dynamic Time-to-detection Occupancy Models*

Description

Fit time-to-detection occupancy models of Garrard et al. (2008, 2013), either single-season or dynamic. Time-to-detection can be modeled with either an exponential or Weibull distribution.

Usage

```
occuTTD(psiformula= ~1, gammaformula =  ~ 1, epsilonformula = ~ 1,
    detformula = ~ 1, data, ttdDist = c("exp", "weibull"),
    linkPsi = c("logit", "cloglog"), starts, method="BFGS", se=TRUE,
    engine = c("C", "R"), ...)
```

Arguments

psiformula	Right-hand sided formula for the initial probability of occupancy at each site.
gammaformula	Right-hand sided formula for colonization probability.
epsilonformula	Right-hand sided formula for extinction probability.
detformula	Right-hand sided formula for mean time-to-detection.
data	unmarkedFrameOccuTTD object that supplies the data (see unmarkedFrameOccuTTD).
ttdDist	Distribution to use for time-to-detection; either "exp" for the exponential, or "weibull" for the Weibull, which adds an additional shape parameter k.
linkPsi	Link function for the occupancy model. Options are "logit" for the standard occupancy model or "cloglog" for the complimentary log-log link, which relates occupancy to site-level abundance.
starts	optionally, initial values for parameters in the optimization.

method	Optimization method used by optim.
se	logical specifying whether or not to compute standard errors.
engine	Either "C" or "R" to use fast C++ code or native R code during the optimization.
...	Additional arguments to optim, such as lower and upper bounds

Details

Estimates site occupancy and detection probability from time-to-detection (TTD) data, e.g. time to first detection of a particular bird species during a point count or time-to-detection of a plant species while searching a quadrat (Garrard et al. 2008). Time-to-detection can be modeled as an exponential (ttdDist="exp") or Weibull (ttdDist="weibull") random variable with rate parameter λ and, for the Weibull, an additional shape parameter k. Note that occuTTD puts covariates on λ and not $1/\lambda$, i.e., the expected time between events.

In the case where there are no detections before the maximum sample time at a site (surveyLength) is reached, we are not sure if the site is unoccupied or if we just didn't wait long enough for a detection. We therefore must censor the exponential or Weibull distribution at the maximum survey length, $Tmax$. Thus, assuming true site occupancy at site i is z_i, an exponential distribution for the TTD y_i, and that $d_i = 1$ indicates y_i is censored (Kery and Royle 2016):

$$d_i = z_i * I(y_i > Tmax_i) + (1 - z_i)$$

and

$$y_i|z_i \sim Exponential(\lambda_i), d_i = 0$$
$$y_i|z_i = Missing, d_i = 1$$

Because in unmarked values of NA are typically used to indicate missing values that were a result of the sampling structure (e.g., lost data), we indicate a censored y_i in occuTTD instead by setting $y_i = Tmax_i$ in the y matrix provided to unmarkedFrameOccuTTD. You can provide either a single value of $Tmax$ to the surveyLength argument of unmarkedFrameOccuTTD, or provide a matrix, potentially with a unique value of $Tmax$ for each value of y. Note that in the latter case the value of y that will be interpreted by occuTTD as a censored observation (i.e., $Tmax$) will differ between observations!

Occupancy and detection can be estimated with only a single survey per site, unlike a traditional occupancy model that requires at least two replicated surveys at at least some sites. However, occuTTD also supports multiple surveys per site using the model described in Garrard et al. (2013). Furthermore, multi-season dynamic models are supported, using the same basic structure as for standard occupancy models (see colext).

When linkPsi = "cloglog", the complimentary log-log link function is used for psi instead of the logit link. The cloglog link relates occupancy probability to the intensity parameter of an underlying Poisson process (Kery and Royle 2016). Thus, if abundance at a site is can be modeled as N_i $Poisson(\lambda_i)$, where $log(\lambda_i) = \alpha + \beta * x$, then presence/absence data at the site can be modeled as Z_i $Binomial(\psi_i)$ where $cloglog(\psi_i) = \alpha + \beta * x$.

Value

unmarkedFitOccuTTD object describing model fit.

Author(s)

Ken Kellner <contact@kenkellner.com>

References

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

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

Kery, Marc, and J. Andrew Royle. 2016. *Applied Hierarchical Modeling in Ecology*, Volume 1. Academic Press.

See Also

unmarked, unmarkedFrameOccuTTD

Examples

```
## Not run:

### Single season model
N <- 500; J <- 1

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

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

#Simulate detection
Tmax <- 10 #Same survey length for all observations
beta_lam <- c(-2, -0.2, 0.7)
rate <- exp(cbind(1, scovs$elev, scovs$wind) %*% beta_lam)
ttd <- rexp(N, rate)
ttd[z==0] <- Tmax #Censor at unoccupied sites
ttd[ttd>Tmax] <- Tmax #Censor when ttd was greater than survey length

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

#Fit model
fit <- occuTTD(psiformula=~elev+forest, detformula=~elev+wind, data=umf)

#Predict psi values
predict(fit, type='psi', newdata=data.frame(elev=0.5, forest=1))
```

```
#Predict lambda values
predict(fit, type='det', newdata=data.frame(elev=0.5, wind=0))

#Calculate p, probability species is detected at a site given it is present
#for a value of lambda. This is equivalent to eq 4 of Garrard et al. 2008
lam <- predict(fit, type='det', newdata=data.frame(elev=0.5, wind=0))$Predicted
pexp(Tmax, lam)

#Estimated p for all observations
head(getP(fit))

### Dynamic model

N <- 1000; J <- 2; T <- 2
scovs <- data.frame(elev=c(scale(runif(N, 0,100))),
                    forest=runif(N,0,1),
                    wind=runif(N,0,1))

beta_psi <- c(-0.69, 0.71, -0.5)
psi <- plogis(cbind(1, scovs$elev, scovs$forest) %*% beta_psi)
z <- matrix(NA, N, T)
z[,1] <- rbinom(N, 1, psi)

#Col/ext process
ysc <- data.frame(forest=rep(scovs$forest, each=T),
                  elev=rep(scovs$elev, each=T))
c_b0 <- -0.4; c_b1 <- 0.3
gam <- plogis(c_b0 + c_b1 * scovs$forest)
e_b0 <- -0.7; e_b1 <- 0.4
ext <- plogis(e_b0 + e_b1 * scovs$elev)

for (i in 1:N){
  for (t in 1:(T-1)){
    if(z[i,t]==1){
      #ext
      z[i,t+1] <- rbinom(1, 1, (1-ext[i]))
    } else {
      #col
      z[i,t+1] <- rbinom(1,1, gam[i])
    }
  }
}

#Simulate detection
ocovs <- data.frame(obs=rep(c('A','B'),N*T))
Tmax <- 10
beta_lam <- c(-2, -0.2, 0.7)
rate <- exp(cbind(1, scovs$elev, scovs$wind) %*% beta_lam)
#Add second observer at each site
rateB <- exp(cbind(1, scovs$elev, scovs$wind) %*% beta_lam - 0.5)
#Across seasons
rate2 <- as.numeric(t(cbind(rate, rateB, rate, rateB)))
ttd <- rexp(N*T*2, rate2)
```

```
ttd <- matrix(ttd, nrow=N, byrow=T)
ttd[ttd>Tmax] <- Tmax
ttd[z[,1]==0,1:2] <- Tmax
ttd[z[,2]==0,3:4] <- Tmax

umf <- unmarkedFrameOccuTTD(y = ttd, surveyLength = Tmax,
                            siteCovs = scovs, obsCovs=ocovs,
                            yearlySiteCovs=ysc, numPrimary=2)

dim(umf@y) #num sites, (num surveys x num primary periods)

fit <- occuTTD(psiformula=~elev+forest,detformula=~elev+wind+obs,
               gammaformula=~forest, epsilonformula=~elev,
               data=umf,se=T,engine="C")

truth <- c(beta_psi, c_b0, c_b1, e_b0, e_b1, beta_lam, -0.5)

#Compare to truth
cbind(coef(fit), truth)


## End(Not run)
```

`optimizePenalty-methods`

Identify Optimal Penalty Parameter Value

Description

Identify the optimal value of the penalty term for `unmarked` models that support penalized likelihood. For each potential value of the penalty term, K-fold cross validation is performed. Log-likelihoods for the test data in each fold are calculated and summed. The penalty term that maximizes the sum of the fold log-likelihoods is selected as the optimal value. Finally, the model is re-fit with the full dataset using the selected penalty term. Right now only Bayes-inspired penalty of Hutchinson et al. (2015) is supported.

Currently the only fitting function that supports `optimizePenalty` is `occuMulti` for multispecies occupancy modeling; see Clipp et al. (2021).

Usage

```
## S4 method for signature 'unmarkedFitOccuMulti'
optimizePenalty(
  object, penalties = c(0, 2^seq(-4, 4)), k = 5, boot = 30, ...)
```

Arguments

`object`	A fitted model inheriting class `unmarkedFit`
`penalties`	Vector of possible penalty values, all of which must be >= 0

k	Number of folds to use for k-fold cross validation
boot	Number of bootstrap samples to use to generate the variance-covariance matrix for the final model.
...	Other arguments, currently ignored

Value

unmarkedFit object of same type as input, with the optimal penalty value applied.

Author(s)

Ken Kellner <contact@kenkellner.com>

References

Clipp, H. L., Evans, A., Kessinger, B. E., Kellner, K. F., and C. T. Rota. 2021. A penalized likelihood for multi-species occupancy models improves predictions of species interactions. Ecology.

Hutchinson, R. A., J. V. Valente, S. C. Emerson, M. G. Betts, and T. G. Dietterich. 2015. Penalized Likelihood Methods Improve Parameter Estimates in Occupancy Models. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.12368

---

ovendata *Removal data for the Ovenbird*

Description

Removal sampling data collected for the Ovenbird (*Seiurus aurocapillus*).

Usage

```
data(ovendata)
```

Format

The format is: chr "ovendata.list" which consists of

data matrix of removal counts

covariates data frame of site-level covariates

Source

J.A. Royle (see reference below)

References

Royle, J. A. (2004). Generalized estimators of avian abundance from count survey data. Animal Biodiversity and Conservation, 27(1), 375-386.

Examples

```
data(ovendata)
str(ovendata.list)
ovenFrame <- unmarkedFrameMPois(ovendata.list$data,
siteCovs=as.data.frame(scale(ovendata.list$covariates[,-1])), type = "removal")
```

`parboot` *Parametric bootstrap method for fitted models inheriting class.*

Description

Simulate datasets from a fitted model, refit the model, and generate a sampling distribution for a user-specified fit-statistic.

Arguments

`object`	a fitted model inheriting class "unmarkedFit"
`statistic`	a function returning a vector of fit-statistics. First argument must be the fitted model. Default is sum of squared residuals.
`nsim`	number of bootstrap replicates
`report`	print fit statistic every 'report' iterations during resampling
`seed`	set seed for reproducible bootstrap
`parallel`	logical (default = `TRUE`) indicating whether to compute bootstrap on multiple cores, if present. If `TRUE`, suppresses reporting of bootstrapped statistics. Defaults to serial calculation when `nsim` < 100. Parallel computation is likely to be slower for simple models when `nsim` < ~500, but should speed up the bootstrap of more complicated models.
`ncores`	integer (default = one less than number of available cores) number of cores to use when bootstrapping in parallel.
`...`	Additional arguments to be passed to statistic

Details

This function simulates datasets based upon a fitted model, refits the model, and evaluates a user-specified fit-statistic for each simulation. Comparing this sampling distribution to the observed statistic provides a means of evaluating goodness-of-fit or assessing uncertainty in a quantity of interest.

Value

An object of class parboot with three slots:

`call`	parboot call
`t0`	Numeric vector of statistics for original fitted model.
`t.star`	nsim by length(t0) matrix of statistics for each simulation fit.

Author(s)

Richard Chandler <rbchan@uga.edu> and Adam Smith

See Also

ranef

Examples

```
data(linetran)
(dbreaksLine <- c(0, 5, 10, 15, 20))
lengths <- linetran$Length

ltUMF <- with(linetran, {
unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
siteCovs = data.frame(Length, area, habitat), dist.breaks = dbreaksLine,
tlength = lengths*1000, survey = "line", unitsIn = "m")
    })

# Fit a model
(fm <- distsamp(~area ~habitat, ltUMF))

# Function returning three fit-statistics.
fitstats <- function(fm, na.rm=TRUE) {
    observed <- getY(fm@data)
    expected <- fitted(fm)
    resids <- residuals(fm)
    sse <- sum(resids^2, na.rm=na.rm)
    chisq <- sum((observed - expected)^2 / expected, na.rm=na.rm)
    freeTuke <- sum((sqrt(observed) - sqrt(expected))^2, na.rm=na.rm)
    out <- c(SSE=sse, Chisq=chisq, freemanTukey=freeTuke)
    return(out)
}

(pb <- parboot(fm, fitstats, nsim=25, report=1))
plot(pb, main="")


# Finite-sample inference for a derived parameter.
# Population size in sampled area

Nhat <- function(fm) {
    sum(bup(ranef(fm, K=50)))
    }

set.seed(345)
(pb.N <- parboot(fm, Nhat, nsim=25, report=5))

# Compare to empirical Bayes confidence intervals
colSums(confint(ranef(fm, K=50)))
```

pcount *Fit the N-mixture model of Royle (2004)*

Description

Fit the N-mixture model of Royle (2004)

Usage

```
pcount(formula, data, K, mixture=c("P", "NB", "ZIP"),
    starts, method="BFGS", se=TRUE, engine=c("C", "R", "TMB"), threads=1, ...)
```

Arguments

formula	Double right-hand side formula describing covariates of detection and abundance, in that order
data	an unmarkedFramePCount object supplying data to the model.
K	Integer upper index of integration for N-mixture. This should be set high enough so that it does not affect the parameter estimates. Note that computation time will increase with K.
mixture	character specifying mixture: "P", "NB", or "ZIP".
starts	vector of starting values
method	Optimization method used by optim.
se	logical specifying whether or not to compute standard errors.
engine	Either "C", "R", or "TMB" to use fast C++ code, native R code, or TMB (required for random effects) during the optimization.
threads	Set the number of threads to use for optimization in C++, if OpenMP is available on your system. Increasing the number of threads may speed up optimization in some cases by running the likelihood calculation in parallel. If threads=1 (the default), OpenMP is disabled.
...	Additional arguments to optim, such as lower and upper bounds

Details

This function fits N-mixture model of Royle (2004) to spatially replicated count data.

See unmarkedFramePCount for a description of how to format data for pcount.

This function fits the latent N-mixture model for point count data (Royle 2004, Kery et al 2005).

The latent abundance distribution, $f(N|\theta)$ can be set as a Poisson, negative binomial, or zero-inflated Poisson random variable, depending on the setting of the mixture argument, mixture = "P", mixture = "NB", mixture = "ZIP" respectively. For the first two distributions, the mean of N_i

is λ_i. If $N_i \sim NB$, then an additional parameter, α, describes dispersion (lower α implies higher variance). For the ZIP distribution, the mean is $\lambda_i(1-\psi)$, where psi is the zero-inflation parameter.

The detection process is modeled as binomial: $y_{ij} \sim Binomial(N_i, p_{ij})$.

Covariates of λ_i use the log link and covariates of p_{ij} use the logit link.

Value

unmarkedFit object describing the model fit.

Author(s)

Ian Fiske and Richard Chandler

References

Royle, J. A. (2004) N-Mixture Models for Estimating Population Size from Spatially Replicated Counts. *Biometrics* 60, pp. 108–105.

Kery, M., Royle, J. A., and Schmid, H. (2005) Modeling Avaian Abundance from Replicated Counts Using Binomial Mixture Models. *Ecological Applications* 15(4), pp. 1450–1461.

Johnson, N.L, A.W. Kemp, and S. Kotz. (2005) Univariate Discrete Distributions, 3rd ed. Wiley.

See Also

unmarkedFramePCount, pcountOpen, ranef, parboot

Examples

```
## Not run:

# Simulate data
set.seed(35)
nSites <- 100
nVisits <- 3
x <- rnorm(nSites)               # a covariate
beta0 <- 0
beta1 <- 1
lambda <- exp(beta0 + beta1*x)   # expected counts at each site
N <- rpois(nSites, lambda)       # latent abundance
y <- matrix(NA, nSites, nVisits)
p <- c(0.3, 0.6, 0.8)            # detection prob for each visit
for(j in 1:nVisits) {
  y[,j] <- rbinom(nSites, N, p[j])
  }

# Organize data
visitMat <- matrix(as.character(1:nVisits), nSites, nVisits, byrow=TRUE)

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

```
summary(umf)

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

plogis(coef(fm1, type="det")) # Should be close to p


# Empirical Bayes estimation of random effects
(fm1re <- ranef(fm1))
plot(fm1re, subset=site %in% 1:25, xlim=c(-1,40))
sum(bup(fm1re))         # Estimated population size
sum(N)                  # Actual population size


# Real data
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
obsCovs = mallard.obs)
(fm.mallard <- pcount(~ ivel+ date + I(date^2) ~ length + elev + forest, mallardUMF, K=30))
(fm.mallard.nb <- pcount(~ date + I(date^2) ~ length + elev, mixture = "NB", mallardUMF, K=30))


## End(Not run)
```

`pcount.spHDS` *Fit spatial hierarchical distance sampling model.*

Description

Function fits an N-mixture model for a discrete state space with raster covariates, and a detection function which decreases with distance from the observer, assumed to be at the centre. See Kery & Royle (2016) Section 9.8.4 for details.

Usage

```
pcount.spHDS(formula, data, K, mixture = c("P", "NB", "ZIP"), starts,
  method = "BFGS", se = TRUE, ...)
```

Arguments

`formula`	Double right-hand side formula describing covariates of detection and abundance, in that order. Detection model should be specified without an intercept, for example: `~ -1 + I(dist^2)`, where `dist` is a covariate giving the distance of each cell of the raster from the observer. Internally this forces the intercept `p(0) = 1`, conventional for distance sampling models (see Kery & Royle (2016) for explanation).

	More general models work but may not honor that constraint. e.g., `~ 1`, `~ dist`, `~ I(dist^2)`, `~ dist + I(dist^2)`
`data`	an `unmarkedFramePCount` object supplying data to the model.
`K`	Integer upper index of integration for N-mixture. This should be set high enough so that it does not affect the parameter estimates. Note that computation time will increase with K.
`mixture`	character specifying mixture: Poisson (P), Negative-Binomial (NB), or Zero Inflated Poisson (ZIP).
`starts`	vector of starting values
`method`	Optimization method used by `optim`.
`se`	logical specifying whether or not to compute standard errors.
`...`	Additional arguments to optim, such as lower and upper bounds

Value

unmarkedFit object describing the model fit.

Author(s)

Kery & Royle

References

Kery & Royle (2016) *Applied Hierarachical Modeling in Ecology* Section 9.8.4

Examples

```
## Simulate some data to analyse
# This is based on Kery and Royle (2016) section 9.8.3
# See AHMbook::sim.spatialDS for more simulation options.

# We will simulate distance data for a logit detection function with sigma = 1,
# for a 6x6 square, divided into a 30 x 30 grid of pixels (900 in all), with the
# observer in the centre.

set.seed(2017)

## 1. Create coordinates for 30 x 30 grid
grx <- seq(0.1, 5.9, 0.2)    # mid-point coordinates
gr <- expand.grid(grx, grx)  # data frame with coordinates of pixel centres

## 2a. Simulate spatially correlated Habitat covariate
# Get the pair-wise distances between pixel centres
tmp <- as.matrix(dist(gr))  # a 900 x 900 matrix
# Correlation is a negative exponential function of distance, with scale parameter = 1
V <- exp(-tmp/1)
Habitat <- crossprod(t(chol(V)), rnorm(900))

## 2b. Do a detection covariate: the distance of each pixel centre from the observer
```

```
dist <- sqrt((gr[,1]-3)^2 + (gr[,2]-3)^2)

## 3. Simulate the true population
# Probability that an animal is in a pixel depends on the Habitat covariate, with
#   coefficient beta:
beta <- 1
probs <- exp(beta*Habitat) / sum(exp(beta*Habitat))
# Allocate 600 animals to the 900 pixels, get the pixel ID for each animal
pixel.id <- sample(1:900, 600, replace=TRUE, prob=probs)

## 4. Simulate the detection process
# Get the distance of each animal from the observer
# (As an approximation, we'll treat animals as if they are at the pixel centre.)
d <- dist[pixel.id]
# Calculate probability of detection with logit detection function with
sigma <- 1
p <- 2*plogis(-d^2/(2*sigma^2))
# Simulate the 1/0 detection/nondetection vector
y <- rbinom(600, 1, p)
# Check the number of animals detected
sum(y)
# Select the pixel IDs for the animals detected and count the number in each pixel
detected.pixel.id <- pixel.id[y == 1]
pixel.count <- tabulate(detected.pixel.id, nbins=900)

## 5. Prepare the data for unmarked
# Centre the Habitat covariate
Habitat <- Habitat - mean(Habitat)
# Construct the unmarkedFramePCount object
umf <- unmarkedFramePCount(y=cbind(pixel.count),     # y needs to be a 1-column matrix
   siteCovs=data.frame(dist=dist, Habitat=Habitat))
summary(umf)

## 6. Fit some models
(fm0 <- pcount.spHDS(~ -1 + I(dist^2) ~ 1, umf, K = 20))
(fm1 <- pcount.spHDS(~ -1 + I(dist^2) ~ Habitat, umf, K = 20))
# The true Habitat coefficient (beta above) = 1
# fm1 has much lower AIC; look at the population estimate
sum(predict(fm1, type="state")[, 1])
```

---

pcountOpen *Fit the open N-mixture models of Dail and Madsen and extensions*

Description

Fit the models of Dail and Madsen (2011) and Hostetler and Chandler (in press), which are generalized forms of the Royle (2004) N-mixture model for open populations.

Usage

```
pcountOpen(lambdaformula, gammaformula, omegaformula, pformula,
  data, mixture = c("P", "NB", "ZIP"), K, dynamics=c("constant", "autoreg",
  "notrend", "trend", "ricker", "gompertz"), fix=c("none", "gamma", "omega"),
  starts, method = "BFGS", se = TRUE, immigration = FALSE,
  iotaformula = ~1, ...)
```

Arguments

lambdaformula	Right-hand sided formula for initial abundance
gammaformula	Right-hand sided formula for recruitment rate (when dynamics is "constant", "autoreg", or "notrend") or population growth rate (when dynamics is "trend", "ricker", or "gompertz")
omegaformula	Right-hand sided formula for apparent survival probability (when dynamics is "constant", "autoreg", or "notrend") or equilibrium abundance (when dynamics is "ricker" or "gompertz")
pformula	Right-hand sided formula for detection probability
data	An object of class unmarkedFramePCO. See details
mixture	character specifying mixture: "P", "NB", or "ZIP" for the Poisson, negative binomial, and zero-inflated Poisson distributions.
K	Integer defining upper bound of discrete integration. This should be higher than the maximum observed count and high enough that it does not affect the parameter estimates. However, the higher the value the slower the compuatation.
dynamics	Character string describing the type of population dynamics. "constant" indicates that there is no relationship between omega and gamma. "autoreg" is an auto-regressive model in which recruitment is modeled as gamma*N[i,t-1]. "notrend" model gamma as lambda*(1-omega) such that there is no temporal trend. "trend" is a model for exponential growth, N[i,t] = N[i,t-1]*gamma, where gamma in this case is finite rate of increase (normally referred to as lambda). "ricker" and "gompertz" are models for density-dependent population growth. "ricker" is the Ricker-logistic model, N[i,t] = N[i,t-1]*exp(gamma*(1-N[i,t-1]/omega)), where gamma is the maximum instantaneous population growth rate (normally referred to as r) and omega is the equilibrium abundance (normally referred to as K). "gompertz" is a modified version of the Gompertz-logistic model, N[i,t] = N[i,t-1]*exp(gamma*(1-log(N[i,t-1]+1)/log(omega+1))), where the interpretations of gamma and omega are similar to in the Ricker model.
fix	If "omega", omega is fixed at 1. If "gamma", gamma is fixed at 0.
starts	vector of starting values
method	Optimization method used by optim.
se	logical specifying whether or not to compute standard errors.
immigration	logical specifying whether or not to include an immigration term (iota) in population dynamics.
iotaformula	Right-hand sided formula for average number of immigrants to a site per time step
...	additional arguments to be passed to optim.

Details

These models generalize the Royle (2004) N-mixture model by relaxing the closure assumption. The models include two or three additional parameters: gamma, either the recruitment rate (births and immigrations), the finite rate of increase, or the maximum instantaneous rate of increase; omega, either the apparent survival rate (deaths and emigrations) or the equilibrium abundance (carrying capacity); and iota, the number of immigrants per site and year. Estimates of population size at each time period can be derived from these parameters, and thus so can trend estimates. Or, trend can be estimated directly using dynamics="trend".

When immigration is set to FALSE (the default), iota is not modeled. When immigration is set to TRUE and dynamics is set to "autoreg", the model will separately estimate birth rate (gamma) and number of immigrants (iota). When immigration is set to TRUE and dynamics is set to "trend", "ricker", or "gompertz", the model will separately estimate local contributions to population growth (gamma and omega) and number of immigrants (iota).

The latent abundance distribution, $f(N|\theta)$ can be set as a Poisson, negative binomial, or zero-inflated Poisson random variable, depending on the setting of the `mixture` argument, `mixture = "P"`, `mixture = "NB"`, `mixture = "ZIP"` respectively. For the first two distributions, the mean of N_i is λ_i. If $N_i \sim NB$, then an additional parameter, α, describes dispersion (lower α implies higher variance). For the ZIP distribution, the mean is $\lambda_i(1-\psi)$, where psi is the zero-inflation parameter.

For "constant", "autoreg", or "notrend" dynamics, the latent abundance state following the initial sampling period arises from a Markovian process in which survivors are modeled as $S_{it} \sim Binomial(N_{it-1}, \omega_{it})$, and recruits follow $G_{it} \sim Poisson(\gamma_{it})$. Alternative population dynamics can be specified using the `dynamics` and `immigration` arguments.

The detection process is modeled as binomial: $y_{ijt} \sim Binomial(N_{it}, p_{ijt})$.

λ_i, γ_{it}, and ι_{it} are modeled using the the log link. p_{ijt} is modeled using the logit link. ω_{it} is either modeled using the logit link (for "constant", "autoreg", or "notrend" dynamics) or the log link (for "ricker" or "gompertz" dynamics). For "trend" dynamics, ω_{it} is not modeled.

Value

An object of class unmarkedFitPCO.

Warning

This function can be extremely slow, especially if there are covariates of gamma or omega. Consider testing the timing on a small subset of the data, perhaps with se=FALSE. Finding the lowest value of K that does not affect estimates will also help with speed.

Note

When gamma or omega are modeled using year-specific covariates, the covariate data for the final year will be ignored; however, they must be supplied.

If the time gap between primary periods is not constant, an M by T matrix of integers should be supplied to `unmarkedFramePCO` using the `primaryPeriod` argument.

Secondary sampling periods are optional, but can greatly improve the precision of the estimates.

Author(s)

Richard Chandler <rbchan@uga.edu> and Jeff Hostetler

References

Royle, J. A. (2004) N-Mixture Models for Estimating Population Size from Spatially Replicated Counts. *Biometrics* 60, pp. 108–105.

Dail, D. and L. Madsen (2011) Models for Estimating Abundance from Repeated Counts of an Open Metapopulation. *Biometrics*. 67, pp 577-587.

Hostetler, J. A. and R. B. Chandler (2015) Improved State-space Models for Inference about Spatial and Temporal Variation in Abundance from Count Data. *Ecology* 96:1713-1723.

See Also

pcount, unmarkedFramePCO

Examples

```
## Simulation
## No covariates, constant time intervals between primary periods, and
## no secondary sampling periods

set.seed(3)
M <- 50
T <- 5
lambda <- 4
gamma <- 1.5
omega <- 0.8
p <- 0.7
y <- N <- matrix(NA, M, T)
S <- G <- matrix(NA, M, T-1)
N[,1] <- rpois(M, lambda)
for(t in 1:(T-1)) {
S[,t] <- rbinom(M, N[,t], omega)
G[,t] <- rpois(M, gamma)
N[,t+1] <- S[,t] + G[,t]
}
y[] <- rbinom(M*T, N, p)


# Prepare data
umf <- unmarkedFramePCO(y = y, numPrimary=T)
summary(umf)


# Fit model and backtransform
(m1 <- pcountOpen(~1, ~1, ~1, ~1, umf, K=20)) # Typically, K should be higher

(lam <- coef(backTransform(m1, "lambda"))) # or
lam <- exp(coef(m1, type="lambda"))
```

```
gam <- exp(coef(m1, type="gamma"))
om <- plogis(coef(m1, type="omega"))
p <- plogis(coef(m1, type="det"))

## Not run:
# Finite sample inference. Abundance at site i, year t
re <- ranef(m1)
devAskNewPage(TRUE)
plot(re, layout=c(5,5), subset = site %in% 1:25 & year %in% 1:2,
     xlim=c(-1,15))
devAskNewPage(FALSE)

(N.hat1 <- colSums(bup(re)))

# Expected values of N[i,t]
N.hat2 <- matrix(NA, M, T)
N.hat2[,1] <- lam
for(t in 2:T) {
    N.hat2[,t] <- om*N.hat2[,t-1] + gam
    }

rbind(N=colSums(N), N.hat1=N.hat1, N.hat2=colSums(N.hat2))



## End(Not run)
```

---

piFuns *Compute multinomial cell probabilities*

Description

Compute the cell probabilities used in the multinomial-Poisson models multinomPois and gmultmix. These functions use piFuns *internally* to calculate multinomial likelihoods from the occasion-wise detection probabilities. The only reason to call them directly is to check their behaviour.

Usage

```
removalPiFun(p)
doublePiFun(p)
```

Arguments

p	matrix of detection probabilities at each site for each observation

Details

These two functions are provided as examples of possible functions to calculate multinomial cell probabilities. Users may write their own functions for specific sampling designs (see the example).

Value

For removalPiFun, a matrix of cell probabilities for each site and sampling period.

For doublePiFun, a matrix of cell probabilities for each site and observer combination. Column one is probability observer 1 but not observer 2 detects the object, column two is probability that observer 2 but not observer 1 detects the object, and column 3 is probability of both detecting.

See Also

makePiFuns for factory functions to create customised piFuns.

Examples

```
(pRem <- matrix(0.5, nrow=3, ncol=3)) # Capture probabilities
removalPiFun(pRem) # Cell probs

(pDouble <- matrix(0.5, 3, 2)) # Observer detection probs
doublePiFun(pDouble) # Cell probs

# A user-defined piFun calculating removal probs when time intervals differ.
# Here 10-minute counts were divided into 2, 3, and 5 minute intervals.
# This function could be supplied to unmarkedFrameMPois along with the obsToY
# argument shown below.

instRemPiFun <- function(p) {
M <- nrow(p)
J <- ncol(p)
pi <- matrix(NA, M, J)
p[,1] <- pi[,1] <- 1 - (1 - p[,1])^2
p[,2] <- 1 - (1 - p[,2])^3
p[,3] <- 1 - (1 - p[,3])^5
for(i in 2:J) {
pi[,i] <- pi[, i - 1]/p[, i - 1] * (1 - p[, i - 1]) * p[, i]
}
return(pi)
}

instRemPiFun(pRem)

# Associated obsToY matrix required by unmarkedFrameMPois
o2y <- diag(3) # if y has 3 columns
o2y[upper.tri(o2y)] <- 1
o2y
```

---

plotEffects *Plot marginal effects of covariates in unmarked models*

Description

This function generates a plot visualizing the effects of a single covariate on a parameter (e.g. occupancy, abundance) in an unmarked model. If the covariate is numeric, the result is a line plot with an error ribbon where the x-axis is the range of the covariate and the y-axis is the predicted parameter value. If the covariate is an R factor (i.e., categorical), the x-axis instead contains each unique value of the covariate.

All covariates in the model besides the one being plotted are held either at their median value (if they are numeric) or at their reference level (if they are factors).

Some types of unmarked models may require additional arguments, which are passed to the matching `predict` method. For example, `unmarkedFitOccuMulti` models require the `species` argument to be included in the function call in order to work properly.

If you want to customize a plot, the easiest approach is to get data formatted for plotting using `plotEffectsData`, and use that. If you want to see and/or modify the code used by `plotEffects` to generate the default plots, run `getMethod("plotEffects", "unmarkedFit")` in the R console.

Usage

```
## S4 method for signature 'unmarkedFit'
plotEffects(object, type, covariate, level=0.95, ...)
## S4 method for signature 'unmarkedFit'
plotEffectsData(object, type, covariate, level=0.95, ...)
```

Arguments

`object`	A fitted model inheriting class `unmarkedFit`
`type`	Submodel in which the covariate of interest can be found, for example `"state"` or `"det"`. This will depend on the fitted model
`covariate`	The name of the covariate to be plotted, as a character string
`level`	Confidence level for the error ribbons or bars
`...`	Other arguments passed to the `predict` function, required for some `unmarkedFit` types such as `unmarkedFitOccuMulti`

Value

A plot (`plotEffects` or a data frame (`plotEffectsData`) containing values to be used in a plot.

Author(s)

Ken Kellner <contact@kenkellner.com>

Examples

```
## Not run:

# Simulate data and build an unmarked frame
set.seed(123)
```

```
dat_occ <- data.frame(x1=rnorm(500))
dat_p <- data.frame(x2=rnorm(500*5))

y <- matrix(NA, 500, 5)
z <- rep(NA, 500)

b <- c(0.4, -0.5, 0.3, 0.5)

re_fac <- factor(sample(letters[1:5], 500, replace=T))
dat_occ$group <- re_fac
re <- rnorm(5, 0, 1.2)
re_idx <- as.numeric(re_fac)

idx <- 1
for (i in 1:500){
  z[i] <- rbinom(1,1, plogis(b[1] + b[2]*dat_occ$x1[i] + re[re_idx[i]]))
  for (j in 1:5){
    y[i,j] <- z[i]*rbinom(1,1,
                  plogis(b[3] + b[4]*dat_p$x2[idx]))
    idx <- idx + 1
  }
}

umf <- unmarkedFrameOccu(y=y, siteCovs=dat_occ, obsCovs=dat_p)

# Fit model
(fm <- occu(~x2 ~x1 + group, umf))

# Plot marginal effects of various covariates
plotEffects(fm, "state", "x1")
plotEffects(fm, "state", "group")
plotEffects(fm, "det", "x2")

# Get raw data used for a plot
plotEffectsData(fm, "state", "group")

# See code used by plotEffects so you can edit it yourself and customize the plot
methods::getMethod("plotEffects", "unmarkedFit")

## End(Not run)
```

`pointtran` *Simulated point-transect data*

Description

Response matrix of animals detected in five distance classes plus two covariates.

Usage

```
data(pointtran)
```

Format

A data frame with 30 observations on the following 7 variables.

`dc1` Counts in distance class 1 [0-5 m)

`dc2` Counts in distance class 2 [5-10 m)

`dc3` Counts in distance class 3 [10-15 m)

`dc4` Counts in distance class 4 [15-20 m)

`dc5` Counts in distance class 5 [20-25 m)

`area` a numeric vector

`habitat` a factor with levels `A` `B` `C`

Examples

```
data(pointtran)
pointtran

# Format for distsamp()
ptUMF <- with(pointtran, {
        unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4, dc5),
        siteCovs = data.frame(area, habitat),
        dist.breaks = seq(0, 25, by=5), survey = "point", unitsIn = "m")
        })
```

---

posteriorSamples *Draw samples from the posterior predictive distribution*

Description

Draw samples from the empirical Bayes posterior predictive distribution derived from unmarked models or ranef objects

Usage

```
## S4 method for signature 'unmarkedRanef'
posteriorSamples(object, nsims=100, ...)
## S4 method for signature 'unmarkedFit'
posteriorSamples(object, nsims=100, ...)
```

Arguments

`object`	An object inheriting class `unmarkedRanef` or `unmarkedFit`
`nsims`	Number of draws to make from the posterior predictive distribution
`...`	Other arguments

Value

unmarkedPostSamples object containing the draws from the posterior predictive distribution. The draws are in the @samples slot.

Author(s)

Ken Kellner <contact@kenkellner.com>

See Also

ranef, predict

Examples

```
# Simulate data under N-mixture model
set.seed(4564)
R <- 20
J <- 5
N <- rpois(R, 10)
y <- matrix(NA, R, J)
y[] <- rbinom(R*J, N, 0.5)

# Fit model
umf <- unmarkedFramePCount(y=y)
fm <- pcount(~1 ~1, umf, K=50)

# Estimates of conditional abundance distribution at each site
(re <- ranef(fm))

#Draw from the posterior predictive distribution
(ppd <- posteriorSamples(re, nsims=100))
```

---

powerAnalysis *Conduct a power analysis on an unmarked model*

Description

This function uses a simulation-based approach to estimate power for parameters in unmarked models. At a minimum, users must provide a fitted unmarked model object (preferably fit with simulated data) which ensures the model has been properly specified, a list of effect sizes for each parameter in the model (coefs), and the desired Type I error (alpha). It is also possible to get power for a range of other sample sizes besides the sample size in the fitted model object using the design argument to subsample within the provided dataset. See the unmarkedPower vignette for more details and examples.

Usage

```
powerAnalysis(object, coefs=NULL, design=NULL, alpha=0.05, nulls=list(),
              datalist=NULL,
              nsim=ifelse(is.null(datalist), 100, length(datalist)),
              parallel=FALSE)
```

Arguments

object	A fitted model inheriting class unmarkedFit. This could potentially be fit using real data, but ideally you would simulate an appropriate dataset using simulate
coefs	A list containing the desired effect sizes for which you want to estimate power. This list must follow a specific format. There is one named entry in the list per submodel (e.g., occupancy, detection). To get the required submodel names call names(object) on your fitted model. Then, each list entry is a named vector with the names corresponding to the parameter names for that submodel, and the values corresponding to the desired effect sizes. It may be easier to leave coefs=NULL, which will generate an error message with a template that you can fill in.
design	An optional list of design/sample size parameters containing at a minimum two named elements: M, the number of sites, and J the number of observations per site. If this list is provided, unmarked will subsample the provided dataset to the specified number of sites and observations, allowing you to test power for different designs. If your model has multiple primary periods you must also include T, the number of periods, in the list.
alpha	Desired Type I error rate
nulls	If provided, a list matching the structure of coefs which defines the null hypothesis value for each parameter. By default the null is 0 for all parameters.
datalist	An optional list of previously-simulated datasets, in the form of unmarkedFrames matching the model type of object, which will be used for the power analysis simulations.
nsim	Number of simulations to conduct
parallel	If TRUE, run folds in parallel. This may speed up the power analysis in some situations

Value

unmarkedPower object containing the results of the power analysis

Author(s)

Ken Kellner <contact@kenkellner.com>

See Also

unmarkedPowerList

Examples

```
## Not run:

# Simulate an occupancy dataset
# Covariates to include in simulation
forms <- list(state=~elev, det=~1)

# Covariate effects and intercept values
coefs <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))

# Study design
design <- list(M=300, J=8) # 300 sites, 8 occasions per site

# Simulate an unmarkedFrameOccu
occu_umf <- simulate("occu", formulas=forms, coefs=coefs, design=design)

# Fit occupancy model to simulated data
# This will contain all the model structure info powerAnalysis needs
# The estimates from the model aren't used
template_model <- occu(~1~elev, occu_umf)

# If we run powerAnalysis without specifying coefs we'll get a template list
powerAnalysis(template_model)

# Set desired effect sizes to pass to coefs
effect_sizes <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))

# Run power analysis and look at summary
(pa <- powerAnalysis(template_model, coefs=effect_sizes, alpha=0.05))

# Try a smaller sample size in the study design
(pa2 <- powerAnalysis(template_model, coefs=effect_sizes, alpha=0.05,
                      design=list(M=100, J=2)))


## End(Not run)
```

`predict-methods` *Methods for Function predict in Package 'unmarked'*

Description

These methods return predicted values from fitted model objects.

Methods

`signature(object = "unmarkedFit")` "type" must be either 'state' or 'det'.

`signature(object = "unmarkedFitColExt")` "type" must be 'psi', 'col', 'ext', or 'det'.

`signature(object = "unmarkedFitGMM")` "type" must be 'lambda', 'psi', 'det'

`signature(object = "unmarkedFitList")` "type" depends upon the fitted models

`signature(object = "unmarkedRanef")` Use this method to generate the empirical Bayes posterior predictive distribution for functions of the random variables (latent abundance or occurrence).

In addition to the output object from `ranef`, you must also supply a custom function to argument `func`. The function must take as input a matrix with dimensions M x T, where M is the number of sites and T is the number of primary periods (T=1 for single-season models). The output of this function should be a vector or matrix containing the derived parameters of interest.

You may also manually set the number of draws from the posterior predictive distribution with argument `nsims`; the default is 100.

The output of `predict` will be a vector or array with one more dimension than the output of the function supplied `func`, corresponding to the number of draws requested `nsims`. For example, if `func` outputs a scalar, the output of `predict` will be a vector with length equal to `nsims`. If `func` outputs a 3x2 matrix, the output of `predict` will be an array with dimensions 3x2x`nsims`. See `ranef` for an example.

Alternatively, you can use the `posteriorSamples` function on the `ranef` output object to obtain the full posterior predictive distribution. This is useful if you are having trouble designing your custom function or if you want to obtain multiple different derived parameters from the same posterior predictive distribution.

`randomTerms` *Extract estimates of random effect terms*

Description

Extract estimates and summary statistics of random effect terms from an `unmarkedFit` model or an `unmarkedEstimate`.

Usage

```
## S4 method for signature 'unmarkedEstimate'
randomTerms(object, level=0.95, ...)
## S4 method for signature 'unmarkedFit'
randomTerms(object, type, level=0.95, ...)
```

Arguments

`object`	An object inheriting class `unmarkedEstimate` or `unmarkedFit`
`level`	Significance level to use for confidence interval
`type`	If provided, return only random effect terms from the chosen submodel type (as a character string)
`...`	Other arguments

Value

`data.frame` containing estimates, SEs, and confidence intervals for random effect terms in the model.

Author(s)

Ken Kellner <contact@kenkellner.com>

`ranef-methods` *Methods for Function* `ranef` *in Package* **unmarked**

Description

Estimate posterior distributions of the random variables (latent abundance or occurrence) using empirical Bayes methods. These methods return an object storing the posterior distributions of the latent variables at each site, and for each year (primary period) in the case of open population models. See unmarkedRanef-class for methods used to manipulate the returned object.

Methods

`signature(object = "unmarkedFitOccu")` Computes the conditional distribution of occurrence given the data and the estimates of the fixed effects, $Pr(z_i = 1|y_{ij}, \hat{\psi}_i, \hat{p}_{ij})$

`signature(object = "unmarkedFitOccuRN")` Computes the conditional abundance distribution given the data and the estimates of the fixed effects, $Pr(N_i = k|y_{ij}, \hat{\psi}_i, \hat{r}_{ij})k = 0, 1, \ldots, K$

`signature(object = "unmarkedFitPCount")` $Pr(N_i = k|y_{ij}, \hat{\lambda}_i, \hat{p}_{ij})k = 0, 1, \ldots, K$

`signature(object = "unmarkedFitMPois")` $Pr(N_i = k|y_{ij}, \hat{\lambda}_i, \hat{p}_{ij})k = 0, 1, \ldots, K$

`signature(object = "unmarkedFitDS")` $Pr(N_i = k|y_{i,1:J}, \hat{\lambda}_i, \hat{\sigma}_i)k = 0, 1, \ldots, K$

`signature(object = "unmarkedFitGMM")` $Pr(M_i = k|y_{i,1:J,t}, \hat{\lambda}_i, \hat{\phi}_{it}, \hat{p}_{ijt})k = 0, 1, \ldots, K$

`signature(object = "unmarkedFitGDS")` $Pr(M_i = k|y_{i,1:J,t}, \hat{\lambda}_i, \hat{\phi}_{it}, \hat{\sigma}_{it})k = 0, 1, \ldots, K$

`signature(object = "unmarkedFitColExt")` $Pr(z_{it} = 1|y_{ijt}, \hat{\psi}_i, \hat{\gamma}_{it}, \hat{\epsilon}_{it}, \hat{p}_{ijt})$

`signature(object = "unmarkedFitPCO")` $Pr(N_{it} = k|y_{ijt}, \hat{\lambda}_i, \hat{\gamma}_{it}, \hat{\omega}_{it}, \hat{\iota}_{it}, \hat{p}_{ijt})k = 0, 1, ..., K$

Warning

Empirical Bayes methods can underestimate the variance of the posterior distribution because they do not account for uncertainty in the hyperparameters (lambda or psi). Eventually, we hope to add methods to account for the uncertainty of the hyperparameters.

Note also that the posterior mode appears to exhibit some bias as an estimator or abundance. Consider using the posterior mean instead, even though it will not be an integer in general. More simulation studies are needed to evaluate the performance of empirical Bayes methods for these models.

Note

From Carlin and Louis (1996): "... the Bayesian approach to inference depends on a prior distribution for the model parameters. This prior can depend on unknown parameters which in turn may follow some second-stage prior. This sequence of parameters and priors consitutes a hierarchical model. The hierarchy must stop at some point, with all remaining prior parameters assumed known. Rather than make this assumption, the basic empirical Bayes approach uses the observed data to estimate these final stage parameters (or to estimate the Bayes rule), and proceeds as in a standard Bayesian analysis."

Author(s)

Richard Chandler <rbchan@uga.edu>

References

Laird, N.M. and T.A. Louis. 1987. Empirical Bayes confidence intervals based on bootstrap samples. Journal of the American Statistical Association 82:739–750.

Carlin, B.P and T.A Louis. 1996. Bayes and Empirical Bayes Methods for Data Analysis. Chapman and Hall/CRC.

Royle, J.A and R.M. Dorazio. 2008. Hierarchical Modeling and Inference in Ecology. Academic Press.

See Also

unmarkedRanef-class

Examples

```
# Simulate data under N-mixture model
set.seed(4564)
R <- 20
J <- 5
N <- rpois(R, 10)
y <- matrix(NA, R, J)
y[] <- rbinom(R*J, N, 0.5)

# Fit model
umf <- unmarkedFramePCount(y=y)
fm <- pcount(~1 ~1, umf, K=50)

# Estimates of conditional abundance distribution at each site
(re <- ranef(fm))
# Best Unbiased Predictors
bup(re, stat="mean")          # Posterior mean
bup(re, stat="mode")          # Posterior mode
confint(re, level=0.9) # 90% CI

# Plots
plot(re, subset=site %in% c(1:10), layout=c(5, 2), xlim=c(-1,20))
```

```
# Compare estimates to truth
sum(N)
sum(bup(re))

# Extract all values in convenient formats
post.df <- as(re, "data.frame")
head(post.df)
post.arr <- as(re, "array")

#Generate posterior predictive distribution for a function
#of random variables using predict()

#First, create a function that operates on a vector of
#length M (if you fit a single-season model) or a matrix of
#dimensions MxT (if a dynamic model), where
#M = nsites and T = n primary periods
#Our function will generate mean abundance for sites 1-10 and sites 11-20
myfunc <- function(x){ #x will be length 20 since M=20

  #Mean of first 10 sites
  group1 <- mean(x[1:10])
  #Mean of sites 11-20
  group2 <- mean(x[11:20])

  #Naming elements of the output is optional but helpful
  return(c(group1=group1, group2=group2))

}

#Get 100 samples of the values calculated in your function
(pr <- predict(re, func=myfunc, nsims=100))

#Summarize posterior
data.frame(mean=rowMeans(pr),
           se=apply(pr, 1, stats::sd),
           lower=apply(pr, 1, stats::quantile, 0.025),
           upper=apply(pr, 1, stats::quantile, 0.975))

#Alternatively, you can return the posterior predictive distribution
#and run operations on it separately
(ppd <- posteriorSamples(re, nsims=100))
```

SE-methods *Methods for Function SE in Package 'unmarked'*

Description

Extract standard errors of parameter estimates from a fitted model.

Methods

obj = "linCombOrBackTrans" A model prediction

obj = "unmarkedEstimate" See unmarkedEstimate-class

obj = "unmarkedFit" A fitted model

---

shinyPower *Launch a Shiny app to help with power analysis*

Description

Launch a Shiny app to test power under various scenarios. Requires the Shiny package to be installed.

Usage

```
shinyPower(object, ...)
```

Arguments

object	A template unmarkedFit object; see documentation for powerAnalysis for details on how to create this
...	Currently ignored

Value

No return value, called for its side effects.

---

sight2perpdist *Convert sight distance and sight angle to perpendicular distance.*

Description

When distance data are collected on line transects using sight distances and sight angles, they need to be converted to perpendicular distances before analysis.

Usage

```
sight2perpdist(sightdist, sightangle)
```

Arguments

sightdist	Distance from observer
sightangle	Angle from center line. In degrees between 0 and 180.

Value

Perpendicular distance

See Also

distsamp

Examples

```
round(sight2perpdist(10, c(0, 45, 90, 135, 180)))
```

---

sigma *Extract estimates of random effect standard deviations*

Description

Extract estimates and summary statistics of random effect standard deviations from an unmarkedFit model or an unmarkedEstimate.

Usage

```
## S4 method for signature 'unmarkedEstimate'
sigma(object, level=0.95, ...)
## S4 method for signature 'unmarkedFit'
sigma(object, type, level=0.95, ...)
```

Arguments

object	An object inheriting class unmarkedEstimate or unmarkedFit
level	Significance level to use for confidence interval
type	If provided, return only random effect SDs from the chosen submodel type (as a character string)
...	Other arguments

Value

data.frame containing estimates, SEs, and confidence intervals for random effect standard deviations in the model.

Author(s)

Ken Kellner <contact@kenkellner.com>

simulate-methods *Methods for Function simulate in Package 'unmarked'*

Description

Simulate data from a fitted model.

Usage

```
## S4 method for signature 'unmarkedFitColExt'
simulate(object, nsim, seed, na.rm)
## S4 method for signature 'unmarkedFitDS'
simulate(object, nsim, seed, na.rm)
## S4 method for signature 'unmarkedFitMPois'
simulate(object, nsim, seed, na.rm)
## S4 method for signature 'unmarkedFitOccu'
simulate(object, nsim, seed, na.rm)
## S4 method for signature 'unmarkedFitOccuRN'
simulate(object, nsim, seed, na.rm)
## S4 method for signature 'unmarkedFitPCount'
simulate(object, nsim, seed, na.rm)
## S4 method for signature 'character'
simulate(object, nsim=1, seed=NULL, formulas, coefs=NULL,
  design, guide=NULL, ...)
```

Arguments

object	Fitted model of appropriate S4 class
nsim	Number of simulations
seed	Seed for random number generator. Not currently implemented
na.rm	Logical, should missing values be removed?
formulas	A named list of formulas, one per submodel (e.g. a formula for occupancy "state" and a formula for detection "det"). To get the correct submodel names for a given model, fit an example for that model, and then call names(fitted_model)
coefs	A named list of vectors of coefficients associated with the regression intercepts and slopes for each submodel. List should be named as with formulas above. Each element of the list should be a named vector, where the names correspond to the names of the parameters in the model (intercept and covariates). If you are not sure how to structure this list, just run simulate with coefs=NULL; this will generate a template list you can copy and fill in.
design	A named list of components of the study design. Must include at least M, the number of sites, and J the number of observations per site. If you are fitting a model with multiple primary periods you must also provide T, the number of primary periods.

guide — An optional list defining the format (continuous or categorical/factor) and distribution, if continuous, of covariates you want to simulate. By default all covariates are simulated from a standard normal. See example below for an example of how to specify entries in the guide list.

... — Additional arguments that are needed to fully specify the simulated dataset for a particular model. For example, mixture for pcount models or keyfun for distsamp models.

Methods

object = "unmarkedFitColExt" A model fit by colext

object = "unmarkedFitDS" A model fit by distsamp

object = "unmarkedFitMPois" A model fit by multinomPois

object = "unmarkedFitOccu" A model fit by occu

object = "unmarkedFitOccuRN" A model fit by occuRN

object = "unmarkedFitPCount" A model fit by pcount

object = "character" An unmarkedFrame of the appropriate type

Examples

```
## Not run:

# Simulation of an occupancy dataset from scratch

# Formulas for each submodel
# occupancy is a function of elevation, detection is intercept-only
forms <- list(state=~elev, det=~1)

# Specify list of coefficients - there must be a value for each
# covariate plus an intercept for each submodel
coefs <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))

# Study design
design <- list(M=300, J=8) # 300 sites, 8 occasions per site

# If we don't specify coefs, unmarked will generate a template you can copy and use
simulate("occu", formulas=forms, design=design)

# Generate unmarkedFrameOccu
occu_umf <- simulate("occu", formulas=forms, coefs=coefs, design=design)
head(occu_umf) # note one covariate, elev

# What if we wanted to add a categorical/factor covariate or
# customize the distribution of elev?
# Use the guide argument

# Updated formulas with new covariate
forms2 <- list(state=~elev+landcover, det=~1)
```

```
# Guide
# landcover is factor, you must provide the levels
guide <- list(landcover=factor(levels=c("forest","grass")),
              elev=list(dist=rnorm, mean=2, sd=0.5)) # custom distribution

# Updated coefficients list
coefs2 <- list(state=c(intercept=0, elev=-0.4, landcovergrass=0.2), det=c(intercept=0))

# Simulate new dataset
head(simulate("occu", formulas=forms2, coefs=coefs2, design=design, guide=guide))
# Note new categorical covariate

# For some models you may want to specify other arguments, such as 'mixture'
# for pcount or 'keyfun' for distsamp
# See the documentation for the associated fitting function and unmarkedFrame
# for what arguments are possible to include for a given model
head(simulate("pcount", formulas=forms, coefs=coefs, design=design, mixture="NB"))

## End(Not run)
```

SSE *Compute Sum of Squared Residuals for a Model Fit.*

Description

Compute the sum of squared residuals for an unmarked fit object. This is useful for a parboot.

Usage

```
SSE(fit, ...)
```

Arguments

fit	An unmarked fit object.
...	Additional arguments to be passed to statistic

Value

A numeric value for the models SSE.

See Also

parboot

`Switzerland` *Swiss landscape data*

Description

Spatially-referenced data on elevation, forest cover, and water at a 1km-sq resolution.

Usage

```
data(Switzerland)
```

Format

A data frame with 42275 observations on the following 5 variables.

`x` Easting (m)

`y` Northing (m)

`elevation` a numeric vector (m)

`forest` a numeric vector (percent cover)

`water` a numeric vector (percent cover)

Details

Forest and water coverage (in percent area) was computed using the 1992-97 landcover dataset of the Swiss Federal Statistical Office (http://www.bfs.admin.ch). Median elevation (in metres) was computed using a median aggregation of the digital elevation model of the Swiss Federal Statistical Office.

x and y are the coordinates of the center of each 1km2 pixel.

The coordinate reference system intentionally not specified.

These data can only be used for non-profit projects. Otherwise, written permission must be obtained from the Swiss Federal Statistical Office

Source

Swiss Federal Statistical Office (http://www.bfs.admin.ch)

Examples

```
library(lattice)
data(Switzerland)
str(Switzerland)

levelplot(elevation ~ x + y, Switzerland, aspect="iso",
    col.regions=terrain.colors(100))

## Not run:
```

```
library(raster)
el.r <- rasterFromXYZ(Switzerland[,c("x","y","elevation")], crs =
"+proj=somerc +lat_0=46.95240555555556 +lon_0=7.439583333333333
+k_0=1 +x_0=600000 +y_0=200000 +ellps=bessel
+towgs84=674.374,15.056,405.346,0,0,0,0 +units=m +no_defs")
plot(el.r)
spplot(el.r)

## End(Not run)
```

---

unmarkedEstimate-class

Class "unmarkedEstimate"

Description

Contains parameter estimates, covariance matrix, and metadata

Objects from the Class

Creating these objects is done internally not by users.

Slots

`name`: Object of class `"character"` storing parameter names

`short.name`: Object of class `"character"` storing abbreviated parameter names

`estimates`: Object of class `"numeric"`

`covMat`: Object of class `"matrix"`

`covMatBS`: Object of class `"matrix"`

`fixed`: Object of class `"numeric"`

`invlink`: Object of class `"character"`

`invlinkGrad`: Object of class `"character"`

`randomVarInfo`: Object of class `"list"`

Methods

backTransform `signature(obj = "unmarkedEstimate")`

coef `signature(object = "unmarkedEstimate")`

confint `signature(object = "unmarkedEstimate")`

linearComb `signature(obj = "unmarkedEstimate",coefficients = "matrixOrVector")`

SE `signature(obj = "unmarkedEstimate")`

show `signature(object = "unmarkedEstimate")`

vcov `signature(object = "unmarkedEstimate")`

Note

These methods are typically called within a call to a method for unmarkedFit-class

Examples

```
showClass("unmarkedEstimate")
```

---

unmarkedEstimateList-class *Class "unmarkedEstimateList"*

Description

Class to hold multiple unmarkedEstimates in an unmarkedFit

Slots

estimates: A "list" of models.

---

unmarkedFit-class *Class "unmarkedFit"*

Description

Contains fitted model information which can be manipulated or extracted using the methods described below.

Slots

fitType: Object of class "character"

call: Object of class "call"

formula: Object of class "formula"

data: Object of class "unmarkedFrame"

sitesRemoved: Object of class "numeric"

estimates: Object of class "unmarkedEstimateList"

AIC: Object of class "numeric"

opt: Object of class "list" containing results from optim

negLogLike: Object of class "numeric"

nllFun: Object of class "function"

knownOcc: unmarkedFitOccu only: sites known to be occupied

K: unmarkedFitPCount only: upper bound used in integration

mixture: unmarkedFitPCount only: Mixing distribution

keyfun: unmarkedFitDS only: detection function used by distsamp

unitsOut: unmarkedFitDS only: density units

Methods

[`signature(x = "unmarkedFit", i = "ANY", j = "ANY", drop = "ANY")`: extract one of names(obj), eg 'state' or 'det'

backTransform `signature(obj = "unmarkedFit")`: back-transform parameters to original scale when no covariate effects are modeled

coef `signature(object = "unmarkedFit")`: returns parameter estimates. type can be one of names(obj), eg 'state' or 'det'. If altNames=TRUE estimate names are more specific.

confint `signature(object = "unmarkedFit")`: Returns confidence intervals. Must specify type and method (either "normal" or "profile")

fitted `signature(object = "unmarkedFit")`: returns expected values of Y

getData `signature(object = "unmarkedFit")`: extracts data

getP `signature(object = "unmarkedFit")`: calculates and extracts expected detection probabilities

getFP `signature(object = "unmarkedFit")`: calculates and extracts expected false positive detection probabilities

getB `signature(object = "unmarkedFit")`: calculates and extracts expected probabilities a true positive detection was classified as certain

hessian `signature(object = "unmarkedFit")`: Returns hessian matrix

linearComb `signature(obj = "unmarkedFit", coefficients = "matrixOrVector")`: Returns estimate and SE on original scale when covariates are present

mle `signature(object = "unmarkedFit")`: Same as coef(fit)?

names `signature(x = "unmarkedFit")`: Names of parameter levels

nllFun `signature(object = "unmarkedFit")`: returns negative log-likelihood used to estimate parameters

parboot `signature(object = "unmarkedFit")`: Parametric bootstrapping method to assess goodness-of-fit

plot `signature(x = "unmarkedFit", y = "missing")`: Plots expected vs. observed values

predict `signature(object = "unmarkedFit")`: Returns predictions and standard errors for original data or for covariates in a new data.frame

profile `signature(fitted = "unmarkedFit")`: used by confint method='profile'

residuals `signature(object = "unmarkedFit")`: returns residuals

sampleSize `signature(object = "unmarkedFit")`: returns number of sites in sample

SE `signature(obj = "unmarkedFit")`: returns standard errors

show `signature(object = "unmarkedFit")`: concise results

summary `signature(object = "unmarkedFit")`: results with more details

update `signature(object = "unmarkedFit")`: refit model with changes to one or more arguments

vcov `signature(object = "unmarkedFit")`: returns variance-covariance matrix

smoothed `signature(object="unmarkedFitColExt")`: Returns the smoothed trajectory from a colonization-extinction model fit. Takes additional logical argument mean which specifies whether or not to return the average over sites.

projected `signature(object="unmarkedFitColExt")`: Returns the projected trajectory from a colonization-extinction model fit. Takes additional logical argument mean which specifies whether or not to return the average over sites.

logLik `signature(object="unmarkedFit")`: Returns the log-likelihood.

LRT `signature(m1="unmarkedFit", m2="unmarkedFit")`: Returns the chi-squared statistic, degrees-of-freedom, and p-value from a Likelihood Ratio Test.

Note

This is a superclass with child classes for each fit type

Examples

```
showClass("unmarkedFit")

# Format removal data for multinomPois
data(ovendata)
ovenFrame <- unmarkedFrameMPois(y = ovendata.list$data,
siteCovs = as.data.frame(scale(ovendata.list$covariates[,-1])),
type = "removal")

# Fit a couple of models
(fm1 <- multinomPois(~ 1 ~ ufc + trba, ovenFrame))
summary(fm1)

# Apply a bunch of methods to the fitted model

# Look at the different parameter types
names(fm1)
fm1['state']
fm1['det']

# Coefficients from abundance part of the model
coef(fm1, type='state')

# Variance-covariance matrix
vcov(fm1, type='state')

# Confidence intervals using profiled likelihood
confint(fm1, type='state', method='profile')

# Expected values
fitted(fm1)

# Original data
getData(fm1)

# Detection probabilities
getP(fm1)

# log-likelihood
logLik(fm1)
```

```
# Back-transform detection probability to original scale
# backTransform only works on models with no covariates or
#     in conjunction with linearComb (next example)
backTransform(fm1, type ='det')

# Predicted abundance at specified covariate values
(lc <- linearComb(fm1, c(Int = 1, ufc = 0, trba = 0), type='state'))
backTransform(lc)

# Assess goodness-of-fit
parboot(fm1)
plot(fm1)

# Predict abundance at specified covariate values.
newdat <- data.frame(ufc = 0, trba = seq(-1, 1, length=10))
predict(fm1, type='state', newdata=newdat)

# Number of sites in the sample
sampleSize(fm1)

# Fit a new model without covariates
(fmNull <- update(fm1, formula = ~1 ~1))

# Likelihood ratio test
LRT(fm1, fmNull)
```

`unmarkedFitList-class` *Class "unmarkedFitList"*

Description

Class to hold multiple fitted models from one of unmarked's fitting functions

Objects from the Class

Objects can be created by using the `fitList` function.

Slots

`fits`: A `"list"` of models.

Methods

coef `signature(object = "unmarkedFitList")`: Extract coefficients

SE `signature(object = "unmarkedFitList")`: Extract standard errors

modSel `signature(object = "unmarkedFitList")`: Model selection

predict `signature(object = "unmarkedFitList")`: Model-averaged prediction

Note

Model-averaging regression coefficients is intentionally not implemented.

See Also

fitList, unmarkedFit

Examples

```
showClass("unmarkedFitList")

data(linetran)
(dbreaksLine <- c(0, 5, 10, 15, 20))
lengths <- linetran$Length * 1000

ltUMF <- with(linetran, {
unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
siteCovs = data.frame(Length, area, habitat), dist.breaks = dbreaksLine,
tlength = lengths, survey = "line", unitsIn = "m")
})

fm1 <- distsamp(~ 1 ~1, ltUMF)
fm2 <- distsamp(~ area ~1, ltUMF)
fm3 <- distsamp( ~ 1 ~area, ltUMF)

fl <- fitList(Null=fm1, A.=fm2, .A=fm3)
fl

coef(fl)
SE(fl)

ms <- modSel(fl, nullmod="Null")
ms
```

---

unmarkedFrame *Create an unmarkedFrame, or one of its child classes.*

Description

Constructor for unmarkedFrames.

Usage

```
unmarkedFrame(y, siteCovs=NULL, obsCovs=NULL, mapInfo, obsToY)
```

Arguments

y	An MxJ matrix of the observed measured data, where M is the number of sites and J is the maximum number of observations per site.
siteCovs	A data.frame of covariates that vary at the site level. This should have M rows and one column per covariate
obsCovs	Either a named list of data.frames of covariates that vary within sites, or a data.frame with MxJ rows in site-major order.
obsToY	optional matrix specifying relationship between observation-level covariates and response matrix
mapInfo	geographic coordinate information. Currently ignored.

Details

unmarkedFrame is the S4 class that holds data structures to be passed to the model-fitting functions in unmarked.

An unmarkedFrame contains the observations (y), covariates measured at the observation level (obsCovs), and covariates measured at the site level (siteCovs). For a data set with M sites and J observations at each site, y is an M x J matrix. obsCovs and siteCovs are both data frames (see data.frame). siteCovs has M rows so that each row contains the covariates for the corresponding sites. obsCovs has M*obsNum rows so that each covariates is ordered by site first, then observation number. Missing values are coded with NA in any of y, siteCovs, or obsCovs.

Additionally, unmarkedFrames contain metadata: obsToY, mapInfo. obsToY is a matrix describing relationship between response matrix and observation-level covariates. Generally this does not need to be supplied by the user; however, it may be needed when using multinomPois. For example, double observer sampling, y has 3 columns corresponding the observer 1, observer 2, and both, but there were only two independent observations. In this situation, y has 3 columns, but obsToY must be specified.

Several child classes of unmarkedFrame require addional metadata. For example, unmarkedFrameDS is used to organize distsance sampling data for the distsamp function, and it has arguments dist.breaks, tlength, survey, and unitsIn, which specify the distance interval cut points, transect lengths, "line" or "point" transect, and units of measure, respectively.

All site-level covariates are automatically copied to obsCovs so that site level covariates are available at the observation level.

Value

an unmarkedFrame object

See Also

unmarkedFrame-class, unmarkedFrameOccu, unmarkedFramePCount, unmarkedFrameDS

Examples

```
# Set up data for pcount()
```

```
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
obsCovs = mallard.obs)
summary(mallardUMF)


# Set up data for occu()
data(frogs)
pferUMF <- unmarkedFrameOccu(pfer.bin)


# Set up data for distsamp()
data(linetran)
ltUMF <- with(linetran, {
unmarkedFrameDS(y = cbind(dc1, dc2, dc3, dc4),
siteCovs = data.frame(Length, area, habitat),
dist.breaks = c(0, 5, 10, 15, 20),
tlength = linetran$Length * 1000, survey = "line", unitsIn = "m")
})
summary(ltUMF)


# Set up data for multinomPois()
data(ovendata)
ovenFrame <- unmarkedFrameMPois(ovendata.list$data,
siteCovs=as.data.frame(scale(ovendata.list$covariates[,-1])),
type = "removal")
summary(ovenFrame)


## Not run:
# Set up data for colext()
frogUMF <- formatMult(masspcru)
summary(frogUMF)

## End(Not run)
```

`unmarkedFrame-class` *Class "unmarkedFrame"*

Description

Methods for manipulating, summarizing and viewing unmarkedFrames

Objects from the Class

Objects can be created by calls to the constructor function `unmarkedFrame`. These objects are passed to the data argument of the fitting functions.

Slots

y: Object of class "matrix"

obsCovs: Object of class "optionalDataFrame"

siteCovs: Object of class "optionalDataFrame"

mapInfo: Object of class "optionalMapInfo"

obsToY: Object of class "optionalMatrix"

Methods

[signature(x = "unmarkedFrame", i = "numeric", j = "missing", drop = "missing"): ...

[signature(x = "unmarkedFrame", i = "numeric", j = "numeric", drop = "missing"): ...

[signature(x = "unmarkedFrame", i = "missing", j = "numeric", drop = "missing"): ...

coordinates signature(object = "unmarkedFrame"): extract coordinates

getY signature(object = "unmarkedFrame"): extract y matrix

numSites signature(object = "unmarkedFrame"): extract M

numY signature(object = "unmarkedFrame"): extract ncol(y)

obsCovs signature(object = "unmarkedFrame"): extract observation-level covariates

obsCovs<- signature(object = "unmarkedFrame"): add or modify observation-level covariates

obsNum signature(object = "unmarkedFrame"): extract number of observations

obsToY signature(object = "unmarkedFrame"):

obsToY<- signature(object = "unmarkedFrame"): ...

plot signature(x = "unmarkedFrame", y = "missing"): visualize response variable. Takes additional argument panels which specifies how many panels data should be split over.

projection signature(object = "unmarkedFrame"): extract projection information

show signature(object = "unmarkedFrame"): view data as data.frame

siteCovs signature(object = "unmarkedFrame"): extract site-level covariates

siteCovs<- signature(object = "unmarkedFrame"): add or modify site-level covariates

summary signature(object = "unmarkedFrame"): summarize data

getL signature(object = "unmarkedFrameOccuCOP"): extract L

Note

This is a superclass with child classes for each fitting function.

See Also

unmarkedFrame, unmarkedFit, unmarked-package

Examples

```
# List all the child classes of unmarkedFrame
showClass("unmarkedFrame")

# Organize data for pcount()
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
obsCovs = mallard.obs)


# Vizualize it
plot(mallardUMF)

mallardUMF


# Summarize it
summary(mallardUMF)

str(mallardUMF)

numSites(mallardUMF)

numY(mallardUMF)

obsNum(mallardUMF)


# Extract components of data
getY(mallardUMF)

obsCovs(mallardUMF)
obsCovs(mallardUMF, matrices = TRUE)

siteCovs(mallardUMF)

mallardUMF[1:5,] # First 5 rows in wide format

mallardUMF[,1:2] # First 2 observations
```

`unmarkedFrameDS` *Organize data for the distance sampling model of Royle et al. (2004) fit by distsamp*

Description

Organizes count data along with the covariates and metadata. This S4 class is required by the data argument of `distsamp`

Usage

```
unmarkedFrameDS(y, siteCovs=NULL, dist.breaks, tlength, survey,
    unitsIn, mapInfo)
```

Arguments

`y`	An RxJ matrix of count data, where R is the number of sites (transects) and J is the number of distance classes.
`siteCovs`	A `data.frame` of covariates that vary at the site level. This should have R rows and one column per covariate
`dist.breaks`	vector of distance cut-points delimiting the distance classes. It must be of length J+1.
`tlength`	A vector of length R containing the trasect lengths. This is ignored when survey="point".
`survey`	Either "point" or "line" for point- and line-transects.
`unitsIn`	Either "m" or "km" defining the measurement units for *both* `dist.breaks` and `tlength`
.	
`mapInfo`	Currently ignored

Details

unmarkedFrameDS is the S4 class that holds data to be passed to the `distsamp` model-fitting function.

Value

an object of class unmarkedFrameDS

Note

If you have continuous distance data, they must be "binned" into discrete distance classes, which are delimited by dist.breaks.

References

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

See Also

`unmarkedFrame-class`, `unmarkedFrame`, `distsamp`

Examples

```
# Fake data
R <- 4 # number of sites
J <- 3 # number of distance classes

db <- c(0, 10, 20, 30) # distance break points

y <- matrix(c(
   5,4,3, # 5 detections in 0-10 distance class at this transect
   0,0,0,
   2,1,1,
   1,1,0), nrow=R, ncol=J, byrow=TRUE)
y

site.covs <- data.frame(x1=1:4, x2=factor(c('A','B','A','B')))
site.covs

umf <- unmarkedFrameDS(y=y, siteCovs=site.covs, dist.breaks=db, survey="point",
    unitsIn="m")            # organize data
umf                         # look at data
summary(umf)                # summarize
fm <- distsamp(~1 ~1, umf)  # fit a model
```

`unmarkedFrameDSO` *Create an object of class unmarkedFrameDSO that contains data used by distsampOpen.*

Description

Organizes distance sampling data and experimental design information from multiple primary periods along with associated covariates. This S4 class is required by the data argument of `distsampOpen`

Usage

```
unmarkedFrameDSO(y, siteCovs=NULL, yearlySiteCovs=NULL, numPrimary,
       primaryPeriod, dist.breaks, tlength, survey, unitsIn)
```

Arguments

`y`	An MxJT matrix of the repeated count data, where M is the number of sites (i.e., points or transects), J is the number of distance classes and T is the maximum number of primary sampling periods per site
`siteCovs`	A `data.frame` of covariates that vary at the site level. This should have M rows and one column per covariate

yearlySiteCovs	Either a named list of MxT data.frames, or a site-major data.frame with MT rows and 1 column per covariate
numPrimary	Maximum number of observed primary periods for each site
primaryPeriod	An MxJT matrix of integers indicating the primary period of each observation
dist.breaks	vector of distance cut-points delimiting the distance classes. It must be of length J+1
tlength	A vector of length R containing the transect lengths. This is ignored when survey="point"
survey	Either "point" or "line" for point- and line-transects
unitsIn	Either "m" or "km" defining the measurement units for *both* dist.breaks and tlength

Details

unmarkedFrameDSO is the S4 class that holds data to be passed to the distsampOpen model-fitting function. Unlike most unmarked functions, obsCovs cannot be supplied.

If you have continuous distance data, they must be "binned" into discrete distance classes, which are delimited by dist.breaks.

When gamma or omega are modeled using year-specific covariates, the covariate data for the final year will be ignored; however, they must be supplied.

If the time gap between primary periods is not constant, an M by T matrix of integers should be supplied using the primaryPeriod argument.

Value

an object of class unmarkedFrameDSO

See Also

unmarkedFrame-class, unmarkedFrame, distsampOpen

Examples

```
# Fake data
M <- 4 # number of sites
J <- 3 # number of distance classes
T <- 2 # number of primary periods

db <- c(0, 10, 20, 30) # distance break points

y <- matrix(c(
   5,4,3, 6,2,1, # In bin 1: 5 detections in primary period 1, 6 in period 2
   0,0,0, 0,1,0,
   2,1,1, 0,0,0,
   1,1,0, 1,1,1), nrow=M, ncol=J*T, byrow=TRUE)
y
```

```
# Primary periods of observations
# In this case there are no gaps
primPer <- matrix(as.integer(c(
    1,2,
    1,2,
    1,2,
    1,2)), nrow=M, ncol=T, byrow=TRUE)

#Site covs: M rows and 1 column per covariate
site.covs <- data.frame(x1=1:4, x2=factor(c('A','B','A','B')))
site.covs

#Yearly site covs on gamma/omega
ysc <- list(
    x3 = matrix(c(
        1,2,
        1,2,
        1,2,
        1,2), nrow=M, ncol=T, byrow=TRUE))

umf <- unmarkedFrameDSO(y=y, siteCovs=site.covs, yearlySiteCovs=ysc,
                        numPrimary=T, primaryPeriod=primPer,
                        dist.breaks=db, survey="point", unitsIn="m")

umf                         # look at data
summary(umf)                # summarize
```

`unmarkedFrameGDR` *Organize data for the combined distance and removal point-count model of Amundson et al. (2014) fit by* `gdistremoval`

Description

Organize data for the combined distance and removal point-count model of Amundson et al. (2014) fit by `gdistremoval`

Usage

```
unmarkedFrameGDR(yDistance, yRemoval, numPrimary=1, siteCovs=NULL, obsCovs=NULL,
                 yearlySiteCovs=NULL, dist.breaks, unitsIn, period.lengths=NULL)
```

Arguments

`yDistance`	An MxTJ matrix of count data, where M is the number of sites (points), T is the number of primary periods (can be 1) and J is the number of distance classes
`yRemoval`	An MxTJ matrix of count data, where M is the number of sites (points), T is the number of primary periods (can be 1) and J is the number of time removal periods

numPrimary	Number of primary periods in the dataset
siteCovs	A data.frame of covariates that vary at the site level. This should have M rows and one column per covariate
obsCovs	A data.frame of covariates that vary at the site level. This should have MxTJ rows and one column per covariate. These covariates are used only by the removal part of the model
yearlySiteCovs	A data.frame of covariates that vary by site and primary period. This should have MxT rows and one column per covariate
dist.breaks	vector of distance cut-points delimiting the distance classes. It must be of length J+1
unitsIn	Either "m" or "km" defining the measurement units for dist.breaks
period.lengths	Optional vector of time lengths of each removal period. Each value in the vector must be a positive integer, and the total length of the vector must be equal to the number of removal periods J. If this is not provided (the default), then all periods are assumed to have an equal length of 1 time unit

Details

unmarkedFrameGDR is the S4 class that holds data to be passed to the gdistremoval model-fitting function.

Value

an object of class unmarkedFrameGDR

Note

If you have continuous distance data, they must be "binned" into discrete distance classes, which are delimited by dist.breaks.

Author(s)

Ken Kellner <contact@kenkellner.com>

References

Amundson, C.L., Royle, J.A. and Handel, C.M., 2014. A hierarchical model combining distance sampling and time removal to estimate detection probability during avian point counts. The Auk 131: 476-494.

See Also

unmarkedFrame-class, unmarkedFrame, gdistremoval

unmarkedFrameMMO *Create an object of class unmarkedFrameMMO that contains data used by multmixOpen.*

Description

Organizes count data and experimental design information from multiple primary periods along with associated covariates. This S4 class is required by the data argument of multmixOpen

Usage

```
unmarkedFrameMMO(y, siteCovs=NULL, obsCovs=NULL, yearlySiteCovs=NULL,
       numPrimary, type, primaryPeriod)
```

Arguments

y	An MxJT matrix of the repeated count data, where M is the number of sites (i.e., points or transects), J is the number of distance classes and T is the maximum number of primary sampling periods per site
siteCovs	A data.frame of covariates that vary at the site level. This should have M rows and one column per covariate
obsCovs	Either a named list of data.frames of covariates that vary within sites, or a data.frame with MxJT rows in site-major order.
yearlySiteCovs	Either a named list of MxT data.frames, or a site-major data.frame with MT rows and 1 column per covariate
numPrimary	Maximum number of observed primary periods for each site
type	Either "removal" for removal sampling, "double" for standard double observer sampling, or "depDouble" for dependent double observer sampling
primaryPeriod	An MxJT matrix of integers indicating the primary period of each observation

Details

unmarkedFrameMMO is the S4 class that holds data to be passed to the multmixOpen model-fitting function.

Options for the detection process (type) include equal-interval removal sampling ("removal"), double observer sampling ("double"), or dependent double-observer sampling ("depDouble"). Note that unlike the related functions multinomPois and gmultmix, custom functions for the detection process (i.e., piFuns) are not supported. To request additional options contact the author.

When gamma or omega are modeled using year-specific covariates, the covariate data for the final year will be ignored; however, they must be supplied.

If the time gap between primary periods is not constant, an M by T matrix of integers should be supplied using the primaryPeriod argument.

Value

an object of class unmarkedFrameMMO

See Also

unmarkedFrame-class, unmarkedFrame, multmixOpen

Examples

```
#Generate some data
set.seed(123)
lambda=4; gamma=0.5; omega=0.8; p=0.5
M <- 100; T <- 5
y <- array(NA, c(M, 3, T))
N <- matrix(NA, M, T)
S <- G <- matrix(NA, M, T-1)

for(i in 1:M) {
  N[i,1] <- rpois(1, lambda)
  y[i,1,1] <- rbinom(1, N[i,1], p)    # Observe some
  Nleft1 <- N[i,1] - y[i,1,1]         # Remove them
  y[i,2,1] <- rbinom(1, Nleft1, p)   # ...
  Nleft2 <- Nleft1 - y[i,2,1]
  y[i,3,1] <- rbinom(1, Nleft2, p)

  for(t in 1:(T-1)) {
    S[i,t] <- rbinom(1, N[i,t], omega)
    G[i,t] <- rpois(1, gamma)
    N[i,t+1] <- S[i,t] + G[i,t]
    y[i,1,t+1] <- rbinom(1, N[i,t+1], p)    # Observe some
    Nleft1 <- N[i,t+1] - y[i,1,t+1]         # Remove them
    y[i,2,t+1] <- rbinom(1, Nleft1, p)   # ...
    Nleft2 <- Nleft1 - y[i,2,t+1]
    y[i,3,t+1] <- rbinom(1, Nleft2, p)
  }
}
y=matrix(y, M)

#Create some random covariate data
sc <- data.frame(x1=rnorm(100))

#Create unmarked frame
umf <- unmarkedFrameMMO(y=y, numPrimary=5, siteCovs=sc, type="removal")

summary(umf)
```

`unmarkedFrameMPois` *Organize data for the multinomial-Poisson mixture model of Royle (2004) fit by multinomPois*

Description

Organizes count data along with the covariates. This S4 class is required by the data argument of `multinomPois`

Usage

```
unmarkedFrameMPois(y, siteCovs=NULL, obsCovs=NULL, type, obsToY,
    mapInfo, piFun)
```

Arguments

`y`	An RxJ matrix of count data, where R is the number of sites (transects) and J is the maximum number of observations per site.
`siteCovs`	A `data.frame` of covariates that vary at the site level. This should have R rows and one column per covariate
`obsCovs`	Either a named list of RxJ `data.frame`s or a `data.frame` with RxJ rows and one column per covariate. For the latter format, the covariates should be in site-major order.
`type`	Either "removal" for removal sampling, "double" for standard double observer sampling, or "depDouble" for dependent double observer sampling. If this argument not specified, the user must provide an `obsToY` matrix. See details.
`obsToY`	A matrix describing the relationship between `obsCovs` and `y`. This is necessary because under some sampling designs the dimensions of `y` do not equal the dimensions of each observation level covariate. For example, in double observer sampling there are 3 observations (seen only by observer A, detected only by observer B, and detected by both), but each observation-level covariate can only have 2 columns, one for each observer. This matrix is created automatically if `type` is specified.
`mapInfo`	Currently ignored
`piFun`	Function used to compute the multinomial cell probabilities from a matrix of detection probabilities. This is created automatically if `type` is specified.

Details

unmarkedFrameMPois is the S4 class that holds data to be passed to the `multinomPois` model-fitting function.

Value

an object of class unmarkedFrameMPois

References

Royle, J. A. (2004). Generalized estimators of avian abundance from count survey data. Animal Biodiversity and Conservation, 27(1), 375-386.

See Also

unmarkedFrame-class, unmarkedFrame, multinomPois, piFuns

Examples

```
# Fake doulbe observer data
R <- 4 # number of sites
J <- 2 # number of observers

y <- matrix(c(
   1,0,3,
   0,0,0,
   2,0,1,
   0,0,2), nrow=R, ncol=J+1, byrow=TRUE)
y

site.covs <- data.frame(x1=1:4, x2=factor(c('A','B','A','B')))
site.covs

obs.covs <- list(
   x3 = matrix(c(
      -1,0,
      -2,0,
      -3,1,
       0,0),
      nrow=R, ncol=J, byrow=TRUE),
   x4 = matrix(c(
      'a','b',
      'a','b',
      'a','b',
      'a','b'),
      nrow=R, ncol=J, byrow=TRUE))
obs.covs


# Create unmarkedFrame
umf <- unmarkedFrameMPois(y=y, siteCovs=site.covs, obsCovs=obs.covs,
    type="double")

# The above is the same as:
o2y <- matrix(1, 2, 3)
pifun <- function(p)
{
    M <- nrow(p)
    pi <- matrix(NA, M, 3)
    pi[, 1] <- p[, 1] * (1 - p[, 2])
```

```
    pi[, 2] <- p[, 2] * (1 - p[, 1])
    pi[, 3] <- p[, 1] * p[, 2]
    return(pi)
}

umf <- unmarkedFrameMPois(y=y, siteCovs=site.covs, obsCovs=obs.covs,
    obsToY=o2y, piFun="pifun")


# Fit a model
fm <- multinomPois(~1 ~1, umf)
```

`unmarkedFrameOccu` *Organize data for the single season occupancy models fit by occu and occuRN*

Description

Organizes detection, non-detection data along with the covariates. This S4 class is required by the data argument of `occu` and `occuRN`

Usage

```
unmarkedFrameOccu(y, siteCovs=NULL, obsCovs=NULL, mapInfo)
```

Arguments

`y`	An RxJ matrix of the detection, non-detection data, where R is the number of sites, J is the maximum number of sampling periods per site.
`siteCovs`	A `data.frame` of covariates that vary at the site level. This should have M rows and one column per covariate
`obsCovs`	Either a named list of `data.frame`s of covariates that vary within sites, or a `data.frame` with RxJ rows in site-major order.
`mapInfo`	Currently ignored

Details

unmarkedFrameOccu is the S4 class that holds data to be passed to the `occu` and `occuRN` model-fitting function.

Value

an object of class unmarkedFrameOccu

See Also

unmarkedFrame-class, unmarkedFrame, occu, occuRN

Examples

```
# Fake data
R <- 4 # number of sites
J <- 3 # number of visits
y <- matrix(c(
   1,1,0,
   0,0,0,
   1,1,1,
   1,0,1), nrow=R, ncol=J, byrow=TRUE)
y

site.covs <- data.frame(x1=1:4, x2=factor(c('A','B','A','B')))
site.covs

obs.covs <- list(
   x3 = matrix(c(
      -1,0,1,
      -2,0,0,
      -3,1,0,
      0,0,0), nrow=R, ncol=J, byrow=TRUE),
   x4 = matrix(c(
      'a','b','c',
      'd','b','a',
      'a','a','c',
      'a','b','a'), nrow=R, ncol=J, byrow=TRUE))
obs.covs

umf <- unmarkedFrameOccu(y=y, siteCovs=site.covs,
    obsCovs=obs.covs)   # organize data
umf                     # look at data
summary(umf)            # summarize
fm <- occu(~1 ~1, umf)  # fit a model
```

---

unmarkedFrameOccuCOP *Organize data for the occupancy model using count data fit by* occuCOP

Description

Organizes count data along with the covariates. The unmarkedFrame S4 class required by the data argument of occuCOP.

Usage

```
unmarkedFrameOccuCOP(y, L, siteCovs = NULL, obsCovs = NULL)
```

Arguments

y	An MxJ matrix of the count data, where M is the number of sites, J is the maximum number of observation periods (sampling occasions, transects, discretised sessions...) per site.
L	An MxJ matrix of the length of the observation periods. For example, duration of the sampling occasion in hours, duration of the discretised session in days, or length of the transect in meters.
siteCovs	A data.frame of covariates that vary at the site level. This should have M rows and one column per covariate
obsCovs	A named list of dataframes of dimension MxJ, with one dataframe per covariate that varies between sites and observation periods

Details

unmarkedFrameOccuCOP is the unmarkedFrame S4 class that holds data to be passed to the occuCOP model-fitting function.

Value

an object of class unmarkedFrameOccuCOP

See Also

unmarkedFrame-class, unmarkedFrame, occuCOP

Examples

```
# Fake data
M <- 4 # Number of sites
J <- 3 # Number of observation periods

# Count data
(y <- matrix(
  c(1, 3, 0,
    0, 0, 0,
    2, 0, 5,
    1, NA, 0),
  nrow = M,
  ncol = J,
  byrow = TRUE
))

# Length of observation periods
(L <- matrix(
  c(1, 3, NA,
    2, 2, 2,
```

```
    1, 2, 1,
    7, 1, 3),
  nrow = M,
  ncol = J,
  byrow = TRUE
))

# Site covariates
(site.covs <- data.frame(
  "elev" = rexp(4),
  "habitat" = factor(c("forest", "forest", "grassland", "grassland"))
))

# Observation covariates (as a list)
(obs.covs.list <- list(
  "rain" = matrix(rexp(M * J), nrow = M, ncol = J),
  "wind" = matrix(
    sample(letters[1:3], replace = TRUE, size = M * J),
    nrow = M, ncol = J)
))

# Organise data in a unmarkedFrameOccuCOP object
umf <- unmarkedFrameOccuCOP(
  y = y,
  L = L,
  siteCovs = site.covs,
  obsCovs = obs.covs.list
)

# Extract L
getL(umf)

# Look at data
print(umf) # Print the whole data set
print(umf[1, 2]) # Print the data of the 1st site, 2nd observation
summary(umf) # Summarise the data set
plot(umf) # Plot the count of detection events


# L is optional, if absent, it will be replaced by a MxJ matrix of 1
unmarkedFrameOccuCOP(
  y = y,
  siteCovs = site.covs,
  obsCovs = obs.covs.list
)

# Covariates are optional
unmarkedFrameOccuCOP(y = y)
```

---

unmarkedFrameOccuFP *Organize data for the single season occupancy models fit by occuFP*

Description

Organizes detection, non-detection data along with the covariates. This S4 class is required by the data argument of occu and occuRN

Usage

```
unmarkedFrameOccuFP(y, siteCovs=NULL, obsCovs=NULL, type, mapInfo)
```

Arguments

y	An RxJ matrix of the detection, non-detection data, where R is the number of sites, J is the maximum number of sampling periods per site.
siteCovs	A data.frame of covariates that vary at the site level. This should have M rows and one column per covariate
obsCovs	Either a named list of data.frames of covariates that vary within sites, or a data.frame with RxJ rows in site-major order.
type	A vector with 3 values designating the number of occassions where data is of type 1, type 2, and type 3 - see occuFP for more details about data types.
mapInfo	Currently ignored

Details

unmarkedFrameOccuFP is the S4 class that holds data to be passed to the occu and occuRN model-fitting function.

Value

an object of class unmarkedFrameOccuFP

See Also

unmarkedFrame-class, unmarkedFrame, occuFP

Examples

```
n = 100
o = 10
o1 = 5
y = matrix(0,n,o)
p = .7
r = .5
fp = 0.05
y[1:(n*.5),(o-o1+1):o] <- rbinom((n*o1*.5),1,p)
y[1:(n*.5),1:(o-o1)] <- rbinom((o-o1)*n*.5,1,r)
y[(n*.5+1):n,(o-o1+1):o] <- rbinom((n*o1*.5),1,fp)
type <- c((o-o1),o1,0)  ### vector with the number of each data type
site <- c(rep(1,n*.5*.8),rep(0,n*.5*.2),rep(1,n*.5*.2),rep(0,n*.8*.5))
occ <- matrix(c(rep(0,n*(o-o1)),rep(1,n*o1)),n,o)
```

```
site <- data.frame(habitat = site)
occ <- list(METH = occ)

umf1 <- unmarkedFrameOccuFP(y,site,occ, type = type)

m1 <- occuFP(detformula = ~ METH, FPformula = ~1, stateformula = ~ habitat, data = umf1)
```

`unmarkedFrameOccuMS` *Organize data for the multi-state occupancy model fit by occuMS*

Description

Organizes multi-state occupancy data (currently single-season only) along with covariates. This S4 class is required by the data argument of `occuMS`

Usage

```
unmarkedFrameOccuMS(y, siteCovs=NULL, obsCovs=NULL,
                            numPrimary=1, yearlySiteCovs=NULL)
```

Arguments

`y`	An MxR matrix of multi-state occupancy data for a species, where M is the number of sites and R is the maximum number of observations per site (across all primary and secondary periods, if you have multi-season data). Values in y should be integers ranging from 0 (non-detection) to the number of total states - 1. For example, if you have 3 occupancy states, y should contain only values 0, 1, or 2.
`siteCovs`	A `data.frame` of covariates that vary at the site level. This should have M rows and one column per covariate
`obsCovs`	Either a named list of `data.frame`s of covariates that vary within sites, or a `data.frame` with MxR rows in the ordered by site-observation (if single-season) or site-primary period-observation (if multi-season).
`numPrimary`	Number of primary time periods (e.g. seasons) for the dynamic or multi-season version of the model. There should be an equal number of secondary periods in each primary period.
`yearlySiteCovs`	A data frame with one column per covariate that varies among sites and primary periods (e.g. years). It should have MxT rows where M is the number of sites and T the number of primary periods, ordered by site-primary period. These covariates only used for dynamic (multi-season) models.

Details

unmarkedFrameOccuMS is the S4 class that holds data to be passed to the `occuMS` model-fitting function.

Value

an object of class unmarkedFrameOccuMS

Author(s)

Ken Kellner <contact@kenkellner.com>

See Also

unmarkedFrame-class, unmarkedFrame, occuMS

Examples

```
# Fake data
#Parameters
N <- 100; J <- 3; S <- 3
psi <- c(0.5,0.3,0.2)
p11 <- 0.4; p12 <- 0.25; p22 <- 0.3

#Simulate state
z <- sample(0:2, N, replace=TRUE, prob=psi)

#Simulate detection
y <- matrix(0,nrow=N,ncol=J)
for (n in 1:N){
  probs <- switch(z[n]+1,
                  c(0,0,0),
                  c(1-p11,p11,0),
                  c(1-p12-p22,p12,p22))

  if(z[n]>0){
    y[n,] <- sample(0:2, J, replace=TRUE, probs)
  }
}

#Covariates
site_covs <- as.data.frame(matrix(rnorm(N*2),ncol=2)) # nrow = # of sites
obs_covs <- as.data.frame(matrix(rnorm(N*J*2),ncol=2)) # nrow = N*J

#Build unmarked frame
umf <- unmarkedFrameOccuMS(y=y,siteCovs=site_covs,obsCovs=obs_covs)

umf                     # look at data
summary(umf)            # summarize
plot(umf)               # visualize
umf@numStates           # check number of occupancy states detected
```

unmarkedFrameOccuMulti *Organize data for the multispecies occupancy model fit by occuMulti*

Description

Organizes detection, non-detection data for multiple species along with the covariates. This S4 class is required by the data argument of occuMulti

Usage

```
unmarkedFrameOccuMulti(y, siteCovs=NULL, obsCovs=NULL,
                                maxOrder, mapInfo)
```

Arguments

y	A list (optionally a named list) of length S where each element is an MxJ matrix of the detection, non-detection data for one species, where M is the number of sites, J is the maximum number of sampling periods per site, and S is the number of species in the analysis.
siteCovs	A data.frame of covariates that vary at the site level. This should have M rows and one column per covariate
obsCovs	Either a named list of data.frames of covariates that vary within sites, or a data.frame with MxJ rows in site-major order.
maxOrder	Optional; specify maximum interaction order. Defaults to number of species (all possible interactions). Reducing this value may speed up creation of unmarked frame if you aren't interested in higher-order interactions.
mapInfo	Currently ignored

Details

unmarkedFrameOccuMulti is the S4 class that holds data to be passed to the occuMulti model-fitting function.

Value

an object of class unmarkedFrameOccuMulti

Author(s)

Ken Kellner <contact@kenkellner.com>

See Also

unmarkedFrame-class, unmarkedFrame, occuMulti

Examples

```
# Fake data
S <- 3 # number of species
M <- 4 # number of sites
J <- 3 # number of visits

y <- list(matrix(rbinom(M*J,1,0.5),M,J), # species 1
          matrix(rbinom(M*J,1,0.5),M,J), # species 2
          matrix(rbinom(M*J,1,0.2),M,J)) # species 3

site.covs <- data.frame(x1=1:4, x2=factor(c('A','B','A','B')))
site.covs

umf <- unmarkedFrameOccuMulti(y=y, siteCovs=site.covs,
    obsCovs=NULL)   # organize data
umf                     # look at data
summary(umf)            # summarize
plot(umf)               # visualize
#fm <- occu(~1 ~1, umf)  # fit a model
```

---

unmarkedFrameOccuTTD *Create an unmarkedFrameOccuTTD object for the time-to-detection model fit by occuTTD*

Description

Organizes time-to-detection occupancy data along with covariates. This S4 class is required by the data argument of occuTTD

Usage

```
unmarkedFrameOccuTTD(y, surveyLength, siteCovs=NULL, obsCovs=NULL,
                         numPrimary=1, yearlySiteCovs=NULL)
```

Arguments

y	An MxR matrix of time-to-detection data for a species, where M is the number of sites and R is the maximum number of observations per site (across all primary periods and observations, if you have multi-season data). Values in y should be positive.
surveyLength	The maximum length of a survey, in the same units as y. You can provide either a single value (if all surveys had the same max length), or a matrix matching the dimensions of y (if surveys had different max lengths).
siteCovs	A data.frame of covariates that vary at the site level. This should have M rows and one column per covariate

obsCovs	Either a named list of data.frames of covariates that vary within sites, or a data.frame with MxR rows in the ordered by site-observation (if single-season) or site-primary period-observation (if multi-season).
numPrimary	Number of primary time periods (e.g. seasons) for the dynamic or multi-season version of the model. There should be an equal number of secondary periods in each primary period.
yearlySiteCovs	A data frame with one column per covariate that varies among sites and primary periods (e.g. years). It should have MxT rows where M is the number of sites and T the number of primary periods, ordered by site-primary period. These covariates only used for dynamic (multi-season) models.

Details

unmarkedFrameOccuTTD is the S4 class that holds data to be passed to the occuTTD model-fitting function.

Value

an object of class unmarkedFrameOccuTTD

Note

If the time-to-detection values in y are very large (e.g., because they are expressed as numbers of seconds) you may have issues fitting models. An easy solution is to convert your units (e.g., from seconds to decimal minutes) to keep the values as close to 0 as possible.

Author(s)

Ken Kellner <contact@kenkellner.com>

Examples

```
# For a single-season model
N <- 100 #Number of sites
psi <- 0.4 #Occupancy probability
lam <- 7 #Parameter for exponential distribution of time to detection
Tmax <- 10 #Maximum survey length

z <- rbinom(N, 1, psi) #Simulate occupancy
y <- rexp(N, 1/lam) #Simulate time to detection
y[z==0] <- Tmax
y[y>Tmax] <- Tmax

sc <- as.data.frame(matrix(rnorm(N*2),ncol=2)) #Site covs
oc <- as.data.frame(matrix(rnorm(N*2),ncol=2)) #obs covs

umf <- unmarkedFrameOccuTTD(y=y, surveyLength=Tmax, siteCovs=sc, obsCovs=oc)
```

`unmarkedFramePCO` *Create an object of class unmarkedFramePCO that contains data used by pcountOpen.*

Description

Organizes repeated count data along with the covariates and possibly the dates on which each survey was conducted. This S4 class is required by the data argument of `pcountOpen`

Usage

```
unmarkedFramePCO(y, siteCovs=NULL, obsCovs=NULL, yearlySiteCovs, mapInfo,
    numPrimary, primaryPeriod)
```

Arguments

`y`	An MxJT matrix of the repeated count data, where M is the number of sites, J is the maximum number of secondary sampling periods per site and T is the maximum number of primary sampling periods per site.
`siteCovs`	A `data.frame` of covariates that vary at the site level. This should have M rows and one column per covariate
`obsCovs`	Either a named list of `data.frame`s of covariates that vary within sites, or a `data.frame` with MxJT rows in site-major order.
`yearlySiteCovs`	Either a named list of MxT `data.frame`s, or a site-major `data.frame` with MT rows and 1 column per covariate.
`mapInfo`	Currently ignored
`numPrimary`	Maximum number of observed primary periods for each site
`primaryPeriod`	matrix of integers indicating the primary period of each survey.

Details

unmarkedFramePCO is the S4 class that holds data to be passed to the `pcountOpen` model-fitting function.

The unmarkedFramePCO class is similar to the unmarkedFramePCount class except that it contains the dates for each survey, which needs to be supplied .

Value

an object of class unmarkedFramePCO

See Also

`unmarkedFrame-class`, `unmarkedFrame`, `pcountOpen`

Examples

```
# Repeated count data with 5 primary periods and
# no secondary sampling periods (ie J==1)
y1 <- matrix(c(
    0, 2, 3, 2, 0,
    2, 2, 3, 1, 1,
    1, 1, 0, 0, 3,
    0, 0, 0, 0, 0), nrow=4, ncol=5, byrow=TRUE)

# Site-specific covariates
sc1 <- data.frame(x1 = 1:4, x2 = c('A','A','B','B'))

# Observation-specific covariates
oc1 <- list(
    x3 = matrix(1:5, nrow=4, ncol=5, byrow=TRUE),
    x4 = matrix(letters[1:5], nrow=4, ncol=5, byrow=TRUE))

# Primary periods of surveys
primaryPeriod1 <- matrix(as.integer(c(
    1, 2, 5, 7, 8,
    1, 2, 3, 4, 5,
    1, 2, 4, 5, 6,
    1, 3, 5, 6, 7)), nrow=4, ncol=5, byrow=TRUE)


# Create the unmarkedFrame
umf1 <- unmarkedFramePCO(y=y1, siteCovs=sc1, obsCovs=oc1, numPrimary=5,
    primaryPeriod=primaryPeriod1)

# Take a look
umf1
summary(umf1)







# Repeated count data with 4 primary periods and
# no 2 secondary sampling periods (ie J=2)
y2 <- matrix(c(
    0,0,  2,2,  3,2,  2,2,
    2,2,  2,1,  3,2,  1,1,
    1,0,  1,1,  0,0,  0,0,
    0,0,  0,0,  0,0,  0,0), nrow=4, ncol=8, byrow=TRUE)


# Site-specific covariates
sc2 <- data.frame(x1 = 1:4, x2 = c('A','A','B','B'))

# Observation-specific covariates
```

```
oc2 <- list(
    x3 = matrix(1:8, nrow=4, ncol=8, byrow=TRUE),
    x4 = matrix(letters[1:8], nrow=4, ncol=8, byrow=TRUE))

# Yearly-site covariates
ysc <- list(
    x5 = matrix(c(
        1,2,3,4,
        1,2,3,4,
        1,2,3,4,
        1,2,3,4), nrow=4, ncol=4, byrow=TRUE))

# Primary periods of surveys
primaryPeriod2 <- matrix(as.integer(c(
    1,2,5,7,
    1,2,3,4,
    1,2,4,5,
    1,3,5,6)), nrow=4, ncol=4, byrow=TRUE)

# Create the unmarkedFrame
umf2 <- unmarkedFramePCO(y=y2, siteCovs=sc2, obsCovs=oc2,
    yearlySiteCovs=ysc,
    numPrimary=4, primaryPeriod=primaryPeriod2)

# Take a look
umf2
summary(umf2)
```

`unmarkedFramePCount` *Organize data for the N-mixture model fit by pcount*

Description

Organizes repeated count data along with the covariates. This S4 class is required by the data argument of `pcount`

Usage

```
unmarkedFramePCount(y, siteCovs=NULL, obsCovs=NULL, mapInfo)
```

Arguments

`y`	An RxJ matrix of the repeated count data, where R is the number of sites, J is the maximum number of sampling periods per site.
`siteCovs`	A `data.frame` of covariates that vary at the site level. This should have R rows and one column per covariate

obsCovs	Either a named list of data.frames of covariates that vary within sites, or a data.frame with RxJ rows in site-major order.
mapInfo	Currently ignored

Details

unmarkedFramePCount is the S4 class that holds data to be passed to the pcount model-fitting function.

Value

an object of class unmarkedFramePCount

See Also

unmarkedFrame-class, unmarkedFrame, pcount

Examples

```
# Fake data
R <- 4 # number of sites
J <- 3 # number of visits
y <- matrix(c(
   1,2,0,
   0,0,0,
   1,1,1,
   2,2,1), nrow=R, ncol=J, byrow=TRUE)
y

site.covs <- data.frame(x1=1:4, x2=factor(c('A','B','A','B')))
site.covs

obs.covs <- list(
   x3 = matrix(c(
      -1,0,1,
      -2,0,0,
      -3,1,0,
      0,0,0), nrow=R, ncol=J, byrow=TRUE),
   x4 = matrix(c(
      'a','b','c',
      'd','b','a',
      'a','a','c',
      'a','b','a'), nrow=R, ncol=J, byrow=TRUE))
obs.covs

umf <- unmarkedFramePCount(y=y, siteCovs=site.covs,
    obsCovs=obs.covs)          # organize data
umf                            # take a l
summary(umf)                   # summarize data
fm <- pcount(~1 ~1, umf, K=10) # fit a model
```

unmarkedMultFrame *Create an unmarkedMultFrame, unmarkedFrameGMM, unmarkedFrameGDS, or unmarkedFrameGPC object*

Description

These functions construct unmarkedFrames for data collected during primary and secondary sampling periods.

Usage

```
unmarkedMultFrame(y, siteCovs, obsCovs, numPrimary, yearlySiteCovs)
unmarkedFrameGMM(y, siteCovs, obsCovs, numPrimary, yearlySiteCovs, type,
  obsToY, piFun)
unmarkedFrameGDS(y, siteCovs, numPrimary, yearlySiteCovs, dist.breaks,
  survey, unitsIn, tlength)
unmarkedFrameGPC(y, siteCovs, obsCovs, numPrimary, yearlySiteCovs)
```

Arguments

y	A matrix of the observed data.
siteCovs	Data frame of covariates that vary at the site level.
obsCovs	Data frame of covariates that vary within site-year-observation level.
numPrimary	Number of primary time periods (seasons in the multiseason model).
yearlySiteCovs	Data frame containing covariates at the site-year level.
type	Set to "removal" for constant-interval removal sampling, "double" for standard double observer sampling, or "depDouble" for dependent double observer sampling. This should be not be specified for other types of survey designs.
obsToY	A matrix specifying relationship between observation-level covariates and response matrix
piFun	A function converting an MxJ matrix of detection probabilities into an MxJ matrix of multinomial cell probabilities.
dist.breaks	see unmarkedFrameDS
survey	see unmarkedFrameDS
unitsIn	see unmarkedFrameDS
tlength	see unmarkedFrameDS

Details

unmarkedMultFrame objects are used by colext.

unmarkedFrameGMM objects are used by gmultmix.

unmarkedFrameGDS objects are used by gdistsamp.

unmarkedFrameGPC objects are used by gpcount.

For a study with M sites, T years, and a maximum of J observations per site-year, the data can be supplied in a variety of ways but are stored as follows. `y` is an $M \times TJ$ matrix, with each row corresponding to a site. `siteCovs` is a data frame with M rows. `yearlySiteCovs` is a data frame with MT rows which are in site-major, year-minor order. `obsCovs` is a data frame with MTJ rows, which are ordered by site-year-observation, so that a column of `obsCovs` corresponds to `as.vector(t(y))`, element-by-element. The number of years must be specified in `numPrimary`.

If the data are in long format, the convenience function `formatMult` is useful for creating the unmarkedMultFrame.

unmarkedFrameGMM and unmarkedFrameGDS are superclasses of unmarkedMultFrame containing information on the survey design used that resulted in multinomial outcomes. For unmarkedFrameGMM and constant-interval removal sampling, you can set type="removal" and ignore the arguments obsToY and piFun. Similarly, for double-observer sampling, setting type="double" or type="depDouble" will automatically create an appropiate obsToY matrix and `piFuns`. For all other situations, the type argument of unmarkedFrameGMM should be ignored and the obsToY and piFun arguments must be specified. piFun must be a function that converts an MxJ matrix of detection probabilities into an MxJ matrix of multinomial cell probabilities. obsToY is a matrix describing how the obsCovs relate to the observed counts y. For further discussion and examples see the help page for `multinomPois` and `piFuns`.

unmarkedFrameGMM and unmarkedFrameGDS objects can be created from an unmarkedMultFrame using the "as" conversion method. See examples.

Value

an unmarkedMultFrame or unmarkedFrameGMM object

Note

Data used with colext, gmultmix, and gdistsamp may be collected during a single year, so yearlySiteCovs may be a misnomer is some cases.

See Also

`formatMult`, `colext`, `gmultmix`, `gpcount`

Examples

```
n <- 50   # number of sites
T <- 4    # number of primary periods
J <- 3    # number of secondary periods

site <- 1:50
```

```
years <- data.frame(matrix(rep(2010:2013, each=n), n, T))
years <- data.frame(lapply(years, as.factor))
occasions <- data.frame(matrix(rep(1:(J*T), each=n), n, J*T))

y <- matrix(0:1, n, J*T)

umf <- unmarkedMultFrame(y=y,
    siteCovs = data.frame(site=site),
    obsCovs=list(occasion=occasions),
    yearlySiteCovs=list(year=years),
    numPrimary=T)

umfGMM1 <- unmarkedFrameGMM(y=y,
    siteCovs = data.frame(site=site),
    obsCovs=list(occasion=occasions),
    yearlySiteCovs=data.frame(year=c(t(years))),
    # or: yearlySiteCovs=list(year=years),
    numPrimary=T, type="removal")


# A user-defined piFun calculating removal probs when time intervals differ.
instRemPiFun <- function(p) {
M <- nrow(p)
J <- ncol(p)
pi <- matrix(NA, M, J)
p[,1] <- pi[,1] <- 1 - (1 - p[,1])^2
p[,2] <- 1 - (1 - p[,2])^3
p[,3] <- 1 - (1 - p[,3])^5
for(i in 2:J) {
pi[,i] <- pi[, i - 1]/p[, i - 1] * (1 - p[, i - 1]) * p[, i]
}
return(pi)
}

# Associated obsToY matrix required by unmarkedFrameMPois
o2y <- diag(ncol(y))
o2y[upper.tri(o2y)] <- 1
o2y


umfGMM2 <- unmarkedFrameGMM(y=y,
    siteCovs = data.frame(site=site),
    obsCovs=list(occasion=occasions),
    yearlySiteCovs=data.frame(year=c(t(years))),
    numPrimary=T, obsToY=o2y, piFun="instRemPiFun")

str(umfGMM2)
```

unmarkedPower-methods *Methods for unmarkedPower objects*

Description

Various functions to summarize and update unmarkedPower objects

Usage

```
## S4 method for signature 'unmarkedPower'
show(object)
## S4 method for signature 'unmarkedPower'
summary(object, ...)
## S4 method for signature 'unmarkedPower'
update(object, ...)
```

Arguments

object	An object of class unmarkedPower created with the powerAnalysis function
...	For update, arguments to change in the updated power analysis. Not used by summary

Value

For show and summary, summary output is printed to the console. For update, a new powerAnalysis object corresponding to the new arguments provided.

Author(s)

Ken Kellner <contact@kenkellner.com>

See Also

powerAnalysis

Examples

```
## Not run:

# Simulate an occupancy dataset
forms <- list(state=~elev, det=~1)
coefs <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))
design <- list(M=300, J=8) # 300 sites, 8 occasions per site
occu_umf <- simulate("occu", formulas=forms, coefs=coefs, design=design)

# Fit occupancy model to simulated data
template_model <- occu(~1~elev, occu_umf)
```

```
# Set desired effect sizes to pass to coefs
effect_sizes <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))

# Run power analysis
pa <- powerAnalysis(template_model, coefs=effect_sizes, alpha=0.05)

# Look at summary
summary(pa)

# Update the analysis with new arguments
(pa2 <- update(pa, alpha=0.01))


## End(Not run)
```

---

unmarkedPowerList *Create or summarize a series of unmarked power analyses*

Description

A list of power analyses created with powerAnalysis can be combined using unmarkedPowerList, allowing comparison e.g. between different study designs/sample sizes. Additionally an unmarkedPowerList can be created directly from an unmarkedFit template model by specifying a series of study designs (number of sites, number of observations) as a data.frame. A series of methods for unmarkedPowerList objects are available including a plot method.

Usage

```
## S4 method for signature 'list'
unmarkedPowerList(object, ...)
## S4 method for signature 'unmarkedFit'
unmarkedPowerList(object, coefs, design, alpha=0.05,
                                    nulls=list(), nsim=100, parallel=FALSE, ...)
## S4 method for signature 'unmarkedPowerList'
show(object)
## S4 method for signature 'unmarkedPowerList'
summary(object, ...)
## S4 method for signature 'unmarkedPowerList,ANY'
plot(x, power=NULL, param=NULL, ...)
```

Arguments

object,x	A list of unmarkedPower objects, a fitted model inheriting class unmarkedFit, or an unmarkedPowerList object, depending on the method
coefs	A named list of effect sizes, see documentation for powerAnalysis

design	A data.frame with one row per study design to test, and at least 2 named columns: M for number of sites and J for number of observations. If you have >1 primary period a T column must also be provided
alpha	Type I error rate
nulls	If provided, a list matching the structure of coefs which defines the null hypothesis value for each parameter. By default the null is 0 for all parameters.
nsim	The number of simulations to run for each scenario/study design
parallel	If TRUE, run simulations in parallel
power	When plotting, the target power. Draws a horizontal line at a given value of power on the plot
param	When plotting, the model parameter to plot power vs. sample size for. By default this is the first parameter (which is usually an intercept, so not very interesting)
...	Not used

Value

A unmarkedPowerList object, a summary of the object in the console, or a summary plot, depending on the method

Author(s)

Ken Kellner <contact@kenkellner.com>

See Also

powerAnalysis

Examples

```
## Not run:

# Simulate an occupancy dataset and build template model
forms <- list(state=~elev, det=~1)
coefs <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))
design <- list(M=300, J=8) # 300 sites, 8 occasions per site
occu_umf <- simulate("occu", formulas=forms, coefs=coefs, design=design)
template_model <- occu(~1~elev, occu_umf)

# Generate two power analysis
effect_sizes <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))
pa <- powerAnalysis(template_model, coefs=effect_sizes, alpha=0.05)
pa2 <- powerAnalysis(template_model, effect_sizes, design=list(M=100,J=2))

# Build unmarkedPowerList and look at summary
(pl <- unmarkedPowerList(list(pa,pa2)))

# Run a bunch of power analyses for different scenarios all at once
scenarios <- expand.grid(M=c(50,200,400),
```

```
                          J=c(3,5,8))
(pl2 <- unmarkedPowerList(template_model, effect_sizes, design=scenarios, nsim=20))

# Look at summary plot for elev effect
plot(pl2, power=0.8, param='elev')


## End(Not run)
```

---

unmarkedRanef-class *Class* "unmarkedRanef"

Description

Stores the estimated posterior distributions of the latent abundance or occurrence variables.

Objects from the Class

Objects can be created by calls of the form ranef.

Slots

post: An array with nSites rows and Nmax (K+1) columns and nPrimaryPeriod slices

Methods

bup signature(object = "unmarkedRanef"): Extract the Best Unbiased Predictors (BUPs) of the latent variables (abundance or occurrence state). Either the posterior mean or median can be requested using the stat argument.

confint signature(object = "unmarkedRanef"): Compute confidence intervals.

plot signature(x = "unmarkedRanef", y = "missing"): Plot the posteriors using xyplot

show signature(object = "unmarkedRanef"): Display the modes and confidence intervals

Warnings

Empirical Bayes methods can underestimate the variance of the posterior distribution because they do not account for uncertainty in the hyperparameters (lambda or psi). Simulation studies indicate that the posterior mode can exhibit (3-5 percent) negatively bias as a point estimator of site-specific abundance. It appears to be safer to use the posterior mean even though this will not be an integer in general.

References

Laird, N.M. and T.A. Louis. 1987. Empirical Bayes confidence intervals based on bootstrap samples. Journal of the American Statistical Association 82:739–750.

Carlin, B.P and T.A Louis. 1996. Bayes and Empirical Bayes Methods for Data Analysis. Chapman and Hall/CRC.

Royle, J.A and R.M. Dorazio. 2008. Hierarchical Modeling and Inference in Ecology. Academic Press.

See Also

ranef

Examples

```
showClass("unmarkedRanef")
```

---

vcov-methods *Methods for Function vcov in Package 'unmarked'*

Description

Extract variance-covariance matrix from a fitted model.

Methods

object = "linCombOrBackTrans" See linearComb-methods

object = "unmarkedEstimate" See unmarkedEstimate-class

object = "unmarkedFit" A fitted model

---

vif *Compute Variance Inflation Factors for an unmarkedFit Object.*

Description

Compute the variance inflation factors (VIFs) for covariates in one level of the model (i.e., occupancy or detection). Calculation of VIFs follows the approach of function `vif` in package `car`, using the correlation matrix of fitted model parameters.

Usage

```
vif(mod, type)
```

Arguments

mod	An unmarked fit object.
type	Level of the model for which to calculate VIFs (for example, `'state'`)

Value

A named vector of variance inflation factor values for each covariate.

`[-methods` *Methods for bracket extraction [in Package 'unmarked'*

Description

Methods for bracket extraction `[` in Package 'unmarked'

Usage

```
## S4 method for signature 'unmarkedEstimateList,ANY,ANY,ANY'
x[i, j, drop]
## S4 method for signature 'unmarkedFit,ANY,ANY,ANY'
x[i, j, drop]
## S4 method for signature 'unmarkedFrame,numeric,numeric,missing'
x[i, j]
## S4 method for signature 'unmarkedFrame,list,missing,missing'
x[i, j]
## S4 method for signature 'unmarkedMultFrame,missing,numeric,missing'
x[i, j]
## S4 method for signature 'unmarkedMultFrame,numeric,missing,missing'
x[i, j]
## S4 method for signature 'unmarkedFrameGMM,numeric,missing,missing'
x[i, j]
## S4 method for signature 'unmarkedFrameGDS,numeric,missing,missing'
x[i, j]
## S4 method for signature 'unmarkedFramePCO,numeric,missing,missing'
x[i, j]
```

Arguments

`x`	Object of appropriate S4 class
`i`	Row numbers
`j`	Observation numbers (eg occasions, distance classes, etc...)
`drop`	Not currently used

Methods

x = "unmarkedEstimateList", i = "ANY", j = "ANY", drop = "ANY" Extract a unmarkedEstimate object from an unmarkedEstimateList by name (either 'det' or 'state')

x = "unmarkedFit", i = "ANY", j = "ANY", drop = "ANY" Extract a unmarkedEstimate object from an unmarkedFit by name (either 'det' or 'state')

x = "unmarkedFrame", i = "missing", j = "numeric", drop = "missing" Extract observations from an unmarkedFrame.

x = "unmarkedFrame", i = "numeric", j = "missing", drop = "missing" Extract rows from an unmarkedFrame

x = "unmarkedFrame", i = "numeric", j = "numeric", drop = "missing" Extract rows and observations from an unmarkedFrame

x = "unmarkedMultFrame", i = "missing", j = "numeric", drop = "missing" Extract primary sampling periods from an unmarkedMultFrame

x = "unmarkedFrame", i = "list", j = "missing", drop = "missing" List is the index of observations to subset for each site.

x = "unmarkedMultFrame", i = "numeric", j = "missing", drop = "missing" Extract rows (sites) from an unmarkedMultFrame

x = "unmarkedGMM", i = "numeric", j = "missing", drop = "missing" Extract rows (sites) from an unmarkedFrameGMM object

x = "unmarkedGDS", i = "numeric", j = "missing", drop = "missing" Extract rows (sites) from an unmarkedFrameGDS object

x = "unmarkedPCO", i = "numeric", j = "missing", drop = "missing" Extract rows (sites) from an unmarkedFramePCO object

Examples

```
data(mallard)
mallardUMF <- unmarkedFramePCount(mallard.y, siteCovs = mallard.site,
obsCovs = mallard.obs)
summary(mallardUMF)

mallardUMF[1:5,]
mallardUMF[,1:2]
mallardUMF[1:5, 1:2]
```

Index